

Florian Kelbert

Data Usage Control for Distributed Systems

Dissertation



Technische Universität München



FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Data Usage Control for Distributed Systems

Florian Manuel Kelbert

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. Georg Carle

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Alexander Pretschner

2. Univ.-Prof. Dr. Stefan Katzenbeisser,
Technische Universität Darmstadt

Die Dissertation wurde am 24.09.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 03.01.2016 angenommen.

Acknowledgements

My thanks go to

- Prof. Pretschner. For introducing me to the academic world with all its diverse aspects: administrative matters, conferences, discussions, learning, reading, presentations, projects, reviewing, teaching, writing, pain and glory. For letting me work on an interesting thesis topic. For guidance, feedback, liberty, criticism and plaudits.
- Prof. Katzenbeisser. For providing feedback on earlier partial results of this thesis.
- all co-authors and internal reviewers. Alexander, Dominik, Enrico, Fatemeh, Hervais, Matthias, Mojdeh, Prachi, Saahil, Sebastian, Tobias. For teaching me how to collaboratively write scientific documents and how to cope best with endless discussions, reviews, corrections, rewriting, etc.
- all remaining current and former colleagues and collaborators from KIT, TUM, Fraunhofer IOSB, and Fraunhofer IESE. Benjamin, Christoph, Cornelius, Denis, Dieter, Hildegard, Florian, Franz, Johan, Kristian, Manuel, Martín, Monika, Pascal, Severin, Thomas, Traudl. For administrative support, interesting discussions, continuous feedback, cake and long nights out.
- all project partners from TU Darmstadt, Universität Freiburg, Universität Kassel, Capurro Fiek Stiftung für Informationsethik, Fraunhofer SIT, Google Germany GmbH, Deutsche Post AG, Nokia, IBM, AGT International, DFKI, Seeburger, Stadtwerke Saarlouis. For innumerable interesting and diverse experiences.
- all developers and authors of free tools and Q&A websites. For developing and maintaining software such as Eclipse, Firefox, Gimp, Gnome, Gnuplot, Inkscape, LaTeX, LibreOffice, Linux, StackExchange, Thunderbird, and many more.
- all friends and family. Which are simply too many to enumerate. For constantly reminding me, both consciously and unconsciously, in countless ways that there is more to life than work and research.

Zusammenfassung

Das Forschungsgebiet der Datennutzungskontrolle erarbeitet Lösungen zur Kontrolle der Nutzung sensibler Daten nachdem der Zugriff auf diese gewährt wurde. Entsprechende Lösungen dienen dem Schutz sensibler Geschäfts-, Militär- und Regierungsgeheimnisse, geistigen Eigentums, sowie den privaten Daten von Endanwendern. Bestehende Lösungen betrachten vielmals ausschließlich die Kontrolle sensibler Daten innerhalb einzelner Systeme. Verteilte Aspekte des Problems, d.h. die Kontrolle der Nutzung von Daten die in mehreren voneinander unabhängigen Systemen vorliegen, bleiben weitgehend unberücksichtigt.

Tatsächlich beziehen sich viele Datennutzungsanforderungen auf Daten sowie deren Nutzung in mehreren voneinander unabhängigen Systemen, z.B. „Zu jedem Zeitpunkt dürfen höchstens zwei Sachbearbeiter eine lokale Kopie dieses Vertrages besitzen“ oder „Ein Vertragsvorschlag muss von mindestens zwei Sachbearbeitern genehmigt werden“. Aus diesem Grund müssen derartige Anforderungen systemübergreifend betrachtet und durchgesetzt werden. Obwohl solche Anforderungen mit Hilfe zentraler Komponenten durchsetzbar sind, sind mit einem solchen Ansatz Probleme wie das Vorhandensein eines Single Point of Failure, sowie zu erwartende übermäßige Wartezeiten und Kommunikationskosten verbunden.

In Anbetracht dieser Herausforderungen entwickelt diese Dissertation (i) ein formales Modell für Datennutzungskontrolle in verteilten Systemen und (ii) die erste dezentrale Infrastruktur zur Durchsetzung von Datennutzungsanforderungen. Das entwickelte Modell ermöglicht die systeminterne und systemübergreifende Nachverfolgung geschützter Daten, sowie die systemübergreifende Koordination der Anforderungsdurchsetzung. Ferner entwickelt diese Dissertation formale und technische Möglichkeiten zur (a) Verteilung von Datennutzungsanforderungen an alle relevanten Komponenten, (b) Identifikation aller zur Durchsetzung einer Anforderung relevanten Komponenten, sowie (c) Identifikation von Situationen in denen die Koordination der Anforderungsdurchsetzung ohne negative Auswirkungen eingeschränkt werden kann. Die Korrektheit der entwickelten formalen Ansätze wird bewiesen.

Die Evaluierung zeigt dass die durch die Datennachverfolgung entstehenden Kosten minimal sind. Des Weiteren zeigt die Evaluierung in welchen Situationen der Einsatz der entworfenen dezentralen Infrastruktur einer zentralen Infrastruktur überlegen ist. Eine Sicherheitsanalyse identifiziert und diskutiert sicherheitsrelevante Annahmen dieser Arbeit sowie die Schwächen und Grenzen der vorgeschlagenen Lösung.

Abstract

Data usage control provides mechanisms for data owners to remain in control over how their data is used after it has been accessed. Corresponding technical solutions are thus applicable in many distinct areas such as the protection of business, military and government secrets, intellectual property, as well as private user data. However, most existing solutions focus on the enforcement of data usage control within single systems and disregard distributed aspects of data usage control, i.e. how the usage of data can be controlled once data has been shared across systems and organizations.

In fact, many data usage policies can only be enforced on a global scale, as they refer to data as well as data usage events happening within several distributed systems, e.g. “at each point in time at most two clerks might have a local copy of this contract”, or “a contract must be approved by at least two clerks before it is sent to the customer”. While such policies can intuitively be enforced using a centralized infrastructure, major drawbacks are that such solutions constitute a single point of failure and that they are expected to cause heavy communication and performance overheads.

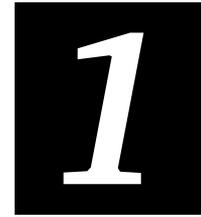
In order to address these open challenges, this dissertation contributes by providing (i) a formal distributed data usage control system model, and (ii) the first fully decentralized infrastructure for the preventive enforcement of data usage policies. More precisely, the provided model allows to track the flow of usage controlled data both within and across systems, as well as to coordinate the decision process of multiple distributed and independent decision points. To this end, this dissertation introduces formal and technical means to (a) propagate data usage policies to all relevant decision points, (b) identify all decision points that are relevant to evaluate a given policy, and (c) identify situations in which no coordination between decision points is necessary without compromising policy enforcement. Proofs of correctness of the presented formal methods are provided.

The evaluation shows that the additional overhead introduced for cross-system data flow tracking and policy propagation is negligible. Further, it reveals in which scenarios the developed decentralized enforcement infrastructure is superior to a centralized approach. A security evaluation discusses security relevant assumptions as well as shortcomings and limitations of the proposed solution.

Contents

1	Introduction	15
1.1	Gap Analysis and Research Question	16
1.2	Solution	20
1.3	Contributions	21
1.4	Threat Model	21
1.5	Running Example	22
1.6	Relevant Publications	24
2	Usage Control Models and Infrastructures	27
2.1	Formal Data Usage Control Model	27
2.2	Enforcement Infrastructure	35
2.3	Instantiation to Unix-like Systems	42
3	Distributed Data Usage Control	49
3.1	Distributed System Model	49
3.2	Cross-System Data Flow Tracking	53
3.3	Coordinating Policy Decisions Across Systems	60
4	Architecture and Implementation	69
4.1	High-level Architecture	70
4.2	Cross-System Data Flow Tracking and Policy Propagation	78
4.3	Taking Distributed Policy Decisions	84
5	Evaluation	95
5.1	Security Evaluation	95
5.2	Cross-System Data Flow Tracking and Policy Propagation	103
5.3	Distributed Policy Decisions	124
5.4	Threats to Validity	151
6	Related Work	155
6.1	Data Usage Control	155
6.2	Cross-System Data Flow Tracking and Policy Propagation	160
6.3	Distributed Policy Decisions	164
6.4	Orthogonal Approaches to Distributed Usage Control	169
6.5	Securing Data Usage Control Infrastructures	171

6.6 Digital Rights Management (DRM)	172
7 Conclusions, Discussion and Future Work	175
7.1 Conclusions	175
7.2 Critical Reflection	176
7.3 Limitations and Future Work	178
Bibliography	183
Indices	209
Index	211
List of Figures	215
List of Tables	217
List of Listings	219
Appendices	221
A Correctness of Function <i>relevant</i>	223
B Correctness of Predicate <i>Sat</i>	233
C Evaluation: Cross-System Data Flow Tracking and Policy Propagation	243



Introduction

Due to the ever increasing value of data, its continuous protection throughout its entire lifetime becomes more and more important. Solutions aiming at such data protection are applicable in many contexts, as the providers and the owners of sensitive data would like to constrain the future usage of their data. For example, businesses, military and governments aim at protecting their internal procedures, research reports and financial reports; individuals want businesses to comply with their privacy policy and constrain them from using or releasing their private data for advertisement or market research; copyright owners want end users to respect their licenses and payment models.

The research field of data usage control [166, 171, 173] addresses such advanced data protection requirements by developing models and infrastructures that allow for the specification and enforcement of data usage control policies [165, 172, 226, 227]. Such policies are capable of expressing complex propositional, temporal, cardinal and spatial constraints on how some data might or might not be used after initial access to it has been granted [79]. Additionally, policies might specify obligations that must be fulfilled before, upon, or after usage of the data [165]. Technical infrastructures then enforce these policies. Most importantly, the enforcement is performed even after access to the data has been granted, making usage control more powerful than traditional access control mechanisms, the responsibility of which usually ends once access to data was granted. While Digital Rights Management (DRM) addresses similar challenges, it can be considered a subset of data usage control that largely focuses on the protection of payment-based content [165, 173].

In order to enforce such usage control requirements, the usage of the protected data must continuously be monitored and usage control decisions must continuously be taken. For this, reference monitors are injected into the data users' computing systems [74, 91, 115, 174]. These monitors intercept relevant system events and evaluate them against the data's usage policy with the help of a decision point. Once the decision is taken, the result of which might be to allow, inhibit, delay or modify the

intercepted event, the decision is enforced. Moreover, the decision point might oblige the execution of additional events. The enforcement of policies may be preventive, meaning that policy violations never occur, or detective, meaning that policy violations can be detected in hindsight [13, 171].

When enforcing data usage control requirements, it is important to not only monitor the usage of one particular representation of some data, such as a word document, a database entry, or a Java object, but rather all of them [174]. The rationale is that the data to be protected constitutes an abstract object which materializes in different forms, formats, and versions within the computing system. For this reason, the differentiation between abstract data and its concrete representations at runtime has been introduced [74, 170]. In order to enforce data usage policies on all those representations of some data, the enforcement infrastructure must be aware of all those representations. For this, data flow tracking technologies are leveraged [174]. Essentially, they track the flow of data both within and across different layers of the computing system by monitoring data flow related system events, such as copying files or loading content from a database into a process. Consequently, all representations of some data are known to the usage control infrastructure and can be protected.

1.1 Gap Analysis and Research Question

While the problem of enforcing usage control requirements within single independent systems has been and is being researched [55, 74, 147, 170, 174, 216], these solutions fall short when it comes to today's interconnected computing systems, in which data is not only independently processed by single systems, but rather by distributed cooperating systems and applications. Even in the presence of such distributed data processing, data owners would like to enforce data usage control policies after the data has been released for means of storage, processing, and further dissemination. Usage control requirements must then be enforced on *all* systems that store and process the sensitive data. These requirements may then also refer to the storage and processing of data *across* different systems, such as "this data must not reside in more than three systems", "not more than five instances of this application may be run at the same time", or "access this data at most five times [within a certain amount of time]".

Since the enforcement of such distributed usage control requirements has not been adequately researched, many questions remain unsolved. Comprehensive solutions that generically tackle the problem of usage control enforcement in distributed systems have not been investigated, proposed and developed.

This thesis investigates distributed aspects of data usage control, essentially tackling the problem of how to enforce data usage control policies if

- (i) the protected data has representations within different distributed systems,

- (ii) the system events being constrained or obliged by data usage control policies happen within different distributed systems.

To this end, this thesis investigates three related research questions. These are identified and motivated in the following sections by providing concise analyses of related works. A more thorough investigation of and comparison with related works is presented in Chapter 6.

1.1.1 Generic Cross System Data Flow Tracking and Policy Propagation

Since data usage control policies ought to be enforced on *all* representations of some data, usage control enforcement infrastructures must have precise knowledge about *where* all these different representations reside. Since this thesis investigates data usage control in distributed systems, it is essential to understand that these representations do not necessarily remain on one single system: In the omnipresence of data networks, data is regularly being exchanged between systems using a multitude of different applications and protocols. Hence, in order for the infrastructure to be aware of all representations of some protected data at each point in time, it is essential to track the flow of data not only within but also across different connected systems. Similarly, whenever data is exchanged between systems, the corresponding data usage policies are expected to be available to those components of the infrastructure that are expected to enforce compliance with them.

In terms of dedicated data flow tracking solutions in the usage control context, most existing works focus on single systems [174, 214, 216] and “do not distinguish between various network connections but treat the whole network as a single container” [74]. While there exist some usage control solutions that are capable of bundling usage controlled data with the corresponding policies and tracking these bundles even across systems [110, 115, 125], they rely on particular applications, application-protocols and/or file types. However, such a limitation to particular technologies is inadequate in today’s heterogeneous world of data processing, in which data is continuously transformed and exchanged using a multitude of applications and protocols. This is similar for DRM solutions [1, 9, 137] (cf. Section 6.6), in which content is usually encrypted and exchanged using proprietary file formats and protocols, while usage of this content is only possible using certain proprietary applications after a digital license has been obtained.

As detailed in Section 6.2, further models and implementations for cross-system data flow tracking and policy propagation exist [53, 89, 161, 162, 220, 222, 224]. However, none of these has been designed with usage control requirements in mind nor integrated with corresponding infrastructures. For this reason, all of these solutions come with one more limitations when considered in the context of distributed data usage control enforcement, such as (i) being limited in the number of distinct data and/or policies that can be tracked [162, 220, 224], (ii) being limited to the propaga-

tion of simple labels rather than complex policies [53, 162, 220, 222, 223], (iii) relying on particular hardware [162, 220], hypervisors [223], and/or application(-protocol)s [89, 161], (iv) necessitating the adaptation of existing applications [161, 162, 222].

Ideally, and different to the above solutions, tracking of data flows across systems should be generic in the sense that it is transparent to the application layer and its protocols. As a consequence, no adaptation of existing applications would be required when integrating such technology into existing systems. Further, if usage controlled data is disseminated to different systems, then all of those systems are expected to enforce the data's policies. Consequently, complex data usage control policies must be propagated to the corresponding decision points whenever data is transferred to remote systems. Thus, the posed research question is

*How can the flow of data across different systems be tracked
in a generic and transparent manner and how can data usage
policies be propagated to the corresponding decision points?* **(RQ1)**

This thesis addresses this research question by providing corresponding models in Section 3.2, their implementation in Section 4.2, and an evaluation in Section 5.2. Section 6.2 describes related works as well as their shortcomings in detail.

1.1.2 Distributed Policy Decisions

Once the usage control infrastructure is aware of the residence of usage controlled data within different distributed systems (cf. **RQ1**), another requirement is to enforce usage control policies that are of a global scale, i.e. policies that refer to data and events thereon that happen across multiple distributed systems. In order to enforce such policies in a consistent manner across all involved systems, information about all relevant data usage events must be readily available to the corresponding decision points. Trivially, such policies can be enforced using a centralized enforcement infrastructure. This, however, comes with several drawbacks such as posing a single point of failure, privacy concerns, and the necessity for the central component to be always available [5, 30]. Intuitively, a centralized infrastructure also poses significant communication and performance overheads [5, 6, 30, 88].

Due to these drawbacks of centralized enforcement infrastructures, “one fundamental question is how to soundly and effectively distribute the monitoring process for a given global system property” [17]. Due to such distributed monitoring, however, monitors would only observe local system behaviors which in turn necessitates the communication between those monitors [17]. As detailed in Section 6.3, several models and implementations aiming at the decentralization of parts of the policy decision process have been proposed [6, 12, 13, 18, 28, 31, 39, 63, 64, 115]. However, all of these solutions come with at least one of the following limitations: (i) the decision process is not entirely distributed since there still exist some central components [39, 63, 64, 115], (ii) the data, resources and/or other objects being protected are assumed

to statically remain within one system [6, 28, 31, 64], (iii) the approach does not allow for the preventive enforcement of policies [12, 13, 18], (iv) the approach can not be deployed within commodity systems/networks [18].

Based on these considerations, this thesis tackles the research question

How can usage control policies be enforced in an effective, preventive, and decentralized manner if data, system events, and policies are distributed across different independent systems? (RQ2)

This thesis addresses this research question by providing corresponding models in Section 3.3, their implementation in Section 4.3, and an evaluation in Section 5.3. Section 6.3 describes related works as well as their shortcomings in detail.

1.1.3 Security Analysis and Provided Guarantees

Because security technology can only be deployed in a beneficial manner if both its capabilities and limitations are known to the entities aiming to protect their data, a security analysis of the proposed data usage control infrastructure ought to be performed. While such a security analysis helps to understand which guarantees are in fact provided by the developed infrastructure, it also reveals in which scenarios the sensitive data's protection is at stake. As far as such weaknesses are identified, corresponding countermeasures ought to be described and discussed.

In particular, different kinds of attackers with different a priori permissions, technical abilities, or criminal intent might try to circumvent or impair the functionality of the proposed infrastructure. Thereby, the Achilles heel of distributed data usage control infrastructures is that reference monitors must be deployed at the data consumer's site. Nevertheless, such remotely deployed components must "behave in a good manner and this manner [must be verifiable] by the policy stakeholder" [227]. Circumventing or tampering with the remotely deployed security infrastructure is thus a major concern. In terms of providing such security guarantees, much work has been carried out in the areas of DRM [1, 65, 124, 137, 196], trusted computing [167, 186], and remote attestation [156, 183, 189].

Hence, the research question posed in this thesis is

Which guarantees for distributed usage control enforcement are provided by the presented infrastructure and what are critical attack vectors that necessitate further investigation? (RQ3)

This research question is addressed by providing a security analysis in Section 5.1, a critical discussion on the developed solution in Section 7.2, and by pointing to security-related future works in Section 7.3. Further, Section 6.5 describes works that provide technical means to secure data usage control infrastructures.

Throughout this thesis the above research questions will be referred to as **RQ1**, **RQ2** and **RQ3**.

1.2 Solution

The solution provided to the above research questions is a generic, comprehensive and integrated model for cross-system data flow tracking, policy propagation, and distributed policy enforcement. In addition, an implementation of the proposed models is provided and thoroughly evaluated in terms of security as well as performance and communication overheads.

(1) As a basis to address the above research questions, Section 3.1 provides a comprehensive formal model for decentralized data usage control. It formalizes that distributed usage controlled systems run in parallel and produce independent system traces. The model describes how these distributed observations correlate and how they can be combined to mimic the behavior of classical centralized models. Thus the model allows to reason about the states of single individual systems as well as the state of the global distributed system.

(2) In terms of **RQ1**, Section 3.2 provides a generic model for cross-system data flow tracking that integrates with the comprehensive model from Section 3.1. The model is exemplarily instantiated for the Transmission Control Protocol (TCP) in Section 3.2.2 and a corresponding decentralized implementation is presented in Section 4.2. Different to existing works, the presented solution allows to track data flows between systems in a manner that is transparent and generic, i.e. independent of applications, application-protocols, and the operating system. The evaluation in Section 5.2 shows that the additional overhead imposed by cross-system data flow tracking and policy propagation is negligible.

(3) In terms of **RQ2**, Section 3.3 provides original analyses on how global policies can be efficiently and preventively enforced in a fully decentralized manner. For this, Section 3.3.1 identifies all systems potentially relevant for evaluating a given policy, while Section 3.3.2 formalizes in which situations communication between systems can safely be omitted without compromising policy enforcement. The correctness of these methods is proven in Appendices A and B. The implementation provided in Section 4.3 is the first to achieve preventive enforcement of global data usage policies in a fully decentralized manner: distributed decisions are decentrally, continuously, and consistently taken, agreed upon, and enforced. The evaluation in Section 5.3 compares this thesis' approach to traditional centralized infrastructures and reveals that the adoption of a fully decentralized infrastructure as developed in this thesis is beneficial in most cases.

(4) In terms of **RQ3**, Section 5.1.1 analyzes security relevant assumptions taken throughout this thesis and discusses why these assumptions have been made. On this basis, Section 5.1.2 analyzes how attackers might be able to use usage controlled in an uncontrolled manner and how it might be possible to impair the functioning of the usage control infrastructure such that no further (satisfactory) data usage by legitimate users is possible. Whenever appropriate, corresponding countermeasures to

the identified threats are proposed. Further existing works that address the technical protection of distributed usage control infrastructures, and which could thus be applied to this thesis' solution, are presented and discussed in Section 6.5.

1.3 Contributions

This thesis contributes by providing

- ▶ a comprehensive and generic model that serves as a basis for the enforcement of data usage control policies in distributed systems. In particular, this model allows for the explicit description and distinction of different systems, their individual behaviors, as well as their interplay. Being tailored to distributed data usage control, the model is the first to allow to reason about both independent systems as well as the distributed system they form.
- ▶ the first model and implementation to track data flows and data usage control policies across systems in a manner that is independent of particular file types, applications, application-protocols, hypervisors, or hardware. Further, the approach does not limit the amount of different data/policies that can be tracked throughout the overall distributed system and it does not necessitate any changes to the operating system and/or the applications being used.
- ▶ the first model and implementation that allow for the fully decentralized and preventive enforcement of data usage control policies that refer to data or events that are distributed across multiple systems. This is in contrast to previous works which effectively rely on central components, do not allow for preventive policy enforcement, require particular communication buses, or do not consider the fact that the protected data keeps being propagated across different systems.
- ▶ a thorough security evaluation of the proposed concepts and implementations.

1.4 Threat Model

Contents of this section have been published in [93].

The purpose of the usage control infrastructure being built within this thesis is to prevent users from using data in a way that does not comply with the corresponding policies—be the attempt intentional or unintentional. As such, potential attackers considered in this thesis are (i) end users without administrative privileges who might act intentionally or unintentionally, (ii) a man-in-the-middle between different remote entities/components of the distributed usage control infrastructure, (iii) the administrators of the involved systems, and (iv) malicious or benign data processing software including all kinds of malware. In particular, scenarios in which non-privileged end users are given ready-to-use computing systems are pervasive in business environments.

This is also the case in this thesis' running example (cf. Section 1.5), in which employees of an insurance company are provided with corresponding systems. Man-in-the-middle attacks might happen whenever data is exchanged between different components of the usage control infrastructure. Such attacks are particularly threatening if valuable data is exchanged via public networks, as is the case in our insurance scenario. The consideration of administrators as attackers is particularly interesting because they are usually able to access and control all or most aspects of the operating system, such as running updates and starting or stopping any kind of process or service. Lastly, both malicious and benign software might pose a threat. While malicious software might deliberately perform attacks, also benign software might be misconfigured or corrupted and thus lead to similar effects.

Goals of the aforementioned attackers might be to use usage controlled data without complying to the corresponding data usage policies, or to render the usage control infrastructure or the system being protected unusable.

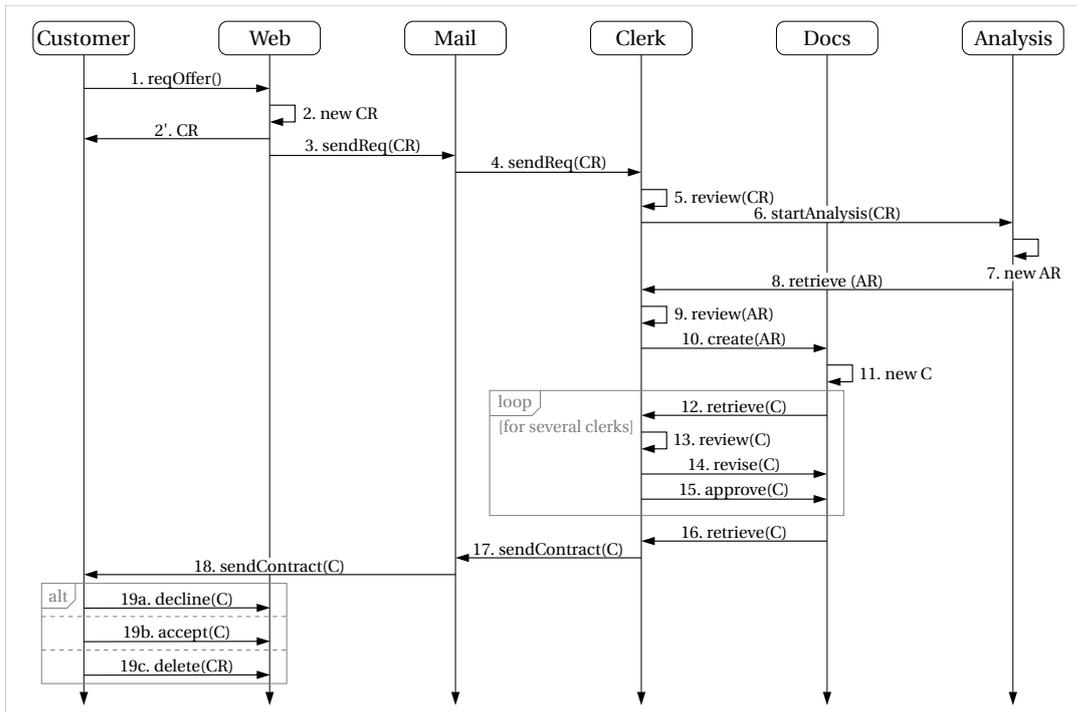
1.5 Running Example

Contents of this section have been published in [93, 95].

The concepts, implementations and evaluations presented in this thesis are illustrated along a running example in the domain of an insurance company. For the customer, the insurance company provides the ability to access several services via a web interface, such as requesting contract offers, changing, renewing or cancelling contracts, accessing bills, and issuing damage reports. Internally, the insurance company performs market research, data analysis, calculations, and financial investments. Those internal processes and analysis results constitute the insurance company's business secrets and its competitive advantage. Thus, besides the customers' personal records, also the insurance company's internal data requires protection from both intended and unintended misuse and leakage. One example use case, depicted in Figure 1.1, follows.

In order to request a contract offer, the customer fills a web form on the insurance provider's website, providing her name, address, and date of birth, as well as some more personal information depending on the type of contract being requested. By submitting the form (1), a new ContractRequest (CR) object is created (2) and the web server sends the CR to a set of clerks via the mail server (3,4). One of the clerks will then review the attached CR (5) and start an analysis job on the internal data analysis server (6), thereby creating a new AnalysisResult (AR) object (7). Once the analysis is performed, the clerk retrieves the AR (8) and performs a manual review on her workstation (9). The clerk then creates a Contract (C) object using a collaborative word processor (10,11). Once created, C might be retrieved (12), reviewed (13) and revised (14) by several clerks. After C has been approved by a predefined number of clerks (15), one of the clerks retrieves its final version (16) and sends it to the

Figure 1.1: Sequence of events in the running example.



customer via the mail server (17,18). Once the customer receives the offer, he might decline (19a) or accept (19b) the Contract. Alternatively, he might delete his initial ContractRequest altogether (19c).

Motivated from this small example, the customer's personal data flows through different systems in various formats and is viewed, stored, and processed by several systems and users. Further, there exist several data that are, directly or indirectly, derived from the data entered by the customer, such as the analysis results or the contract offer. All these different data thus originate from the customer's personal data entered on the web form and must consequently be treated as containing the customer's personal data.

Internally, the insurance company produces and maintains different data that is crucial for its success. Among others, those are results of market researches, spreadsheets for contracts and liability cases, life tables, as well as probabilities of liability cases, and predictions of the development of financial markets, society, demography, and technology. Hence this data is of utmost importance for the insurance company's success and must be kept confidential and not be leaked to any competitors.

The above scenario with the appearance of different kinds of sensitive data in multiple places and contexts would benefit from data usage control infrastructures in several ways. Example policies are

Policy 1: 'Exactly one contract offer must be sent to the customer not later than 30 days after a request for a contract offer has been received.'

Policy 2: ‘If the customer declines a contract offer, then all associated and derived data must not be used anymore.’

Policy 3: ‘Each contract must be reviewed and approved by at least two clerks.’

Policy 4: ‘At each point in time no two clerks might have a local copy of the same customer record.’

Policy 5: ‘There may be at most one ongoing edit process for each contract offer at each point in time and no editing of contracts is allowed after their final version has been archived by a manager.’

Policy 6: ‘Market research results must never leave the company unless approved by a manager.’

As can be seen from these high-level policy examples, usage control is not only capable of expressing and enforcing constraints on the usage of data, but it might also oblige certain events in order to improve service (Policy 1). Note that Policy 6 is indefinite in terms of how the data leakage happens and as such it is capable of prohibiting both intended data leakage by malicious insiders, as well as inadvertent data leakages by loyal but careless employees. Further note that all of the above policies are *global policies*, meaning that they refer to data and events that are distributed across several systems.

global policy

1.6 Relevant Publications

The results presented in this thesis are based on various research papers that have been published since starting the dissertation undertaking. This section gives an overview over those research papers. Wherever appropriate, these research papers will be referred to throughout this thesis.

The research questions presented and tackled in this dissertation have first been described in a PhD Symposium paper [92]. This paper also describes an overarching use case scenario, the considered threat model, expected contributions, as well as initial ideas on how to tackle those research challenges and the evaluation of the corresponding solutions.

Results on cross-system data flow tracking and policy propagation (**RQ1**) have been published in [94, 96, 107]. The papers present a generic model for tracking data flows across systems (Section 3.2), its instantiation for the Transmission Control Protocol (TCP) (Section 3.2.2), as well as a technical infrastructure performing cross-system data flow tracking and policy propagation (Section 4.2). Further, [94] presents an evaluation of the presented model and infrastructure in terms of security (**RQ3**, Section 5.1) and performance overhead (Section 5.2).

The problem of enforcing global data usage policies in a decentralized manner (**RQ2**) has been addressed in [93, 95]. The work in [95] presents a formal distributed data usage control model that caters to the fact that distributed systems run in parallel (Section 3.1.1). Moreover, the paper presents formal methods for identifying all

systems relevant for evaluating a given data usage policy and for identifying situations in which no coordination between systems is necessary without compromising policy enforcement (Section 3.3). Based on these results, [93] shows how these formal concepts can be implemented (Section 4.3) and presents an evaluation in terms of communication and performance overhead (Section 5.3).

While not strictly related to the research questions posed in this thesis and the corresponding solutions and contributions, further work in the area of data usage control was published in [61, 127]. In [61], data usage control technology is integrated with smart metering technology, while [127] proposes a method to reduce data flow tracking overapproximations by leveraging additional knowledge about the data's inherent structure.

2

Usage Control Models and Infrastructures

This chapter describes previous work upon which this thesis builds, namely a formal data usage control model (Section 2.1), a corresponding enforcement infrastructure (Section 2.2), and an instantiation of those concepts to Unix-like systems (Section 2.3). As such, the presented concepts are *not* a contribution of this thesis and sources are cited accordingly. Most of those concepts have also been described in papers published within the context of this thesis (cf. Section 1.6), in particular [93, 94, 95]. Verbatim quotes taken from those publications are not explicitly marked.

2.1 Formal Data Usage Control Model

This section describes a formal data usage control model from the literature [74, 79, 172, 174, 176] upon which this thesis builds. In a nutshell, data usage control policies define constraints over the set of system traces and system states. To this end, Section 2.1.1 formally defines the underlying concepts of system events and system traces, while Section 2.1.2 introduces system states in terms of a generic data flow model. On these grounds, Section 2.1.3 defines both the syntax and semantics of data usage control policies.

2.1.1 System Events and System Traces

Events \mathcal{E} are defined by a name (set \mathcal{N}) and a set of parameters, which are, in turn, defined by a name (set \mathcal{N}) and a value (set \mathcal{V}):

events

$$\mathcal{E} \subseteq \mathcal{N} \times \mathbb{P}(\mathcal{N} \times \mathcal{V})$$

For an event $e \in \mathcal{E}$, let $e.name$ denote the event's name and $e.p$ its set of parameters. An event e with n parameters $p_1, \dots, p_n \in \mathcal{N}$ and corresponding values $v_1, \dots, v_n \in \mathcal{V}$ is also written as tuple $(e.name, \{(p_1, v_1), \dots, (p_n, v_n)\})$.

Each event carries two mandatory first-class parameters, $obj, actual \in \mathcal{N}$, which are referred to using notation $e.obj$ and $e.actual$. Thereby, $e.obj$ denotes the primary object of event e , such as a file, a mail, or a database table. The value of parameter $e.actual$ is boolean, $\mathbb{B} = \{true, false\} \subseteq \mathcal{V}$: $e.actual = true$ indicates that event e has already happened, while $e.actual = false$ indicates that event e is attempted to happen. On these grounds, the set of events \mathcal{E} is divided into two disjoint subsets, *actual events* $\mathcal{E}^A = \{e \in \mathcal{E} \mid e.actual = true\}$, and *intended events* $\mathcal{E}^I = \{e \in \mathcal{E} \mid e.actual = false\}$, with $\mathcal{E}^A \cap \mathcal{E}^I = \emptyset$.

Further, the set of events \mathcal{E} is categorized into two, possibly overlapping, subsets: *data usage events* and *data flow events*. Intuitively, data usage events \mathcal{E}_U are events that enable users and applications to view, modify, and process data. Data flow events \mathcal{E}_F cause data to migrate from one representation to another, e.g. by copying some data (Section 2.1.2). Notably, an event $e \in \mathcal{E}$ might be a data usage event and a data flow event at the same time, $e \in \mathcal{E}_U \cap \mathcal{E}_F$.

For example, the event $e_r = (review, \{(obj, d), (actual, true), (role, manager)\}) \in \mathcal{E}$ denotes that some user with role *manager* actually reviews data object d . Hence, e_r is an actual event and intuitively reviewing data d is also considered using that object, hence $e_r \in \mathcal{E}^A \cap \mathcal{E}_U$. Whether e_r is also a data flow event ($e_r \in \mathcal{E}_F$) depends on implementation-level details, i.e. whether performing event e_r leads to the creation of a new representation of data d .

Event refinement. When specifying policies (Section 2.1.3), it is not useful to define all possible parameters of events whose usage ought to be constrained. Instead, one would like to specify only relevant parameters, quantifying over all unmentioned ones. Hence, $refines \subseteq \mathcal{E} \times \mathcal{E}$ defines a refinement relation on events: event $e_1 \in \mathcal{E}$ refines event $e_2 \in \mathcal{E}$ iff they have the same event name and if the parameters of e_1 are a superset of the parameters of e_2 :

$$\forall e_1, e_2 \in \mathcal{E} : e_1 \text{ refines } e_2 \iff e_1.name = e_2.name \wedge e_1.p \supseteq e_2.p$$

In particular $refines$ considers first-class parameters $obj, actual \in \mathcal{N}$.

System events \mathcal{S} are events that are observed in real systems at runtime. Different to events \mathcal{E} , system events \mathcal{S} are always maximally refined, meaning that all parameters are determined:

$$\mathcal{S} = \{e \in \mathcal{E} \mid \nexists e' \in \mathcal{E} : e' \neq e \wedge e' \text{ refines } e\}$$

In correspondence with \mathcal{E} , system events \mathcal{S} are categorized into actual system events $\mathcal{S}^A = \mathcal{E}^A \cap \mathcal{S}$, intended system events $\mathcal{S}^I = \mathcal{E}^I \cap \mathcal{S}$, data usage system events $\mathcal{S}_U = \mathcal{E}_U \cap \mathcal{S}$, and data flow system events $\mathcal{S}_F = \mathcal{E}_F \cap \mathcal{S}$. For convenience, also the set of actual data flow system events, \mathcal{S}_F^A , is defined as $\mathcal{S}_F^A = \mathcal{S}^A \cap \mathcal{S}_F$. Further, each system event is required to carry parameter $time \in \mathcal{N}$, indicating the point in time in which the event was observed: $\forall e \in \mathcal{S} : \exists r \in \mathbb{R}^{\geq 0} : (time, r) \in e.p$. Notation $e.time$ is used to access parameter value r .

Table 2.1: Example event trace spanning several timesteps.

Timestep [days]	Event
...	...
12	$(requestOffer, \{(obj, d), (time, 11.43), (customer, dave), \dots\})$ ----- $(createOffer, \{(obj, d), (time, 11.82), (clerk, john), \dots\})$
13	\emptyset
14	$(review, \{(obj, d), (time, 13.21), (clerk, mary), \dots\})$ ----- $(review, \{(obj, d), (time, 13.70), (clerk, chris), \dots\})$
15	$(sendOffer, \{(obj, d), (time, 14.37), (clerk, john), \dots\})$
...	...

Getting back to the above example, event

$e_r' = (review, \{(obj, d), (actual, true), (time, 127.3), (role, manager), (user, tom)\}) \in \mathcal{E}$ refines event e_r . Consequently, $e_r \notin \mathcal{S}$. Assuming that no other event refines e_r' , $e_r' \in \mathcal{S}$ and therefore $e_r' \in \mathcal{S}^A \cap \mathcal{S}_U$.

Traces \mathcal{T} are used to model actual system runs. For this, abstract points in time are introduced. Each abstract point in time $i \in \mathbb{N}$ represents the continuous time interval since the previous abstract point in time, i.e. the interval $(i-1, i]$. The interval between two such abstract points in time is also called a *timestep*, the necessity of which is further motivated in Section 2.2.3. Consequently, a system trace maps each abstract point in time to the set of system events that happened since the previous abstract point in time:

traces

timesteps

$\mathcal{T}: \mathbb{N} \rightarrow \mathbb{P}(\mathcal{S})$, such that

$$\forall t \in \mathcal{T}, \forall i \in \mathbb{N}, i > 0, \forall e \in t(i) : i-1 < e.time \leq i$$

Notably, traces consist of both actual and intended events, as well as data usage events and data flow events. For any trace $t \in \mathcal{T}$ it is assumed that no two events happen at exactly the same point in time: $\forall t \in \mathcal{T}, i \in \mathbb{N} : \nexists e_1, e_2 \in t(i) : e_1.time = e_2.time$. In other words, the monitors observing system events are assumed to impose a strict sequential order on the observed events.

Table 2.1 shows one short event trace within the context of the running example: After a contract has been requested by a customer, a clerk creates a corresponding offer. After the offer was reviewed by two other clerks, the offer is sent to the customer. Note how several events may be attributed to the same timestep on the basis of the events' *time* parameter.

2.1.2 Generic Data Flow Model and System States

As motivated in Section 1.5, at runtime the data to be protected by usage control policies may exist in multiple representations, and this set of representations is contin-

uously evolving. Hence, the system's *data flow state* captures which data takes which representations at which point in time. This state is maintained using a generic data flow model [74, 170, 174], which (over-)approximates the existence of data item copies in a system by capturing the flow of data within this system. State transitions are initiated by data flow events \mathcal{E}_F , which change the mapping between data and their representations.

data **Data \mathcal{D} , Containers \mathcal{C} , Identifiers \mathcal{I} .** Within this model, the set of *data* to be protected by usage control policies is denoted by \mathcal{D} . The representations containing data are called *containers* and the set of containers is denoted by \mathcal{C} , and $\mathcal{D} \cap \mathcal{C} = \emptyset$. \mathcal{D} , \mathcal{C} and special value *nil* constitute possible values for an event's *obj* parameter: events $e \in \mathcal{E}$ usually refer to the usage of some data $d \in \mathcal{D}$ (cf. Section 2.1.3), while system events $e' \in \mathcal{S}$ refer to the container $c \in \mathcal{C}$ on which they are operating. Hence, $\mathcal{D} \cup \mathcal{C} \cup \{\text{nil}\} \subseteq \mathcal{V}$ and $\forall e \in \mathcal{E}, \exists v \in \mathcal{D} \cup \mathcal{C} \cup \{\text{nil}\} : e.\text{obj} = v$. Containers are identified using *identifiers* \mathcal{I} : most system events carry parameters, the values of which refer to objects and resources being accessed by those system events. Hence, $\mathcal{I} \subseteq \mathcal{V}$.

data flow states **Data flow states Σ .** Using those definitions, the set of all possible data flow states Σ is defined as

$$\Sigma = (\mathcal{C} \rightarrow \mathbb{P}(\mathcal{D})) \times (\mathcal{C} \rightarrow \mathbb{P}(\mathcal{C})) \times (\mathcal{I} \rightarrow \mathcal{C})$$

where a state consists of three mappings:

- storage function* 1. A *storage function* $s : \mathcal{C} \rightarrow \mathbb{P}(\mathcal{D})$ capturing which containers potentially store which data.
- alias function* 2. An *alias function* $a : \mathcal{C} \rightarrow \mathbb{P}(\mathcal{C})$ capturing that some containers may implicitly get updated whenever other containers do: If $c_2 \in a(c_1)$ for $c_1, c_2 \in \mathcal{C}$, then any data written into c_1 is immediately propagated to c_2 .
- naming function* 3. A *naming function* $n : \mathcal{I} \rightarrow \mathcal{C}$ mapping identifiers to containers.

Given a state $\sigma \in \Sigma$, those mappings will be referred to as $\sigma.s$, $\sigma.a$, and $\sigma.n$. For a system trace $t \in \mathcal{T}$, the system's *initial state* is denoted σ_t^0 .

principals **Principals \mathcal{P} .** The set of all *principals*, i.e. entities capable of issuing system events \mathcal{S} , is denoted by \mathcal{P} . Principals are considered a subset of \mathcal{C} , $\mathcal{P} \subseteq \mathcal{C}$, since they might have read sensitive data in the past and might further propagate that data in the future. As opposed to other containers, principals may invoke events from set \mathcal{S} . In particular, execution of data flow system events \mathcal{S}_F might change the system's data flow state, i.e. the above mappings.

transition relation **Transition relation \mathcal{R}** defines how the system's state changes in correspondence with the execution of actual data flow system events, $\mathcal{R} \subseteq \Sigma \times \mathcal{S}_F^A \times \Sigma$. Notably, intended system events \mathcal{S}^I do not cause data flows. Given a state $\sigma \in \Sigma$ and a set

of actual data flow system events $S \in \{S' \subseteq \mathcal{S}_F^A \mid \nexists e, e' \in S' : e.time = e'.time\}$, function $\tilde{\mathcal{R}} : \Sigma \times \mathbb{P}(\mathcal{S}_F^A) \rightarrow \Sigma$ advances the data flow state by recursively applying state transitions \mathcal{R} while respecting the events' times of observation:

$$\forall \sigma \in \Sigma, S \in \{S' \subseteq \mathcal{S}_F^A \mid \nexists e, e' \in S' : e.time = e'.time\} :$$

$$\tilde{\mathcal{R}}(\sigma, S) = \begin{cases} \sigma & \text{if } S = \emptyset \\ \tilde{\mathcal{R}}(\mathcal{R}(\sigma, e), S \setminus \{e\}) & \text{otherwise,} \\ & \text{for } e \in S : \nexists e' \in S : e'.time < e.time \end{cases}$$

Knowing that for each trace $t \in \mathcal{T}$ and for each timestep $i \in \mathbb{N}$ the set of system events actually causing data flows is $t(i) \cap \mathcal{S}_F^A$, the system's state at the end of timestep $i \in \mathbb{N}$, $i > 0$, is computed as

$$\sigma_t^i = \tilde{\mathcal{R}}(\sigma_t^{i-1}, t(i-1) \cap \mathcal{S}_F^A).$$

Instantiations of this generic data flow model, in particular the semantics of \mathcal{R} , have been described for various system layers [74, 119, 125, 170, 174, 188, 216]. An example instantiation for Unix-like operating systems from the literature is provided in Section 2.3.

Considering the running example of the insurance company, the data items \mathcal{D} to be protected are customer data, contracts, as well as the insurance's business secrets. Among others, containers \mathcal{C} of these data are the clerk's workstations, emails, files, database records, and all data processing software. Identifiers \mathcal{I} correspond to these containers and include unique IDs for workstations and emails, URIs for files, database- and table-relative IDs for database records, as well as host-relative process IDs for processes. Principals \mathcal{P} capable of issuing system events \mathcal{S} are all kinds of data processing software.

Event refinement in the presence of states. Extending the event refinement from Section 2.1.1, $refines_\Sigma \subseteq (\mathcal{S} \times \Sigma) \times \mathcal{E}$ describes the refinement between two events in the presence of a given system state [174]. The rationale is that system events \mathcal{S} *always* operate on containers ($\forall e_1 \in \mathcal{S}, \exists c \in \mathcal{C} : e_1.obj = c$), while policies (Section 2.1.3) might either be specified in terms of data or in terms of containers. Consequently, the system's current state $\sigma \in \Sigma$ must be evaluated in order to decide whether an event refines another. Thus, (e_1, σ) refines $e_2 \in \mathcal{E}$ if either both e_1 and e_2 operate on the same container and if e_1 refines e_2 , or if e_1 operates on some container $c \in \mathcal{C}$ and e_2 operates on some data $d \in \mathcal{D}$ within that container, $d \in \sigma.s(c)$, and e_1 refines e_2 when

ignoring the *obj* parameter:

$$\begin{aligned}
& \forall e_1 \in \mathcal{S}, e_2 \in \mathcal{E}, \sigma \in \Sigma : \\
& \quad (e_1, \sigma) \text{ refines}_{\Sigma} e_2 \\
& \iff \exists c \in \mathcal{C} : e_1.\text{obj} = c \wedge e_2.\text{obj} = c \wedge e_1 \text{ refines } e_2 \\
& \quad \vee \exists c \in \mathcal{C}, d \in \mathcal{D} : e_1.\text{name} = e_2.\text{name} \wedge e_1.\text{obj} = c \wedge e_2.\text{obj} = d \\
& \quad \quad \wedge d \in \sigma.s(c) \wedge e_1.p \setminus \{(obj, c)\} \supseteq e_2.p \setminus \{(obj, d)\}
\end{aligned}$$

Leveraging those concepts of system traces and data flow states, the following section describes both the syntax and semantics of data usage policies.

2.1.3 Specification of Data Usage Policies

While there exist several approaches to specify data usage policies, this thesis builds upon the Obligation Specification Language (OSL) [79, 80, 172, 174], specifically tailored to usage control requirements. Since previous work showed how high-level OSL policies can be translated into technical Event-Condition-Action (ECA) rules for enforcement purposes [108, 109], this thesis focuses on such ECA rules. While the formal semantics of ECA rules are provided in [106, 126], their intuitive semantics are as follows: Once a system event $e' \in \mathcal{S}$ refining the *trigger Event* $e \in \mathcal{E}$ (cf. *refines* and *refines_Σ*) is observed within the system and if the execution of this event would make the *Condition* true, then additional *Actions* might be performed. These actions include (i) allowance of the triggering event, (ii) delaying of the triggering event, i.e. postponing its execution to a later point time, (iii) inhibition of the triggering event, i.e. disallowing its execution altogether, (iv) execution of additional system events, e.g. notifying administrators or triggering other compensating actions [172]. Notably, the trigger event might also be an artificial event, e.g. to indicate that a certain amount of time has passed. According to [108, 109, 156, 172, 174], ECA conditions (Φ) are specified in terms of past linear temporal logics [62, 118, 133]. Their syntax is specified as:

$$\begin{aligned}
\Psi &= \text{true} \mid \text{false} \mid \mathcal{E} \\
\Omega &= \text{isNotIn}(\mathcal{D}, \mathbb{P}(\mathcal{C})) \mid \text{isCombined}(\mathcal{D}, \mathcal{D}, \mathbb{P}(\mathcal{C})) \mid \text{isMaxIn}(\mathcal{D}, \mathbb{N}, \mathbb{P}(\mathcal{C})) \\
\Phi &= (\Phi) \mid \Psi \mid \Omega \mid \text{not}(\Phi) \mid \Phi \text{ and } \Phi \mid \Phi \text{ or } \Phi \mid \Phi \text{ since } \Phi \mid \Phi \text{ before } \mathbb{N} \mid \\
&\quad \text{replem}(\mathbb{N}, \mathbb{N}, \mathcal{E}) \mid \text{repmax}(\mathbb{N}, \mathbb{N}, \mathcal{E}) \mid \text{replim}(\mathbb{N}, \mathbb{N}, \mathbb{N}, \mathcal{E}) \mid \text{always}(\Phi)
\end{aligned}$$

The intuitive semantics are as follows. Ψ is trivial by referring to boolean constants (*true*, *false*) and events \mathcal{E} . Ω defines so-called *state-based operators* that constrain the system's data flow state: *isNotIn*(d, C) is true iff data d is not in any of the containers C ; *isCombined*(d_1, d_2, C) is true iff there exists at least one container in C that contains both data d_1 and d_2 ; *isMaxIn*(d, m, C) is true iff data d is contained in

at most m containers in C . Φ defines *propositional operators*, *temporal operators*, and *cardinality operators*. The semantics of propositional operators *not*, *and* and *or* are intuitive. In terms of temporal operators, α *since* β is true iff β was true some time earlier and α was true ever since, or if α was always true; α *before* j is true iff α was true exactly j timesteps ago. Further, *always* is defined as $\text{always}(\alpha) \equiv \alpha \text{ since false}$. Cardinality operator $\text{repmmin}(j, m, e)$ is true iff event e happened at least m times in the last j timesteps. Further definitions are $\text{repmmax}(j, m, e) \equiv \text{not}(\text{repmmin}(j, m + 1, e))$ and $\text{replim}(j, m, n, e) \equiv \text{repmmin}(j, m, e) \text{ and } \text{repmmax}(j, n, e)$.

propositional operators
temporal operators
cardinality operators

For trace $t \in \mathcal{T}$, timestep $i \in \mathbb{N}$, and condition $\varphi \in \Phi$, notation $(t, i) \models \varphi$ expresses that trace t satisfies formula φ at time i . With this, the formal semantics of Φ is:

$$\begin{aligned}
& \forall t \in \mathcal{T}, i \in \mathbb{N}, \sigma_t^i \in \Sigma, \varphi \in \Phi \bullet (t, i) \models \varphi \iff (\varphi \neq \text{false}) \wedge \\
& \quad (\exists e \in \mathcal{E}, e' \in t(i) \bullet (\varphi = e \wedge (e', \sigma_t^i) \text{refines}_\Sigma e)) \\
& \quad \vee \exists d \in \mathcal{D}, C \subseteq \mathcal{C} \bullet (\varphi = \text{isNotIn}(d, C) \wedge \forall c \in C \bullet d \notin \sigma_t^i.s(c)) \\
& \quad \vee \exists d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C} \bullet (\varphi = \text{isCombined}(d_1, d_2, C) \\
& \quad \quad \wedge \exists c \in C \bullet \{d_1, d_2\} \subseteq \sigma_t^i.s(c)) \\
& \quad \vee \exists d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C} \bullet (\varphi = \text{isMaxIn}(d, m, C) \\
& \quad \quad \wedge |\{c \in C \mid d \in \sigma_t^i.s(c)\}| \leq m) \\
& \quad \vee \exists \alpha, \beta \in \Phi \bullet ((\varphi = \text{not}(\alpha) \wedge \neg((t, i) \models \alpha)) \\
& \quad \quad \vee (\varphi = \alpha \text{ and } \beta \wedge (t, i) \models \alpha \wedge (t, i) \models \beta) \\
& \quad \quad \vee (\varphi = \alpha \text{ or } \beta \wedge ((t, i) \models \alpha \vee (t, i) \models \beta)) \\
& \quad \quad \vee (\varphi = \alpha \text{ since } \beta \wedge (\exists j \in [0, i] \bullet ((t, j) \models \beta \wedge \forall k \in (j, i] \bullet (t, k) \models \alpha)) \\
& \quad \quad \quad \vee \forall k \in [0, i] \bullet (t, k) \models \alpha))) \\
& \quad \vee \exists \alpha \in \Phi, j \in \mathbb{N} \bullet (\varphi = \alpha \text{ before } j \wedge (t, i - j) \models \alpha) \\
& \quad \vee \exists j, m \in \mathbb{N}, e \in \mathcal{E} \bullet (\varphi = \text{repmmin}(j, m, e) \\
& \quad \quad \wedge m \leq \sum_{k=0}^{\min\{i, j\}-1} |\{e' \in t(i - k) \mid (e', \sigma_t^{i-k}) \text{refines}_\Sigma e\}|))
\end{aligned}$$

Fixing one data item d , Table 2.2 shows how the example policies from the running example (cf. Section 1.5) can be expressed as ECA rules. Rule 1a expresses that the manager must be notified if no contract offer has been sent to the customer 30 days after a corresponding contract request. Note that this event does have a wildcard trigger event, implying that the rule is evaluated upon every event. Further, this rule is detective only: satisfaction of the condition results in a compensating action; actual violation of the policy is not prevented. Rule 1b expresses that a contract offer must not be sent if there was no corresponding contract request, or if a contract offer was already sent. Rule 2 expresses that any attempt to use data d is inhibited if the corresponding contract offer was declined in the past. Note, that trigger event *use* refers to a set of events. This set might include events such as editing, reviewing,

Table 2.2: Specification of the example policies from Section 1.5 as ECA rules.

Policy 1	Event: $\langle any \rangle$
(a)	Condition: $((requestOffer, \{(obj, d)\}) \textit{ before } 30)$ $\textit{ and } repmax(30, 0, (sendOffer, \{(obj, d)\}))$
	Action: $(notifyManager, \{(obj, d)\})$
	Event: $(sendOffer, \{(obj, d)\})$
(b)	Condition: $repmax(30, 0, (requestOffer, \{(obj, d)\}))$ $\textit{ or } repmin(30, 1, (sendOffer, \{(obj, d)\}))$
	Action: $inhibit$
Policy 2	Event: $(use, \{(obj, d)\})$
	Condition: $not(always(not((declineOffer, \{(obj, d)\}))))$
	Action: $inhibit$
Policy 3	Event: $(sendOffer, \{(obj, d)\})$
	Condition: $repmax(30, 1, (review, \{(obj, d)\}))$ $\textit{ or } repmax(30, 1, (approve, \{(obj, d)\}))$
	Action: $inhibit$
Policy 4	Event: $\langle any \rangle (d)$
	Condition: $not(isMaxIn(1, d, C_{Workstation}))$
	Action: $inhibit$
Policy 5	Event: $(edit, \{(obj, d)\})$
	Condition: $not(isMaxIn(d, 0, C_{EditProcess}))$ $\textit{ or } (archive, \{(obj, d), (role, manager)\}) \textit{ since false}$
	Action: $inhibit$
Policy 6	Event: $(send, \{(obj, d)\})$
	Condition: $not(isNotIn(d, C \setminus C_{Insurance}))$ $\textit{ and } not((approvedSend, \{(obj, d), (role, manager)\}))$
	Action: $inhibit$

analyzing, and sending of corresponding contracts or contract offers. Rule 3 expresses that sending of a contract is inhibited if this contract was not reviewed or approved by at least two clerks in the last 30 days. Rule 4 expresses that any event must be inhibited if its execution would lead to a state in which data d is in more than one of the clerk's workstations, whereby $C_{Workstation}$ denotes the set of containers representing the insurance's workstations. With $C_{EditProcess}$ denoting the set of all processes with the capability to edit documents, rule 5 expresses that a contract may only be edited if there is not yet any ongoing edit process and if the contract was not archived by a manager in the past. Finally, rule 6 expresses that sending of market research results is only allowed if the data is not sent to containers outside of the insurance company ($C_{Insurance}$), or if sending of the data has been approved by a manager.

The following section sketches how such ECA rules can technically be enforced.

2.2 Enforcement Infrastructure

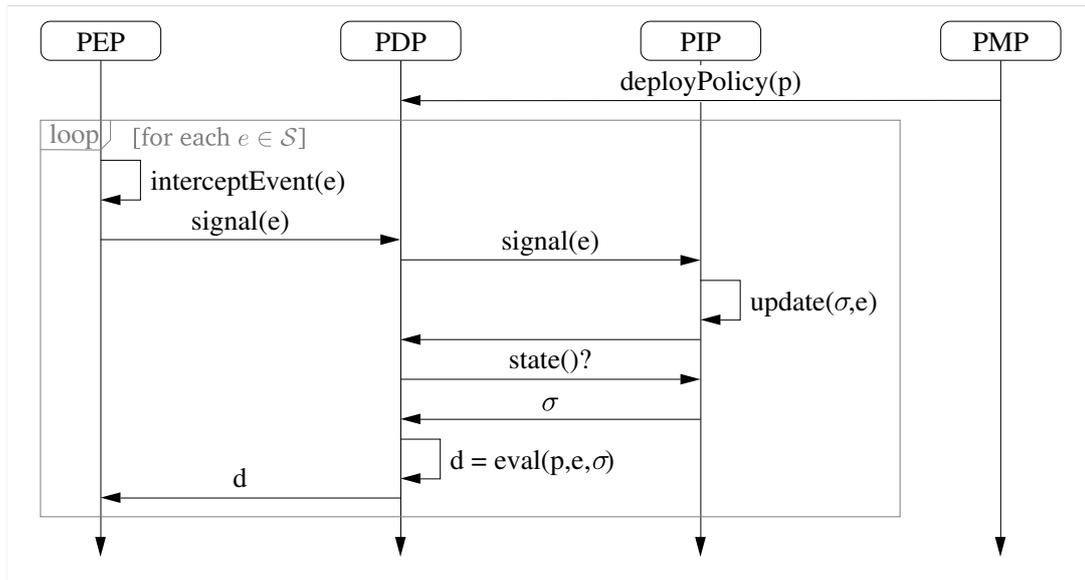
In order to implement the concepts introduced in Section 2.1 and to enforce the corresponding ECA rules, technical data usage control infrastructures have been developed. This thesis builds upon an existing technical infrastructure [110, 156, 174], the core concepts of which are sketched in this section. Chapter 4 provides more technical details about the infrastructure and its implementation. Once deployed within a computing system, the infrastructure's tasks are as follows:

- ▶ Monitor relevant data usage system events \mathcal{S}_U and data flow system events \mathcal{S}_F .
- ▶ Track the flow of data within the system, potentially across several system layers, using data flow system events \mathcal{S}_F , transition relation \mathcal{R} , and data flow states Σ .
- ▶ Decide whether any data usage system event \mathcal{S}_U ought to be allowed, modified, inhibited, delayed and/or whether any compensating actions need to be taken.
- ▶ Enforce the decisions taken by blocking or delaying system events and/or by executing additional actions.

While the concepts described in this section are *not* a contribution of this thesis, parts of the described infrastructure have been (re-)developed and (re-)implemented within this thesis. This includes the technical system architecture and the interconnection of the individual components (Section 4.1), the implementation of the policy evaluation engine (Section 2.2.3), and parts of the generic data flow model (Section 2.2.4).

2.2.1 Architecture Overview

Data usage control infrastructures are commonly implemented in correspondence with the XACML standard architecture [160] and the proposed COPS standard [49, 104] as follows and depicted in Figure 2.1: Initially, the Policy Management Point (PMP, Section 2.2.5) deploys the ECA rules to be enforced at the Policy Decision Point (PDP, Section 2.2.3). It is the task of system-layer specific Policy Enforcement Points (PEPs, Section 2.2.2) to observe the system layer's events and to intercept both actual and intended system events \mathcal{S} . Those intercepted events are then temporarily blocked from execution and signaled to the PDP [31, 115, 174]. The PDP first forwards these events to the Policy Information Point (PIP, Section 2.2.4) which maintains the system's current data flow state $\sigma \in \Sigma$. In the presence of a data flow system event, the PIP updates the system's data flow state σ in accordance with the event's semantics as defined by transition relation \mathcal{R} (cf. Section 2.3 for an example instantiation). The PDP then evaluates each signaled system event against all deployed ECA rules, as

Figure 2.1: Interactions of PEP, PDP, PIP and PMP upon observation of system event e .

further detailed in Section 2.2.3, and sends its decision back to the corresponding PEP for enforcement. For taking this decision, the PDP might ask the PIP for information about the system’s current data flow state σ . Since the PIP maintains the system’s data flow state, the terms *the system’s data flow state* and *the PIP’s data flow state* are used interchangeably. Note that the usage control infrastructure keeps *all* management information, such as policies and the data flow state, separate from the actual payload data, i.e. the data being stored, processed and disseminated by the monitored system layers. Several related works take a different approach by embedding such management information within the actual payload, cf. Chapter 6.

Having given a high-level overview of the involved components and their interplay, the following sections detail the tasks of the single components. Chapter 4 will provide in-depth technical details.

2.2.2 Policy Enforcement Point

Policy Enforcement Points (PEPs) have been built for many different system layers such as Android [55, 180], ChromiumOS [214], Java [61, 63], JavaScript [168], Mozilla Firefox [110], Mozilla Thunderbird [125], MS Office [188], MS Windows [216], MySQL [119], OpenBSD [74], OpenNebula [115], and X11 [170]. It is their task to intercept intended and/or actual system events (i.e. $S^I \cup S^A$) within the layer they have been built into. Whether actual or intended events are intercepted depends on the considered attacker model and the technicalities of the system layer. E.g., for some system layers it might be technically impossible to intercept intended events. Further, for certain scenarios and attacker models preventive policy enforcement, and

thus the interception of intended events, might not be needed or desired, e.g. because data users are trusted to not misuse data intentionally.

Because PEPs are unaware of the currently deployed policies as well as of the data flow semantics of the intercepted system events, *every* intercepted system event must be signaled to the PDP. The further execution of an intercepted and signaled event is then blocked until the PDP's decision is available. Once available, the decision is enforced by the PEP. Besides data usage system events, the PEP must also signal all potential data flow system events to the PDP. The PDP will then take care of forwarding those events to the PIP, which will update the system's data flow state in correspondence with those signaled events.

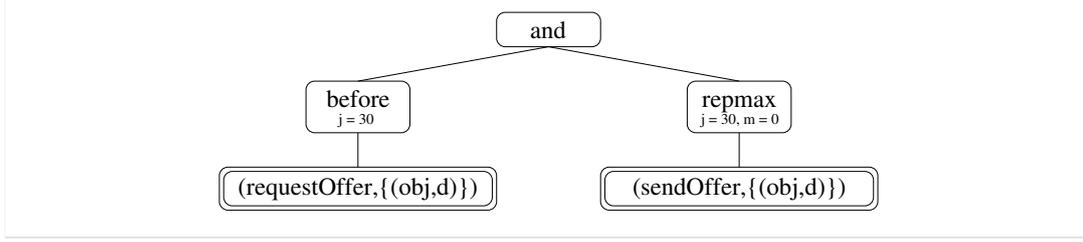
Regarding the running example, PEPs must be integrated into the different system layers processing and storing the usage controlled data. These layers include email clients and servers, web servers, data analysis and word processing software, databases, etc. Additional PEPs at the operating system layer can assure that data is not used or copied in an uncontrolled manner, e.g. by leveraging applications that are not equipped with data usage control technology.

2.2.3 Policy Decision Point

The Policy Decision Point (PDP) is configured with ECA rules as described in Section 2.1.3 and each ECA rule p consists of a trigger event $e_p \in \mathcal{E}$, a condition $\varphi_p \in \Phi$ and an action a_p . Moreover, each ECA rule is configured with a timestep interval, a concept not only needed to model system runs in terms of traces (Section 2.1.1), but also for practical reasons: Consider a part of the condition of ECA rule 1a as defined in Table 2.2, $\varphi_p = (\text{requestOffer}, \{(obj, d)\}) \text{ before } 30[\text{days}]$, with $(\text{requestOffer}, \{(obj, d)\}) \in \mathcal{E}$. Whenever φ_p is evaluated it is quite unlikely that an event refining $(\text{requestOffer}, \{(obj, d)\})$ happened *exactly* 30 days (i.e. 2592000 seconds) ago. What is more likely and practical, however, is that a refining event happened 'approximately' 30 days ago. E.g., if the timestep interval is configured to be 24 hours, then the PDP evaluates whether a refining event happened before 29.5 days \pm 12 hours. Similarly, consider the conjunction and disjunction of operators, *and* and *or*. While it is unlikely that two events happen at *exactly* the same point in time, what is more likely and practical is that two events happen within a specified time interval, i.e. within the same timestep.

Once policies have been deployed, the PDP's decisions ought to ensure the system's compliance with those policies. To this end, the PDP continuously evaluates the deployed policies, considering conditions $\varphi_p \in \Phi$ as *expression trees* as depicted in Figure 2.2 and explained in the following: Leaves represent the constants *true* and *false*, events \mathcal{E} (e.g., $(\text{requestOffer}, \{(obj, d)\})$ and $(\text{sendOffer}, \{(obj, d)\})$ in Figure 2.2), and state-based operators Ω . Internal nodes represent all further propositional, temporal, and cardinal operators such as *not*, *and*, *before*, *since*, and *repm*. Both the expression trees' leaves and internal nodes are stateful by storing if and how the represented

expression tree

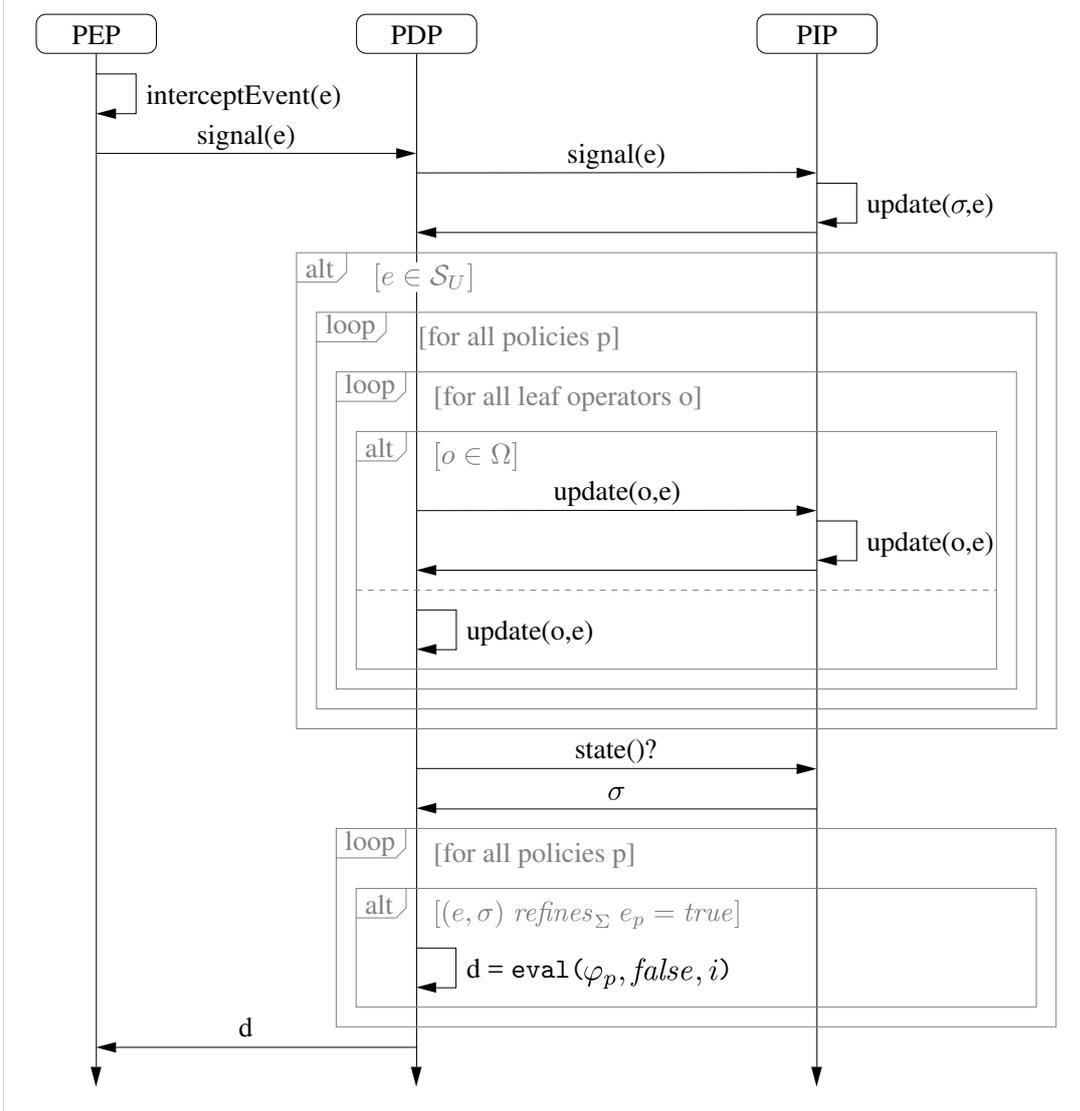
Figure 2.2: Expression tree of the condition of ECA rule 1a.**Table 2.3: Evaluation of operators Ψ by the PDP; Java-like syntax.**

$\varphi \in \Psi$	<code>eval(φ, eot, i), $eot \in \mathbb{B}$, $i \in \mathbb{N}$</code>	State variables
<code>true</code>	<code>return true;</code>	
<code>false</code>	<code>return false;</code>	
$e' \in \mathcal{E}$	<pre> int oldcount = count; if (eot) count = 0; return oldcount > 0; </pre>	<pre> int count = 0; // how many events refining // e' happened in the // current timestep </pre>

operator's state changed during the current, and, for temporal operators, previous timesteps.

Now, consider the case that some PEP signals a system event $e \in \mathcal{S}$, cf. Figure 2.3. The PDP first forwards e to the PIP, which updates its internal data flow state in correspondence with e 's data flow semantics as defined by transition relation \mathcal{R} (Section 2.2.4). This ensures that the subsequent evaluation of the deployed policies is grounded on an up-to-date data flow state. If the signaled event is a data usage event, $e \in \mathcal{S}_U$, then the PDP updates the internal states of all *leaf nodes* of the conditions of all deployed policies. For leaf nodes o representing state-based operators, $o \in \Omega$, this update process is delegated to the PIP. The PDP takes care of updating leaf nodes representing operators `true`, `false`, and events \mathcal{E} . Trivially, the states of operators `true` and `false` are invariant. For a leaf representing event $e' \in \mathcal{E}$, the leaf's state is a counter indicating how many events refining e' happened within the current timestep. This counter is incremented if $(e, \sigma) \text{refines}_\Sigma e' = \text{true}$, whereby $\sigma \in \Sigma$ refers to the system's current data flow state as maintained by the PIP (Section 2.2.4). In sum, the expression trees' leaves thus track how often events have happened and which state-based operators have changed their state during the ongoing timestep. Note again that this update of leaf nodes is performed for *all* deployed policies independent of whether the policy's trigger event was refined by the signaled event e . This procedure ensures that the states of all leaf nodes of all expression trees are consistent with the events that have actually happened within the system.

For any policy p , the actual evaluation of the entire condition φ_p may be triggered for two reasons: (1) In case the signaled event e refines the policy's trigger event e_p , i.e. $(e, \sigma) \text{refines}_\Sigma e_p = \text{true}$, then it must be evaluated whether e complies with

Figure 2.3: Policy evaluation by the PDP in the presence of event e at timestep i .

the policy; (2) In case a timestep has passed, then it must be evaluated whether temporal constraints such as *before* and *since* are still satisfied. In both cases, the entire expression tree of φ_p is evaluated recursively, starting from the root node. The result of this evaluation is denoted $eval(\varphi_p)$, which essentially reflects the formal notation $(t, i) \models \varphi_p$ with $t \in \mathcal{T}$ denoting the currently executing trace und $i \in \mathbb{N}$ the current point in time. If $eval(\varphi_p) = true$ (i.e. $(t, i) \models \varphi_p$), then p 's actions a_p are triggered, which may include execution of additional system events, or, if evaluating in the presence of a signaled event, its *allowance*, *inhibition* or *delaying*. If $eval(\varphi_p) = false$ (i.e. $(t, i) \not\models \varphi_p$), then the default is to *allow* the event. However, the implementation is more flexible by allowing to specify arbitrary default actions on a per-policy basis.

Tables 2.3 and 2.4 sketch the recursive evaluation process for operators Ψ and Φ after updating the expression trees' leaf nodes as described above. Thereby, arguments of the implementation-level evaluation function $eval(\varphi, eot, i)$ are as follows: φ is the

Table 2.4: Evaluation of operators Φ by the PDP; Java-like syntax.

$\varphi \in \Phi$	<code>eval(φ, eot, i), eot $\in \mathbb{B}$, i $\in \mathbb{N}$</code>	State variables
$\underline{not}(\alpha)$	<code>return !eval(α, eot, i);</code>	
α <u>and</u> β	<code>return eval(α, eot, i) && eval(β, eot, i);</code>	
α <u>or</u> β	<code>return eval(α, eot, i) eval(β, eot, i);</code>	
α <u>since</u> β	<code>bool stateA = eval(α, eot, i); bool stateB = eval(β, eot, i); alwaysA &= stateA; if (alwaysA) return true; if (stateB) alwaysASinceB = true; else alwaysASinceB &= stateA; return alwaysASinceB;</code>	<code>bool alwaysA = true; // whether α was // true at all // previous timesteps bool alwaysASinceB = true; // whether α was true // at all timesteps since // the last time β was // true</code>
α <u>before</u> j	<code>bool res = prev[i % j]; if (eot) prev[i % j] = eval(α, eot, i); return res;</code>	<code>bool[] prev = new bool[j]; // evaluation results // of α of the last j // timesteps</code>
$\underline{repm}in(j, m, e')$	<code>eval(e', eot, i); total -= prev[i % j]; prev[i % j] = e'.count; total += prev[i % j]; return total >= m;</code>	<code>int[] prev = new int[j]; // refinements of e' // for each of the // last j timesteps int total = 0; // total refinements // of e' in the // last j timesteps</code>
$\underline{always}(\alpha)$	<code>if (!always) return false; always &= eval(α, eot, i); return always;</code>	<code>bool always = true; // whether α was // always true</code>

condition to be evaluated, $eot \in \mathbb{B}$ indicates whether evaluation is taking place at the end of a timestep ($eot = true$) or in the presence of an event ($eot = false$), and $i \in \mathbb{N}$ is the timestep in which the evaluation is taking place. Evaluation of operators *true* and *false* (Table 2.3) as well as *not*, *and* and *or* (Table 2.4) is trivial. Evaluation of the more complex operators is explained in the following.

For $\varphi = e' \in \mathcal{E}$ (Table 2.3), `eval()` returns *true* if at least one event refining e' happened within the current timestep; the corresponding counter (count) is reset if evaluation takes place at the end of a timestep (i.e. if $eot = true$).

For α *since* β (Table 2.4), state variable `alwaysA` indicates whether α was *true* at all times, while `alwaysASinceB` indicates whether α was *true* at all timesteps since the last time β was true; both state variables are initialized to *true* (Table 2.4, third column). After recursive evaluation of α and β , `alwaysA` is updated with this most recent evaluation of α , thus indicating whether α was (still) *true* at all times. If this is the case, `eval()` returns *true*. Otherwise, if the evaluation of β at this timestep yielded *true*, then α was always true since the last time β was true (since β is true in this timestep), which is why `alwaysASinceB` is set to *true*. If, however, β is not true at this timestep, then `alwaysASinceB` only remains a value of *true* if its previous value was *true* and if the evaluation of α yielded *true*, thus indicating that α was in fact *true* ever since the last time β was *true*.

For α *before* j (Table 2.4) a boolean array stores the evaluation results of α of the last j timesteps. The result of `eval()` thus corresponds to the value that was stored in the array j timesteps ago. If evaluation is taking place at the end of a timestep, then the oldest entry within that array is overwritten with the evaluation result of α at this timestep.

For *repm* $in(j, m, e')$ (Table 2.4), an integer array of size j stores the number of refinements of e' for each of the last j timesteps. The initial evaluation of e' updates e' 's state variable `count` (cf. Table 2.3), which is then used to update the array's latest value. The state variable `total` keeps the sum of the amount of refinements within the last j timesteps for performance reasons and is updated accordingly upon each evaluation. `eval()` returns *true* if the value of `total` is greater than or equal to m .

For *always*(α) (Table 2.4), state variable `always`, which is initialized to *true*, stores whether α was always *true*. Upon evaluation, *false* can immediately be returned if the value of variable `always` is *false*. Otherwise, `always` is updated in correspondence with the evaluation result of α at this timestep. Note that once `always` was assigned *false*, no further evaluation of α is ever needed and `eval()` always returns *false* immediately.

Note the following: If the event originally signaled by the PEP was an intended event, then the changes made to φ_p 's expression tree's nodes' states, as well as all changes made to the PIP's data flow state are rolled back after the above evaluation of φ_p has been performed. This is because both the PDP and PIP simulate how the allowance of the signaled intended event *would* change both the PDP's and the PIP's state, which have major influence on the evaluation of φ_p . However, since the signaled event was

not actual, any such changes have not happened in the real system and modeling them would thus be incorrect. Consequently, the mentioned changes are undone.

2.2.4 Policy Information Point

The Policy Information Point's (PIP) tasks are to maintain the system's data flow state and to evaluate state-based operators. Whenever the PDP signals a system event $e \in \mathcal{S}$, the PIP first evaluates whether e is a data flow event for which there exists a corresponding state transition, i.e. whether $\exists \sigma, \sigma' \in \Sigma : (\sigma, e, \sigma') \in \mathcal{R}$. If this is the case, then the data flow state is updated according to the semantics of e as defined by $\mathcal{R} \subseteq \Sigma \times \mathcal{S}_F^A \times \Sigma$. If the PDP signaled an event $e \notin \mathcal{S}_F^A$, then the PIP's state remains unchanged.

In order to allow the PDP to evaluate event refinement $refines_{\Sigma}$, the PIP provides an appropriate interface. Recapping parts of the definition of $(e_1, \sigma) refines_{\Sigma} e_2$ for $e_1 \in \mathcal{S}$, $e_2 \in \mathcal{E}$, $\sigma \in \Sigma$, the PDP needs to examine whether $d \in \sigma.s(c)$ for $c \in \mathcal{C}$, $d \in \mathcal{D}$ and $e_1.obj = c$, $e_2.obj = d$ (Section 2.1.2). For this, the PIP provides an interface using which the PDP is able to query all data contained in a specific container c , effectively returning set $\sigma.s(c)$. Using this result set, the PDP is able to evaluate whether $d \in \sigma.s(c)$ and consequently whether $(e_1, \sigma) refines_{\Sigma} e_2$. Note that this has been simplified in Figures 2.1 and 2.3.

Since state-based operators Ω are all about the system's data flow state, the PDP delegates their evaluation to the PIP. Thus, the PIP evaluates operators Ω in correspondence with their semantics described in Section 2.1.3 on the basis of its current data flow state $\sigma \in \Sigma$ using simple set operations as sketched in Table 2.5. The result returned by the PIP, *true* or *false*, is used by the PDP to update the state of the corresponding state-based operator within the expression tree of the condition φ_p being evaluated.

2.2.5 Policy Management Point

As the name suggests, the Policy Management Point's (PMP) main tasks are related to the management of data usage policies. Those tasks include deployment and revocation of policies at the PDP, as well as conversion and translation of policies between different formal description languages, system layers, and technical formats [106]. As the PMP is of no further relevance at this point, its detailed description is deferred to Chapter 4.

2.3 Instantiation to Unix-like Systems

As mentioned in Section 2.2.2, the aforementioned concepts have been instantiated for many different system layers. While PDP, PIP and PMP are generic and operate independently of concrete system layers and their events, PEPs must be tailored to and built into those layers. This section demonstrates the instantiation for an operating

Table 2.5: Evaluation of operators Ω by the PIP; Java-like syntax.

$\varphi \in \Omega$	<code>eval(φ, eot, i), $eot \in \mathbb{B}, i \in \mathbb{N}; \sigma \in \Sigma$</code>
$\underline{isCombined}(d_1, d_2, C)$	<pre> for (Container c : C) { Set D = $\sigma.s(c)$; if (D.contains(d_1) && D.contains(d_2)) { return true; } } return false; </pre>
$\underline{isNotIn}(d, C)$	<pre> for (Container c : C) { Set D = $\sigma.s(c)$; if (D.contains(d)) { return false; } } return true; </pre>
$\underline{isMaxIn}(d, m, C)$	<pre> int j = 0; for (Container c : C) { Set D = $\sigma.s(c)$; if (D.contains(d)) { j++; } } return j <= m; </pre>

system layer, namely OpenBSD, as originally described in [74]. This thesis will further build on this instantiation in Section 3.2.2. Instantiations for other operating system layers have been shown to be similar [55, 214, 216]. In particular, the concepts described in the following can be directly applied to any other Unix-like system such as Linux. The reason is that the considered system events, i.e. system calls, and their corresponding event semantics are, besides minor exceptions, identical.

Notation. Before describing \mathcal{R} , further notation must be introduced. For specifying state changes \mathcal{R} , the following notation is used. For mappings $m, m' : S \rightarrow T$ and an element $x \in X \subseteq S$, define $m[x \leftarrow expr]_{x \in X} = m'$ such that

$$m'(y) = \begin{cases} expr & \text{if } y \in X \\ m(y) & \text{otherwise} \end{cases}$$

Multiple updates of disjoint sets are combined by function composition \circ . The replacements are done simultaneously and atomically; the semicolon is syntactic sugar:

$$\begin{aligned} & m[x_1 \leftarrow expr_{x_1}; \dots; x_n \leftarrow expr_{x_n}]_{x_1 \in X_1, \dots, x_n \in X_n} \\ & = m[x_n \leftarrow expr_{x_n}]_{x_n \in X_n} \circ \dots \circ m[x_1 \leftarrow expr_{x_1}]_{x_1 \in X_1} \end{aligned}$$

Further, $\forall \sigma \in \Sigma, c \in \mathcal{C}, \sigma.a^*(c)$ denotes the reflexive transitive closure of $\sigma.a(c)$, i.e. the smallest set satisfying $\sigma.a^*(c) = \{c\} \cup \{c' \in \mathcal{C} \mid c' \in \sigma.a(c) \vee (\exists c'' \in \sigma.a(c) \wedge c' \in \sigma.a^*(c''))\}$.

Instantiation. At the OpenBSD operating system layer, *events* are system calls, such as *open*, *read*, *write*, *close*, *pipe*, and *fork*. Corresponding monitors (e.g. on the basis of ptrace [120] or systrace [178]) are able to intercept system calls both before and after their execution by the kernel, resulting in intended system calls and actual system calls, \mathcal{S}^I and \mathcal{S}^A . *Principals* executing system calls are processes $\mathcal{P}_{Proc} \subseteq \mathcal{P}$ and *containers* are their memory, $\mathcal{C}_{Proc} \subseteq \mathcal{C}$, files $\mathcal{C}_{File} \subseteq \mathcal{C}$, pipes $\mathcal{C}_{Pipe} \subseteq \mathcal{C}$, and sockets $\mathcal{C}_{Sock} \subseteq \mathcal{C}$. Corresponding *identifiers* for those containers are process ids $\mathcal{I}_{Pid} \subseteq \mathcal{I}$, filenames $\mathcal{I}_{Fname} \subseteq \mathcal{I}$, and process-relative file descriptors $(\mathcal{I}_{Pid} \times \mathcal{I}_{Fdsc}) \subseteq \mathcal{I}$. Data flow semantics, i.e. state transitions \mathcal{R} , for selected system calls are detailed in the following. By applying those state transitions to the system's data flow state $\sigma \in \Sigma$ in correspondence with the observed actual data flow system events \mathcal{S}_F^A , σ evolves accordingly. In the following, state transitions for some of the most important system calls, i.e. *open*, *read*, *write*, *close*, *pipe*, *fork*, are described.

If a process with process id $pid \in \mathcal{I}_{Pid}$ (parameter *proc*) issues system call *open* with filename $fn \in \mathcal{I}_{Fname}$ (parameter *fname*), a new file descriptor $fd \in \mathcal{I}_{Fdsc}$ is created and returned (parameter *ret*). File descriptor fd can then be used by process pid to access the corresponding file. The following state transition models this behavior by creating a new identifier (pid, fd) for the container that was already identified by fn :

$$\begin{aligned} & \forall \sigma, \sigma' \in \Sigma, \forall fn \in \mathcal{I}_{Fname}, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc} : \\ & (\sigma, (open, \{(obj, \sigma.n(fn)), (proc, pid), (fname, fn), (ret, fd)\}), \sigma') \in \mathcal{R} \\ & \implies \sigma'.s = \sigma.s \\ & \quad \wedge \sigma'.a = \sigma.a \\ & \quad \wedge \sigma'.n = \sigma.n[(pid, fd) \leftarrow \sigma.n(fn)] \end{aligned}$$

System call *read* allows a process with process id $pid \in \mathcal{I}_{Pid}$ (parameter *proc*) to read data from any resource for which this process owns a file descriptor $fd \in \mathcal{I}_{Fdsc}$ (parameter *fdscr*). This is modeled by propagating all data from the corresponding resource to the process' memory. Further, the data read is immediately propagated to all containers that are transitively aliased by $\sigma.n(pid)$, i.e. all containers in set $\sigma.a^*(\sigma.n(pid))$:

$$\begin{aligned} & \forall \sigma, \sigma' \in \Sigma, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc} : \\ & (\sigma, (read, \{(obj, \sigma.n((pid, fd))), (proc, pid), (fdscr, fd)\}), \sigma') \in \mathcal{R} \\ & \implies \sigma'.s = \sigma.s[t \leftarrow \sigma.s(t) \cup \sigma.s(\sigma.n((pid, fd)))]_{t \in \sigma.a^*(\sigma.n(pid))} \\ & \quad \wedge \sigma'.a = \sigma.a \\ & \quad \wedge \sigma'.n = \sigma.n \end{aligned}$$

Similarly, process $pid \in \mathcal{I}_{Pid}$ (parameter *proc*) might use system call *write* to write some data from its process memory to a file descriptor $fd \in \mathcal{I}_{Fdsc}$ (parameter *fdscr*).

This is modeled by propagating all data from the process' memory to the corresponding container identified by file descriptor fd . Again, the data is immediately propagated to all containers that are transitively aliased from container $\sigma.n((pid, fd))$:

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc} : \\
& (\sigma, (write, \{(obj, \sigma.n((pid, fd))), (proc, pid), (fdscr, fd)\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s[t \leftarrow \sigma.s(t) \cup \sigma.s(\sigma.n(pid))]_t \in \sigma.a^*(\sigma.n(pid, fd)) \\
& \quad \wedge \sigma'.a = \sigma.a \\
& \quad \wedge \sigma'.n = \sigma.n
\end{aligned}$$

A file descriptor $fd \in \mathcal{I}_{Fdsc}$ (parameter $fdscr$) may be closed by the owning process $pid \in \mathcal{I}_{Pid}$ (parameter $proc$) using system call $close$. Subsequently, fd can no longer be used for accessing the corresponding resource. The reserved value $nil \in \mathcal{C}$ is used to refer to non-existing containers.

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc} : \\
& (\sigma, (close, \{(obj, \sigma.n((pid, fd))), (proc, pid), (fdscr, fd)\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s \\
& \quad \wedge \sigma'.a = \sigma.a \\
& \quad \wedge \sigma'.n = \sigma.n[(pid, fd) \leftarrow nil]
\end{aligned}$$

Process $pid \in \mathcal{I}_{Pid}$ (parameter $proc$) can create a pipe for communication purposes using system call $pipe$. Besides creating the new pipe $c_p \in \mathcal{C}_{Pipe}$, $pipe$ returns two new file descriptors, $fd_1, fd_2 \in \mathcal{I}_{Fdsc}$ (parameter ret), which can be used to read from and write to c_p :

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall pid \in \mathcal{I}_{Pid}, \forall c_p \in \mathcal{C}_{Pipe}, \forall fd_1, fd_2 \in \mathcal{I}_{Fdsc} : \\
& (\sigma, (pipe, \{(obj, c_p), (proc, pid), (ret, (fd_1, fd_2))\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s \\
& \quad \wedge \sigma'.a = \sigma.a \\
& \quad \wedge \sigma'.n = \sigma.n[(pid, fd_1) \leftarrow c_p; (pid, fd_2) \leftarrow c_p]
\end{aligned}$$

System call $fork$ allows process $pid \in \mathcal{I}_{Pid}$ (parameter $proc$) to create a new child process. The process id of the created child, $cpid \in \mathcal{I}_{Pid}$, is the return value of $fork$ (parameter ret). Notably, the child process is an exact copy of the parent process. Consequently, all process-internal identifiers (e.g. the file descriptor table) and the process memory are cloned. However, from now on these structures, in particular the data stored within the two process memories as well as the file descriptor tables, evolve differently. Consequently, a new process container $c_{cpid} \in \mathcal{C}_{Proc}$ is created and

all mappings from the calling process pid are cloned for the new process c_{pid} :

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall pid, cpid \in \mathcal{I}_{Pid}, \forall c_{cpid} \in \mathcal{C}_{Proc} : \\
& (\sigma, (fork, \{(obj, c_{cpid}), (proc, pid), (ret, cpid)\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s[c_{cpid} \leftarrow \sigma.s(\sigma.n(pid))] \\
& \quad \wedge \sigma'.a = \sigma.a[c_{cpid} \leftarrow \sigma.a(\sigma.n(pid)); \\
& \quad \quad t \leftarrow \sigma.a(t) \cup \{c_{cpid}\}]_{t \in \{t' | \sigma.n(pid) \in \sigma.a(t')\}} \\
& \quad \wedge \sigma'.n = \sigma.n[cpid \leftarrow c_{cpid}; (cpid, fd) \leftarrow \sigma.n((pid, fd))]_{fd \in \mathcal{I}_{Fds}}
\end{aligned}$$

Having given an overview over an instantiation of the model described in Section 2.1 for Unix-like systems, [74] details the semantics of additional system calls such as *execve*, *dup*, *rename*, *mmap*, *exit*, and *kill*. Since this thesis extends the above instantiation (Section 3.1), most of the above event semantics have been selected for presentation because they will be of further relevance within this thesis.

3

Distributed Data Usage Control

This chapter describes major contributions of this dissertation, namely usage control models and formal methods that allow to enforce data usage control policies in distributed system environments. The need for such extended concepts arises because in real-world scenarios the data to be protected by usage control policies is disseminated throughout different systems, cf. the example policies in Section 1.5. Generally, those systems are designed to be independent and to operate individually, without the need for central components. Hence, the motive of this work was to improve the model from Section 2.1 in such a way that it accommodates both the independence of individual systems as well as the possibility to enforce distributed data usage control requirements. Notably, the concepts introduced in Section 2.1 do not cater to any form of distribution.

This chapter describes how usage control policies can be enforced within such distributed environments. Section 3.1 introduces a distributed system model by incorporating distributed aspects into the model described in Section 2.1. Based on this model, Section 3.2 describes how data flows can be tracked across systems such that the usage control infrastructure is aware of all copies of some data even across system boundaries (**RQ1**, Section 1.1.1). Further, Section 3.3 describes how policies can be enforced in an efficient and decentralized manner if data and events are distributed (**RQ2**, Section 1.1.2).

As a part of this dissertation, most of the work described in the subsequent sections has been published in [93, 94, 95]. Citations and verbatim quotes from these sources are not explicitly cited. However, each section initially clarifies from which papers such citations and quotes have been taken.

3.1 Distributed System Model

Contents of this section have been published in [95].

The model described in Section 2.1 suggests a monolithic view on policy enforcement, meaning that at runtime there exists one single trace and one single system state at

any point in time. Technically, one single PDP observes and regulates the execution of all system events, while one single PIP maintains the global data flow state. As also pointed out in [17, 88], such a centralized approach is expected to be impractical if data is shared between systems and if usage policies do impose global requirements, e.g. by referring to data flow states or data usage events of remote systems. In particular, this is the case for the example scenario of the insurance company introduced in Section 1.5 and the corresponding policies.

This section introduces an extended model that allows for the explicit distinction of different systems, their individual behaviors, as well as their interplay. Aforementioned usage control models do not incorporate any such concepts, implicitly operating on one single large monolithic system. In the following, this monolithic system will be referred to as the *distributed system*, in which multiple PDPs and PIPs observe and regulate different/disjoint parts. Further, this section formalizes how these individual observations can be combined such that the observations of a single monolithic PDP and PIP are reassembled.

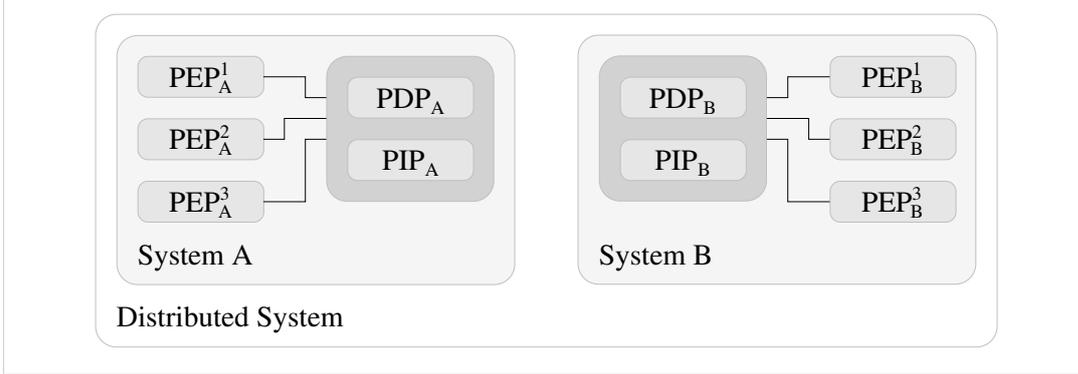
3.1.1 Individual Systems

systems **Systems** \mathcal{Y} constitute autonomous parts of the entire distributed system. A *system* is defined as a non-empty set of system layers whose PEPs share the same PDP and PIP for policy decision and data flow tracking purposes. Hence, a system consists of exactly one PDP/PIP and at least one PEP (cf. Figure 3.1). Note that those system layers sharing the same PDP/PIP must not necessarily remain on the same physical or virtual host. A system may thus be an operating system instance, a physical or virtual machine, a set of applications, or even a set of physical or virtual machines. Section 3.2.1 defines the set of systems \mathcal{Y} in a more technical manner.

The entire distributed system is then composed of the set of all systems \mathcal{Y} . While those systems \mathcal{Y} are generally independent from another, they are capable of interacting among each other using communication infrastructures such as network sockets. In the running example, such systems are the insurance company's webserver, the email server, the database, as well as the clerks' workstations.

For each system $y \in \mathcal{Y}$, $\mathcal{S}_y \subseteq \mathcal{S}$ denotes the system's unique set of system events, $\mathcal{C}_y \subseteq \mathcal{C}$ its unique set of containers, and $\mathcal{I}_y \subseteq \mathcal{I}$ its unique set of identifiers. To be able to differentiate system events originating from different systems, each system event $e \in \mathcal{S}_y$ is required to carry parameter $sys \in \mathcal{N}$ with value $y \in \mathcal{Y}$, hence $\mathcal{Y} \subseteq \mathcal{V}$:

$$\begin{aligned} \forall y_1, y_2 \in \mathcal{Y}, e \in \mathcal{S}_{y_1} : \\ (sys, y_1) \in e.p \\ \wedge y_1 \neq y_2 \implies \mathcal{S}_{y_1} \cap \mathcal{S}_{y_2} = \emptyset \wedge \mathcal{C}_{y_1} \cap \mathcal{C}_{y_2} = \emptyset \wedge \mathcal{I}_{y_1} \cap \mathcal{I}_{y_2} = \emptyset \end{aligned}$$

Figure 3.1: Two independent systems, each featuring three PEPs.

On this basis, $\mathcal{T}_y \subseteq \mathcal{T}$ and $\Sigma_y \subseteq \Sigma$ constitute the set of all possible system runs and the set of all of its possible data flow states of system $y \in \mathcal{Y}$, respectively:

$$\forall y \in \mathcal{Y}: \quad \mathcal{T}_y \subseteq \mathcal{T} : \mathbb{N} \rightarrow \mathbb{P}(\mathcal{S}_y)$$

$$\text{and} \quad \Sigma_y \subseteq \Sigma : (\mathcal{C}_y \rightarrow \mathbb{P}(\mathcal{D})) \times (\mathcal{C}_y \rightarrow \mathbb{P}(\mathcal{C}_y)) \times (\mathcal{I}_y \rightarrow \mathcal{C}_y)$$

This concept of individual systems and their individual event traces and data flow states are the fundamentals to build models and infrastructures that allow for the enforcement of distributed data usage policies. In addition, a corresponding system specification, formalized as state transitions \mathcal{R} (cf. Section 2.3 for an example), must be defined.

The following section describes how the union of the above individual systems re-assembles the entire distributed system.

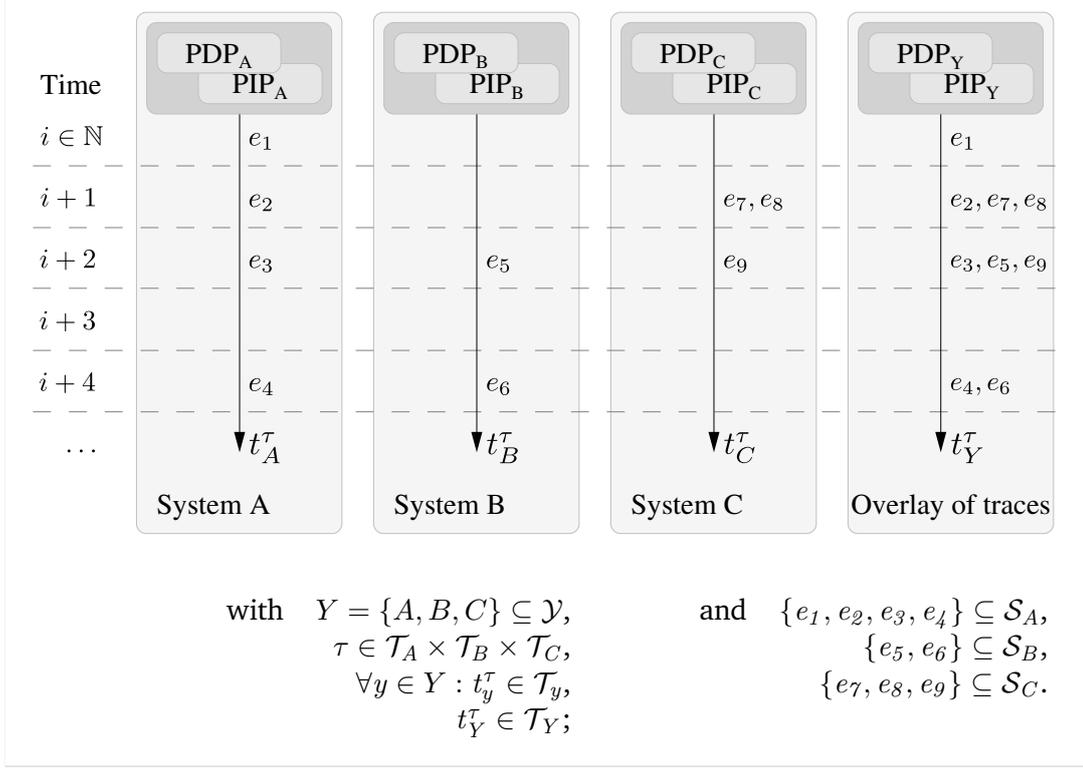
3.1.2 Reassembling the Distributed System

In practice, aforementioned systems run in parallel and produce independent system traces and data flow states. E.g., the PDP of system $y \in \mathcal{Y}$, PDP_y , observes trace $t_y \in \mathcal{T}_y$, while the corresponding PIP, PIP_y , observes data flow states $\sigma_{t_y} \in \Sigma_y$. Assuming sufficiently synchronized system clocks as further discussed in Section 4.3.1, it is the union of these local observations that one single global PDP, $\text{PDP}_\mathcal{Y}$, and PIP, $\text{PIP}_\mathcal{Y}$, as presumed in Section 2.1, would observe. This correlation between decentrally observed traces and data flow states and the observations of a central PDP/PIP is formalized in the following.

First, for a set of systems $Y \subseteq \mathcal{Y}$, the set of system events $\mathcal{S}_Y \subseteq \mathcal{S}$, the set of traces $\mathcal{T}_Y \subseteq \mathcal{T}$, the set of containers $\mathcal{C}_Y \subseteq \mathcal{C}$, and the set of identifiers $\mathcal{I}_Y \subseteq \mathcal{I}$ are defined as the union of the corresponding sets of systems $y \in Y$:

$$\forall Y \subseteq \mathcal{Y}: \mathcal{S}_Y = \bigcup_{y \in Y} \mathcal{S}_y \wedge \mathcal{T}_Y = \bigcup_{y \in Y} \mathcal{T}_y \wedge \mathcal{C}_Y = \bigcup_{y \in Y} \mathcal{C}_y \wedge \mathcal{I}_Y = \bigcup_{y \in Y} \mathcal{I}_y$$

Combining Concurrently Executing System Traces. Let \prod denote the Cartesian product. Then $\tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y$ is a tuple of traces of all systems and $t_y^\tau \in \mathcal{T}_y$ refers to the

Figure 3.2: Three systems running in parallel and the overlay of their traces.

overlay of traces tuple's trace of system $y \in \mathcal{Y}$. In order to reason about the concurrent execution of traces of multiple systems, the *overlay* of a set of traces of systems $Y \subseteq \mathcal{Y}$, $t_Y^\tau \in \mathcal{T}_Y$, is defined. The intuition is that for each timestep $i \in \mathbb{N}$ the set of system events observed in the set of systems Y corresponds to the union of the system events observed in the single systems. This is depicted in Figure 3.2 and formalized as follows:

$$\forall Y \subseteq \mathcal{Y}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, t_Y^\tau \in \mathcal{T}_Y : t_Y^\tau(i) = \bigcup_{y \in Y} t_y^\tau(i)$$

Hence, t_Y^τ resembles the trace that one single central PDP would have observed and it can be deduced by combining the observations of the independent distributed PDPs.

global data flow state **Combining Distributed Data Flow States.** Similar to the overlay of event traces, the systems' individual data flow states must be combined to a single *global data flow state* in order to reason about the behavior of the distributed system as a whole in correspondence with the monolithic model described in Section 2.1.

Consequently, the set of all possible states of the set of systems $Y \subseteq \mathcal{Y}$ is

$$\forall Y \subseteq \mathcal{Y} : \Sigma_Y = (\mathcal{C}_Y \rightarrow \mathbb{P}(\mathcal{D})) \times (\mathcal{C}_Y \rightarrow \mathbb{P}(\mathcal{C}_Y)) \times (\mathcal{I}_Y \rightarrow \mathcal{C}_Y)$$

and the combined global data flow state of systems $Y \subseteq \mathcal{Y}$ at time $i \in \mathbb{N}$ is

$$\begin{aligned} \forall Y \subseteq \mathcal{Y}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, y \in Y, t_Y^\tau \in \mathcal{T}_Y, t_y^\tau \in \mathcal{T}_y, \\ \sigma_{t_Y^\tau}^i \in \Sigma_Y, \sigma_{t_y^\tau}^i \in \Sigma_y, c \in \mathcal{C}_y, j \in \mathcal{I}_y : \\ \sigma_{t_Y^\tau}^i . s(c) = \sigma_{t_y^\tau}^i . s(c) \\ \wedge \sigma_{t_Y^\tau}^i . a(c) = \sigma_{t_y^\tau}^i . a(c) \\ \wedge \sigma_{t_Y^\tau}^i . n(j) = \sigma_{t_y^\tau}^i . n(j) \end{aligned}$$

The above definitions allow to investigate and to reason about the independent behavior of individual systems as well as their interactions and interrelations. Hence, in the following it is possible to argue about the distributed system both from the perspective of single individual systems as well as from a global perspective. The subsequent sections leverage these definitions to achieve data flow tracking across systems (Section 3.2) and the efficient enforcement of data usage policies if data and events are distributed (Section 3.3).

3.2 Cross-System Data Flow Tracking

Contents of this section have been published in [94].

As shown in Section 2.3 and [119, 168, 170, 174, 188, 216], the generic data flow model presented in Section 2.1.2 has been instantiated to track data flows within single systems for the sake of enforcing data-centric usage control policies, i.e. policies that protect *all* existing representations of some data rather than only particular ones (cf. Section 2.1.2). Along these lines, cross-system data flow tracking addresses the challenge to be aware of all representations of some data even if those representations are distributed across multiple systems: Whenever usage controlled data is exchanged between systems, this flow of data must be recorded in order to assure the data's future protection.

Hence, this section provides mechanisms for tracking data flows across systems in line with **RQ1** (cf. Section 1.1.1). On the basis of this approach, the distributed PIPs are aware which protected data remains in which containers throughout the entire distributed system at each point in time. The mechanisms are generic in that they allow to track data flows independent of the application and application-protocol being used; they are transparent since neither the applications nor the operating system are aware that data flow tracking is taking place. Such genericity and transparency are essential because in the real world data might be shared using a multitude of applications and protocols. E.g., in the example of the insurance company, data is propagated in-between systems using applications and protocols such as email, web browsing, file transfer, and possibly proprietary protocols.

3.2.1 A Generic Model for Cross-System Data Flow Tracking

State-of-the-art cross-system communication methods, such as the Internet Protocol (IP) in version 4 (IPv4) and version 6 (IPv6) and the Media Access Control Protocol (MAC), make use of addresses to identify participants of a communication network. Since a system may feature multiple network communication interfaces, multiple such addresses may be assigned to each system. This also reflects the fact that a system as defined in Section 3.1 may be distributed in itself. Within this thesis, *addresses* \mathcal{A} are assumed to be globally unique: no address may be assigned to more than one system over time. While not tamper-proof, such uniqueness of addresses is designed into MAC addresses and IPv6. Further, systems can refer to themselves using reserved addresses such as ‘127.0.0.1’ (IPv4) and ‘::1’ (IPv6). Those special addresses are referred to as *localhost* ($lo \in \mathcal{A}$). Using these terms, the set of systems $\mathcal{Y} \subseteq \mathbb{P}(\mathcal{A} \setminus \{lo\})$ as introduced in Section 3.1 can be formally characterized as follows:

$$\forall y_1, y_2 \in \mathcal{Y} \subseteq \mathbb{P}(\mathcal{A} \setminus \{lo\}) : y_1 \neq \emptyset \wedge y_2 \neq \emptyset \wedge (y_1 \cap y_2 \neq \emptyset \implies y_1 = y_2)$$

Hence, each system y is defined by its set of unique addresses and $\forall y \in \mathcal{Y} : lo \notin y$.

Each principal $p \in \mathcal{P}$ is associated with exactly one system $y \in \mathcal{Y}$ in which it may invoke system events from set \mathcal{S}_y . At the same time, system y might host multiple principals $P \subseteq \mathcal{P}$. For a principal $p \in \mathcal{P}$, its corresponding system is denoted $y_p \in \mathcal{Y}$. In order to model the technical difference between address $lo \in \mathcal{A}$ and addresses $\mathcal{A} \setminus \{lo\}$, i.e. the fact that lo can only be used to communicate with principals hosted on the same system, function $reach : \mathcal{Y} \times \mathcal{A} \rightarrow \mathbb{P}(\mathcal{Y})$ is defined. Given a system $y \in \mathcal{Y}$ and an address $a \in \mathcal{A}$, $reach$ returns all systems with which communication via the specified address is possible:

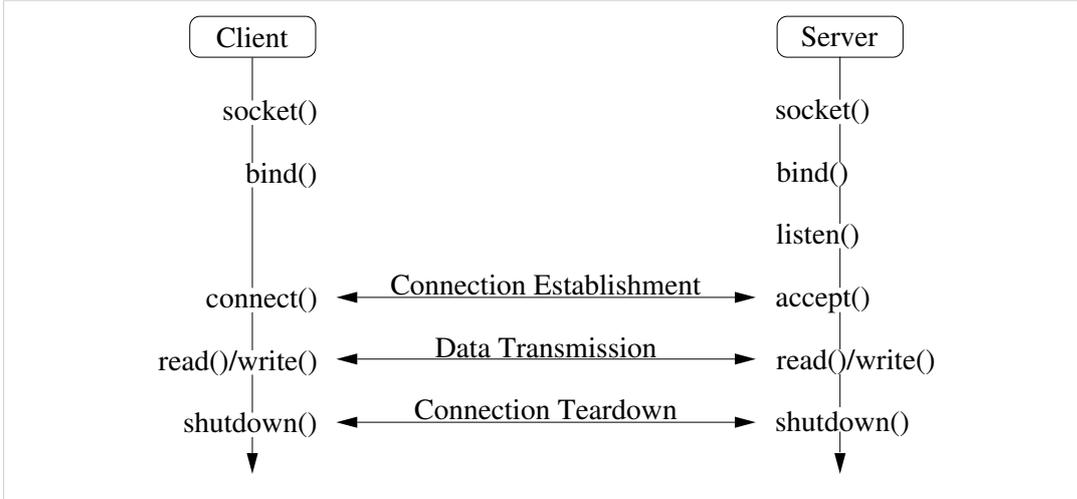
$$\forall y \in \mathcal{Y}, a \in \mathcal{A} : reach(y, a) = \begin{cases} \{y\} & \text{if } a = lo \\ \mathcal{Y} & \text{otherwise} \end{cases}$$

When considering more complex techniques such as NAT (Network Address Translation) [8] or DHCP (Dynamic Host Configuration Protocol) [48], i.e. if the initial assumption about the uniqueness of addresses does not hold, function $reach$ must be adjusted accordingly—possibly incorporating a temporal dimension.

The following section shows an instantiation of this generic model, allowing to track cross-system data flows for TCP/IP networking at the operating system layer. As such, the presented solution integrates seamlessly into the intra-system data flow tracking technology described in Section 2.3.

3.2.2 Data Flow Tracking for TCP/IP

Since all major application-level protocols, such as HTTP (web browsing), FTP (file transfer), SMTP, IMAP (both email), and SSH (general purpose secure remote shell),

Figure 3.3: Sequence of TCP-related system calls.

build upon TCP/IP (Transmission Control Protocol / Internet Protocol), this thesis instantiates cross-system data flow tracking for the latter protocol. Building on the instantiation of the generic data flow model for Unix-like systems described in Section 2.3, this instantiation allows for generic, transparent, and application-protocol independent cross-system data flow tracking—as long as TCP/IP is used as the underlying communication protocol. Hence, in the following, transition relation \mathcal{R} is defined for TCP/IP-related system calls.

Identifiers and Containers. As in Section 2.3, principals invoking system events are processes \mathcal{P}_{Proc} , which, in a distributed environment, are identified by a tuple consisting of an identifier for their system $\mathcal{Y} = \mathcal{I}_{Sys} \subseteq \mathcal{I}$ and their system-relative process id \mathcal{I}_{Pid} , hence $(\mathcal{I}_{Sys} \times \mathcal{I}_{Pid}) \subseteq \mathcal{I}$. Since processes communicate by writing to and reading from communication endpoints called sockets, network sockets \mathcal{C}_{Sock} are considered containers, $\mathcal{C}_{Sock} \subseteq \mathcal{C}$, which can be identified via process-relative file descriptors \mathcal{I}_{Fdsc} , $(\mathcal{I}_{Sys} \times \mathcal{I}_{Pid} \times \mathcal{I}_{Fdsc}) \subseteq \mathcal{I}$. Further, processes also refer to network sockets using addresses $\mathcal{A} = \mathcal{I}_{Addr} \subseteq \mathcal{I}$ and ports $\mathcal{I}_{Port} \subseteq \mathcal{I}$: a socket is uniquely identified by the address and port of the sender, and the address and port of the receiver, called *local socket name* and *remote socket name*, respectively. Since a socket using address $lo \in \mathcal{A}$ is only reachable from within the same system, the set of network socket identifiers is system-relative: $(\mathcal{I}_{Sys} \times ((\mathcal{I}_{Addr} \times \mathcal{I}_{Port}) \times (\mathcal{I}_{Addr} \times \mathcal{I}_{Port}))) \subseteq \mathcal{I}$. Besides the system calls described in Section 2.3, additional events to consider are all system calls related to networking such as *socket*, *bind*, *listen*, *accept*, *connect*, and *shutdown*. Transition relations \mathcal{R} for these events in the context of TCP/IP are explained in the following; Figure 3.3 depicts the sequence of system calls for TCP/IP connection establishment, data transmission and connection teardown.

TCP Connection Establishment. First, each communication partner (i.e. the client and the server process) creates a new unconnected socket for connection-based com-

munication (i.e. TCP/IP) by issuing system call *socket* with parameter `SOCK_STREAM` (parameter *type*). This call creates and returns a new file descriptor $fd \in \mathcal{I}_{Fdsc}$ (parameter *ret*) identifying the newly created socket container $c_s \in \mathcal{C}_{Sock}$, which is accessible by the calling process $pid \in \mathcal{I}_{Pid}$ (parameter *proc*) within system $y \in \mathcal{Y}$ (parameter *sys*):

$$\begin{aligned} & \forall \sigma, \sigma' \in \Sigma, \forall y \in \mathcal{I}_{Sys}, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc}, \forall c_s \in \mathcal{C}_{Sock} : \\ & (\sigma, (socket, \{(obj, c_s), (sys, y), (proc, pid), \\ & \quad (type, SOCK_STREAM), (ret, fd)\}), \sigma') \in \mathcal{R} \\ \implies & \sigma'.s = \sigma.s \\ & \wedge \sigma'.a = \sigma.a \\ & \wedge \sigma'.n = \sigma.n[(y, pid, fd) \leftarrow c_s] \end{aligned}$$

After the socket has been created, each communication partner binds a local socket name, i.e. an IP address and a port, to it using system call *bind*. This step is particularly important for the server process, because it will be waiting for incoming connections and its socket name must therefore be fixed and known by potential clients. After that, the server marks its socket as passive using system call *listen*. Note, that a listening socket may neither initiate connections nor be part of an actual communication channel. Despite their importance for connection establishment, *bind* and *listen* are not formalized in terms of state transitions as they do not change the system's data flow state. In particular, the result of the execution of *bind* (i.e. the assignment of an IP address and port to a socket) can be easily retrieved in the presence of the subsequent system calls *accept* and *connect*.

The server process then issues system call *accept* on its passive socket, effectively making the socket listen for incoming connections. *accept* does not return until an actual connection establishment request to that socket has been made. The client process then initializes the actual connection establishment using system call *connect*. Notably, upon each of those two system calls some information necessary for modeling the connection establishment is not available. In addition, the fact that the return order of *accept* and *connect* is nondeterministic complicates modeling of the connection establishment, since there exists a cyclic dependency between those two system calls. Consequently, connection establishment can only be modeled once the second of these two system calls returns. When modeling the two system calls *accept* and *connect* in the following, for each of them it is assumed that it returns second. Section 4.2.1 describes how the implementation copes with this assumption.

Upon successful return of *accept*, a new socket c_s has been created by the underlying operating system and this newly created socket constitutes the connection to the client's socket that initiated connection establishment. The server's passive and listening socket remains passive and listening. The newly created socket c_s is identified using the returned file descriptor $fd \in \mathcal{I}_{Fdsc}$ (parameter *ret*). Further return parameters of *accept* are the remote socket name $(a_c, o_c) \in (\mathcal{I}_{Addr} \times \mathcal{I}_{Port})$ (param-

eter *remotesock*) (i.e. the name of the client's connected socket), and the server's local socket name $(a_s, o_s) \in (\mathcal{I}_{Addr} \times \mathcal{I}_{Port})$ (parameter *localsock*). The connection establishment is modeled by creating an alias from the server's socket container to the client's socket container. Note, however, that the latter can only be retrieved via $\sigma.n((y, ((a_c, o_c), (a_s, o_s))))$ if *connect* did happen before *accept*.

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall y \in \mathcal{I}_{Sys}, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc}, \forall c_s \in \mathcal{C}_{Sock}, \\
& \quad \forall a_s, a_c \in \mathcal{I}_{Addr}, \forall o_s, o_c \in \mathcal{I}_{Port} : \\
& (\sigma, (accept, \{(obj, c_s), (sys, y), (proc, pid), (ret, fd), \\
& \quad (localsock, (a_s, o_s)), (remotesock, (a_c, o_c))\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s \\
& \quad \wedge \sigma'.a = \sigma.a [c_s \leftarrow \sigma.n((y, ((a_c, o_c), (a_s, o_s))))] \\
& \quad \wedge \sigma'.n = \sigma.n [(y, pid, fd) \leftarrow c_s; (z, ((a_s, o_s), (a_c, o_c))) \leftarrow c_s]_{z \in reach(y, a_s)}
\end{aligned}$$

For system call *connect*, parameters are the file descriptor $fd \in \mathcal{I}_{Fdsc}$ (parameter *fdscr*) of the client's socket, as well as IP address $a_s \in \mathcal{I}_{Addr}$ and port $o_s \in \mathcal{I}_{Port}$ of the server's listening socket (parameter *remotesock*). If the client's socket has not been bound explicitly before, *connect* does an implicit call to *bind*. Once the connection to (a_s, o_s) has been successfully established, *connect* returns. At this point, the client's local socket name $(a_c, o_c) \in (\mathcal{I}_{Addr} \times \mathcal{I}_{Port})$ (parameter *localsock*) has been provided by the operating system. *connect* is modeled by creating an alias from the client's connected socket to the corresponding server's remote connected socket. Effectively, this last step aliases the two sockets bidirectionally. Note, however, that the server's remote connected socket only exists if *accept* did happen before *connect*; it can then be retrieved via $\sigma.n((y, ((a_s, o_s), (a_c, o_c))))$.

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall y \in \mathcal{I}_{Sys}, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc}, \\
& \quad \forall a_s, a_c \in \mathcal{I}_{Addr}, \forall o_s, o_c \in \mathcal{I}_{Port} : \\
& (\sigma, (connect, \{(obj, \sigma.n((y, pid, fd))), (sys, y), (proc, pid), (fdscr, fd), \\
& \quad (localsock, (a_c, o_c)), (remotesock, (a_s, o_s))\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s \\
& \quad \wedge \sigma'.a = \sigma.a [\sigma.n((y, pid, fd)) \leftarrow \sigma.n((y, ((a_s, o_s), (a_c, o_c))))] \\
& \quad \wedge \sigma'.n = \sigma.n [(z, ((a_c, o_c), (a_s, o_s))) \leftarrow \sigma.n((y, pid, fd))]_{z \in reach(y, a_c)}
\end{aligned}$$

TCP Data Transmission. Once the connection has been established, the processes may exchange any kind of information by writing to and reading from the network sockets using a variety of system calls such as *write* and *read*. Modelling sending and receiving of data corresponds to writing to and reading from any other file descriptor as described in Section 2.3. Other system calls for sending are *sendmsg*, *pwritev*, *pwrite*, *writev*, *send*, and *sendto*; system calls for reading are *recvmsg*, *preadv*, *pread*, *readv*, *recv*, and *recvfrom*. Their state transitions are analogous to *write* and *read* as described in Section 2.3.

Different from the above system calls, system call *sendfile* copies data directly from one file descriptor to another. Hence, $fd_i \in \mathcal{I}_{Fdsc}$ (parameter $fdscr_i$) is a file descriptor opened for reading, while $fd_o \in \mathcal{I}_{Fdsc}$ (parameter $fdscr_o$) is a file descriptor opened for writing. Analogous to *read* and *write*, the data is propagated to all aliased containers.

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall y \in \mathcal{I}_{Sys}, \forall pid \in \mathcal{I}_{Pid}, \forall fd_i, fd_o \in \mathcal{I}_{Fdsc} : \\
& (\sigma, (sendfile, \{(obj, \sigma.s((y, pid, fd_o))), (sys, y), (proc, pid), \\
& \quad (fdscr_i, fd_i), (fdscr_o, fd_o)\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s[t \leftarrow \sigma.s(t) \cup \sigma.s(\sigma.n((y, pid, fd_i)))]_{t \in \sigma.a^*(\sigma.n((y, pid, fd_o)))} \\
& \quad \wedge \sigma'.a = \sigma.a \\
& \quad \wedge \sigma'.n = \sigma.n
\end{aligned}$$

TCP Connection Teardown. Finally, the communication channel is shut down. System calls *shutdown*, *close*, and *exit* cause a, potentially partial, connection teardown.

Using system call *shutdown*, a process may shut down all or part of the connection constituted by the socket identified by file descriptor $fd \in \mathcal{I}_{Fdsc}$ (parameter $fdscr$). Parameter *how* describes how the socket is shutdown: SHUT_RD disallows further receptions, SHUT_WR disallows further transmission, and SHUT_RDWR forbids further receptions and transmissions. This is modeled as follows.

In case of SHUT_RD, the socket container is emptied and all aliases to it are deleted, effectively making any further *read* on that socket not propagate any data to the reading process:

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall y \in \mathcal{I}_{Sys}, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc} : \\
& (\sigma, (shutdown, \{(obj, \sigma.s((y, pid, fd))), (sys, y), (proc, pid), \\
& \quad (fdscr, fd), (how, SHUT_RD)\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s[\sigma.n((y, pid, fd)) \leftarrow \emptyset] \\
& \quad \wedge \sigma'.a = \sigma.a[c \leftarrow \sigma.a(c) \setminus \{\sigma.n((y, pid, fd))\}]_{c \in C} \\
& \quad \wedge \sigma'.n = \sigma.n
\end{aligned}$$

In case of SHUT_WR, all aliases from the socket container are deleted, effectively making any further *write* on that socket not propagate any data to the socket to which the connection was originally established.

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall y \in \mathcal{I}_{Sys}, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc} : \\
& (\sigma, (shutdown, \{(obj, \sigma.s((y, pid, fd))), (sys, y), (proc, pid), \\
& \quad (fdscr, fd), (how, SHUT_WR)\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s \\
& \quad \wedge \sigma'.a = \sigma.a[\sigma.n((y, pid, fd)) \leftarrow \emptyset] \\
& \quad \wedge \sigma'.n = \sigma.n
\end{aligned}$$

In case of SHUT_RDWR, the socket container is emptied and all aliases to and from it are deleted; additionally, all its identifiers of type $(\mathcal{I}_{Sys} \times ((\mathcal{I}_{Addr} \times \mathcal{I}_{Port}) \times (\mathcal{I}_{Addr} \times$

\mathcal{I}_{Port})) are deleted:

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall y \in \mathcal{I}_{Sys}, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc} : \\
& (\sigma, (shutdown, \{(obj, \sigma.s((y, pid, fd))), (sys, y), (proc, pid), \\
& \quad (fdscr, fd), (how, SHUT_RDWR)\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s[\sigma.n((y, pid, fd)) \leftarrow \emptyset] \\
& \quad \wedge \sigma'.a = \sigma.a[\sigma.n((y, pid, fd)) \leftarrow \emptyset; c \leftarrow \sigma.a(c) \setminus \{\sigma.n((y, pid, fd))\}]_{c \in \mathcal{C}} \\
& \quad \wedge \sigma'.n = \sigma.n[x \leftarrow nil]_{x \in \{z \in (\mathcal{I}_{Sys} \times ((\mathcal{I}_{Addr} \times \mathcal{I}_{Port}) \times (\mathcal{I}_{Addr} \times \mathcal{I}_{Port}))) \mid \sigma.n(z) = \sigma.n((y, pid, fd))\}}
\end{aligned}$$

A process $pid \in \mathcal{I}_{Pid}$ on system $y \in \mathcal{Y}$ may close a file descriptor $fd \in \mathcal{I}_{Fdsc}$ using system call *close* as described in Section 2.3. Additionally, if (y, pid, fd) is the last remaining file descriptor for socket $c_s = \sigma.n((y, pid, fd))$ (i.e. if $\nexists pid' \in \mathcal{I}_{Pid}, fd' \in \mathcal{I}_{Fdsc} : (y, pid, fd) \neq (y, pid', fd') \wedge \sigma.n((y, pid', fd')) = \sigma.n((y, pid, fd)) = c_s$), then the connection constituted by c_s is implicitly shut down. This is modeled by an implicit *shutdown* with parameter SHUT_RDWR. If a process exits using system call *exit*, all of its file descriptors are closed alike.

Other system calls. Note that the presence of event parameter *sys* necessitates the redefinition of event semantics for certain system calls introduced in Section 2.3. As one example, the redefined event semantics for system call *write* follows. Redefinition of the event semantics for other system calls is analogous.

$$\begin{aligned}
& \forall \sigma, \sigma' \in \Sigma, \forall y \in \mathcal{I}_{Sys}, \forall pid \in \mathcal{I}_{Pid}, \forall fd \in \mathcal{I}_{Fdsc} : \\
& (\sigma, (write, \{(obj, \sigma.n((y, pid, fd))), (sys, y), (proc, pid), (fdscr, fd)\}), \sigma') \in \mathcal{R} \\
& \implies \sigma'.s = \sigma.s[t \leftarrow \sigma.s(t) \cup \sigma.s(\sigma.n((y, pid)))]_{t \in \sigma.a^*(\sigma.n((y, pid, fd)))} \\
& \quad \wedge \sigma'.a = \sigma.a \\
& \quad \wedge \sigma'.n = \sigma.n
\end{aligned}$$

The above definitions of state transition \mathcal{R} for networking-related system calls enable the tracking of data flows both within and across systems. The applications themselves are not aware that such data flow tracking is being performed. Note that all of the above state transitions can be applied locally and individually by the server's and the client's PIP. In particular, this is the case for system calls *accept* and *connect*, which bidirectionally alias the network socket containers of the two remote systems. Details on how this is technically achieved are provided in Section 4.2.1.

At the model level, system calls *read*, *write*, *sendfile*, and equivalent, propagate all data written to socket containers to the corresponding remote connected socket. However, to technically track data flows in the presence of those system calls, information about the data flow state must be exchanged between the two remote PIPs. How such cross-system data flow tracking is technically achieved is explained in Section 4.2.2. Further, it will be shown how data usage policies are transferred to and deployed at the remote system.

3.3 Coordinating Policy Decisions Across Systems

Contents of this section have been published in [95].

Once data and their corresponding policies have been disseminated to several systems, the PDPs within those systems are expected to consistently enforce those policies at all times (RQ2, Section 1.1.2). This is of no further challenge for policies that can be individually decided by each single PDP, such as ‘delete this data after 30 days’ and ‘do not edit this document [using editor X]’. However, if policies refer to data, events, or system states of multiple systems, then each PDP’s decisions might depend on past decisions and observations of other PDPs and PIPs. Hence, the PDPs and PIPs must coordinate their decisions and exchange corresponding information. Naively, in such situations all PDPs/PIPs could disclose all of their knowledge to all other PDPs/PIPs of the entire distributed system (i.e. to all systems \mathcal{Y}). Note that all global policies of the running example (cf. Section 1.5 and Table 2.2 on page 34) are of such a kind.

To decrease the number of systems with which such coordination is actually performed, Section 3.3.1 provides methods to approximate the set of systems relevant for evaluating a given policy. While naively each PDP/PIP could then disclose all of its knowledge to all other ‘relevant’ PDPs/PIPs, Section 3.3.2 formally analyzes in which cases such exchange of information between relevant PDPs/PIPs can be safely omitted without compromising consistent policy enforcement.

3.3.1 Identifying Relevant Systems

Towards distributed enforcement of data usage policies, this section identifies the set of systems relevant for evaluating a policy p given as ECA rule. By and large, p ’s condition $\varphi_p \in \Phi$ is the most important and complex part of p that must be evaluated, which is why this section focuses on finding all systems that *potentially* contribute to the evaluation of φ_p at a given point in time $i \in \mathbb{N}$. Hence, given a set of concurrently executing traces $\tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y$, function *relevant* : $\Phi \times \mathbb{N} \times \prod \mathcal{T} \rightarrow \mathbb{P}(\mathcal{Y})$ is defined to return all systems that are potentially relevant for evaluating φ_p at time i . In particular, if $|\text{relevant}(\varphi_p, i, \tau)| \leq 1$, then no coordination is needed in order to evaluate φ_p . Before defining *relevant*, three auxiliary functions are defined:

(1) $\text{know}C : \mathbb{P}(C) \rightarrow \mathbb{P}(\mathcal{Y})$ returns for a given set of containers the set of all systems that ‘know’ at least one of the specified containers, meaning that a system’s PIP is responsible for the management of this container. Formally:

$$\forall C \subseteq \mathcal{C} : \text{know}C(C) = \{y \in \mathcal{Y} \mid \mathcal{C}_y \cap C \neq \emptyset\}$$

Note that this definition does not impose any constraints on the input set $C \subseteq \mathcal{C}$. Hence, C might contain containers that are attributed to different systems: $\forall y_1, y_2 \in \mathcal{Y}, C \subseteq \mathcal{C}, c_1 \in \mathcal{C}_{y_1}, c_2 \in \mathcal{C}_{y_2} : y_1 \neq y_2 \wedge \{c_1, c_2\} \subseteq C \implies \{y_1, y_2\} \subseteq \text{know}C(C)$. Further: Because \mathcal{C}_y (for all $y \in \mathcal{Y}$) denotes the set of *all* containers that might *ever* exist within system y , $\text{know}C$ is not parameterized in a temporal dimension.

(2) $knowD : \mathbb{P}(\mathcal{D}) \times \mathbb{N} \times \prod \mathcal{T} \rightarrow \mathbb{P}(\mathcal{Y})$ returns for a given set of data items, a point in time $i \in \mathbb{N}$, and a tuple of concurrently executing traces $\tau \in \prod \mathcal{T}$ the set of systems in which there exists a container that contains at least one of the specified data items:

$$\forall D \subseteq \mathcal{D}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y :$$

$$knowD(D, i, \tau) = \{y \in \mathcal{Y} \mid \exists c \in \mathcal{C}_y, t_y^\tau \in \mathcal{T}_y, \sigma_{t_y^\tau}^i \in \Sigma_y : D \cap \sigma_{t_y^\tau}^i.s(c) \neq \emptyset\}$$

Different to $knowC$, function $knowD$ is parameterized in time. This is because the addressed data D does not statically remain within a fixed set of systems but keeps being propagated and deleted across/within different systems. Consequently, at all times the result of $knowD$ depends on the systems' *current* data flow states.

(3) $happens : \mathcal{E} \times \mathbb{N} \times \prod \mathcal{T} \rightarrow \mathbb{P}(\mathcal{Y})$ returns for an event $e \in \mathcal{E}$, a point in time $i \in \mathbb{N}$, and a tuple of concurrently executing traces $\tau \in \prod \mathcal{T}$ the set of systems in which an event refining e happens:

$$\forall e \in \mathcal{E}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y :$$

$$happens(e, i, \tau) = \{y \in \mathcal{Y} \mid \exists t_y^\tau \in \mathcal{T}_y, e' \in t_y^\tau(i), \sigma_{t_y^\tau}^i \in \Sigma_y :$$

$$(e', \sigma_{t_y^\tau}^i) \text{ refines}_\Sigma e\}$$

The reason for parameterizing $happens$ in time is similar to the one provided for function $knowD$: Due to the definition of $refines_\Sigma$ (cf. Section 2.1.2), the system's *current* data flow state must be considered in order to determine whether an event refines another.

Using these auxiliary functions, $relevant : \Phi \times \mathbb{N} \times \prod \mathcal{T} \rightarrow \mathbb{P}(\mathcal{Y})$ is defined to return the set of all *potentially* relevant systems for evaluating $\varphi \in \Phi$ at time $i \in \mathbb{N}$ given a tuple of concurrently executing traces $\tau \in \prod \mathcal{T}$. The central observation behind the definition of $relevant$ is that it depends on the condition φ , the time i , and the traces τ which systems are relevant to evaluate φ . Considering φ as an expression tree (cf. Section 2.2.3), it is essentially the tree's leaves (i.e. operators \mathcal{E} , *isCombined*, *isNotIn*, *isMaxIn*, and *repmIn*) that determine which systems are of interest. The intuition

behind the definition of *relevant* is explained after providing the formal definition.

$$\begin{aligned}
& \forall \varphi \in \Phi, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y : \text{relevant}(\varphi, i, \tau) = \{y \in Y \mid \\
& \quad ((\varphi = \text{true} \vee \varphi = \text{false}) \wedge Y = \emptyset) \\
& \quad \vee \exists e \in \mathcal{E} \bullet (\varphi = e \wedge Y = \text{happens}(e, i, \tau)) \\
& \quad \vee \exists d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C} \bullet ((\varphi = \text{isNotIn}(d, C) \vee \varphi = \text{isMaxIn}(d, m, C)) \\
& \quad \quad \wedge Y = \text{knowD}(\{d\}, i, \tau) \cap \text{knowC}(C)) \\
& \quad \vee \exists d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C} \bullet (\varphi = \text{isCombined}(d_1, d_2, C) \\
& \quad \quad \wedge Y = \text{knowD}(\{d_1\}, i, \tau) \cap \text{knowD}(\{d_2\}, i, \tau) \cap \text{knowC}(C)) \\
& \quad \vee \exists \alpha \in \Phi \bullet (\varphi = \text{not}(\alpha) \wedge Y = \text{relevant}(\alpha, i, \tau)) \\
& \quad \vee \exists \alpha, \beta \in \Phi \bullet ((\varphi = \alpha \text{ and } \beta \vee \varphi = \alpha \text{ or } \beta) \\
& \quad \quad \wedge Y = \text{relevant}(\alpha, i, \tau) \cup \text{relevant}(\beta, i, \tau)) \\
& \quad \vee \exists \alpha, \beta \in \Phi \bullet (\varphi = \alpha \text{ since } \beta \\
& \quad \quad \wedge Y = \bigcup_{j=0}^i (\text{relevant}(\alpha, j, \tau) \cup \text{relevant}(\beta, j, \tau))) \\
& \quad \vee \exists \alpha \in \Phi, j \in \mathbb{N} \bullet (\varphi = \alpha \text{ before } j \wedge Y = \text{relevant}(\alpha, i - j, \tau)) \\
& \quad \vee \exists j, m \in \mathbb{N}, e \in \mathcal{E} \bullet (\varphi = \text{repmin}(j, m, e) \\
& \quad \quad \wedge Y = \bigcup_{k=0}^{\min\{i, j\}-1} \text{happens}(e, i - k, \tau)) \\
& \quad \left. \vphantom{\prod_{y \in \mathcal{Y}} \mathcal{T}_y} \right\}
\end{aligned}$$

The rationale behind this definition of *relevant* is as follows: If $\varphi = \text{true}$ or $\varphi = \text{false}$, then this trivial formula can always be evaluated locally to *true* or *false*, respectively. For $\varphi = e$, relevant systems are exactly those systems in which an event refining e happens at the current timestep as defined by *happens*. For state-based operators *isNotIn*(d, C) and *isMaxIn*(d, m, C), the set of relevant systems is defined by all systems that know data d and at least one of the containers in C . This is similar for *isCombined*(d_1, d_2, C), in which case, however, relevant systems are only those systems that know both data d_1 and data d_2 . For *repmin*(j, m, e), the set of relevant systems are those in which an event refining e has happened in the last j timesteps as defined by *happens*. For operators *not*, *and*, *or*, *since* and *before*, the set of relevant systems is determined by applying *relevant* recursively and unifying the corresponding result sets. Notably, for operator *since* all previous timesteps must be considered. For α *before* j , the set of relevant systems is evaluated by applying *relevant* j timesteps ago.

Note, however, that the set of systems $\text{relevant}(\varphi, i, \tau)$ is not minimal: Generally, $\text{relevant}(\varphi, i, \tau)$ contains more systems than actually required for the conclusive evaluation of condition φ . For example, consider an event $e \in \mathcal{E}$, a condition $\varphi = e$, a set of concurrently executing traces $\tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y$, two systems $y_1, y_2 \in \mathcal{Y}$, two concrete traces

$t_{y_1}^\tau \in \mathcal{T}_{y_1}, t_{y_2}^\tau \in \mathcal{T}_{y_2}$, a point in time $i \in \mathbb{N}$, and system states $\sigma_{t_{y_1}^\tau}^i \in \Sigma_{y_1}, \sigma_{t_{y_2}^\tau}^i \in \Sigma_{y_2}$. Further assume that events refining e happen within traces $t_{y_1}^\tau$ and $t_{y_2}^\tau$ at time i (i.e. $\exists e_1 \in t_{y_1}^\tau(i), e_2 \in t_{y_2}^\tau(i) : (e_1, \sigma_{t_{y_1}^\tau}^i) \text{refines}_\Sigma e \wedge (e_2, \sigma_{t_{y_2}^\tau}^i) \text{refines}_\Sigma e$). Then, $\text{relevant}(\varphi, i, \tau) = \{y_1, y_2\}$ (since $\text{relevant}(\varphi, i, \tau) = \text{relevant}(e, i, \tau) = \text{happens}(e, i, \tau) = \{y_1, y_2\}$ due to the definition of *happens*). However, in order to conclusively evaluate condition $\varphi = e$, the consideration of either of the singleton sets $\{y_1\}$ or $\{y_2\}$ would have sufficed. Consequently, $\text{relevant}(\varphi, i, \tau)$ overapproximates the set of systems that are relevant for evaluating φ .

As proven in Appendix A, the set of systems $\text{relevant}(\varphi, i, \tau)$ is sufficient to evaluate φ at time i given the set of concurrently executing traces τ . Systems $\mathcal{Y} \setminus \text{relevant}(\varphi, i, \tau)$ do not influence the evaluation of φ and adding any such system to the evaluation process does not change the evaluation result. Formally:

$$\forall \varphi \in \Phi, i \in \mathbb{N}, \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y : \\ (t_Y^\tau, i) \models \varphi \iff (t_{Y \cup X}^\tau, i) \models \varphi.$$

Due to these observations and for brevity, trace $t_{\text{relevant}(\varphi, i, \tau)}^\tau$ is also referred to as $t_{\mathcal{Y}}^\tau$, $t_{\text{relevant}(\varphi, i, \tau)}^\tau = t_{\mathcal{Y}}^\tau$. In particular, if $|\text{relevant}(\varphi, i, \tau)| \leq 1$ then no coordination is needed for the evaluation of φ .

Using function $\text{relevant}(\varphi, i, \tau)$, it is possible to retrieve all systems potentially relevant for evaluating condition φ at time i . Naively, an implementation could then reveal all necessary information, such as which events are happening and which state changes occur, to all corresponding systems $\text{relevant}(\varphi, i, \tau)$. While this already constitutes an improvement over revealing all information to all systems \mathcal{Y} as discussed in the beginning of Section 3.3, the following section identifies situations in which communication with less or even no systems is needed.

3.3.2 Omitting Unnecessary Communication

In general, coordination between systems is required if an ECA mechanism's triggering event is observed and if $|\text{relevant}(\varphi, i, \tau)| > 1$ for the ECA's condition $\varphi \in \Phi$, a point in time $i \in \mathbb{N}$, and a tuple of executing traces $\tau \in \prod \mathcal{T}$. When considering a set of systems $Y \subseteq \mathcal{Y}$, it is generally not possible to conclusively evaluate a given formula $\varphi \in \Phi$ at a certain point in time $i \in \mathbb{N}$, since evaluation of φ might depend on information unavailable within the set of systems Y . However, given $\tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y$ there are situations in which a formula $\varphi' \in \Phi$ can be deduced from $\varphi \in \Phi$ such that the set of systems $Y \subseteq \mathcal{Y}$ satisfies φ' at time i , $(t_Y^\tau, i) \models \varphi'$, and this satisfaction of φ' by systems Y implies the global satisfaction of φ , $(t_{\mathcal{Y}}^\tau, i) \models \varphi$. Formally: $(t_Y^\tau, i) \models \varphi' \implies (t_{\mathcal{Y}}^\tau, i) \models \varphi$. The same argument holds for the violation of formulas φ' and φ : $(t_Y^\tau, i) \not\models \varphi' \implies (t_{\mathcal{Y}}^\tau, i) \not\models \varphi$. Intuitively, this boils down to the implication seen above by negating formulas φ and φ' : $(t_Y^\tau, i) \models \text{not}(\varphi') \implies (t_{\mathcal{Y}}^\tau, i) \models \text{not}(\varphi)$. While those implications are not generally true, it depends on the characteristics of φ and φ' whether such an implication holds.

Formula Projections. The above implications can in fact be satisfied for so called *formula projections*. In a nutshell, the projection of some formula $\varphi \in \Phi$ for a set of systems $Y \subseteq \mathcal{Y}$ ‘hides’ parts of φ that are unknown within the set of systems Y , i.e. parts that can only be evaluated by systems $\mathcal{Y} \setminus Y$. Hence, the projection $\varphi_Y \in \Phi$ of formula φ for systems Y is defined as follows:

$$\begin{aligned}
& \forall \varphi \in \Phi, Y \subseteq \mathcal{Y}, \exists \varphi_Y : \\
& ((\varphi = \text{true} \vee \varphi = \text{false}) \implies \varphi_Y = \varphi) \\
& \forall \exists e \in \mathcal{E} \bullet (\varphi = e \implies \varphi_Y = e) \\
& \forall \exists d \in \mathcal{D}, C \subseteq \mathcal{C} \bullet (\varphi = \text{isNotIn}(d, C) \implies \varphi_Y = \text{isNotIn}(d, C \cap \mathcal{C}_Y)) \\
& \forall \exists d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C} \bullet (\varphi = \text{isCombined}(d_1, d_2, C) \\
& \implies \varphi_Y = \text{isCombined}(d_1, d_2, C \cap \mathcal{C}_Y)) \\
& \forall \exists d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C} \bullet (\varphi = \text{isMaxIn}(d, m, C) \\
& \implies \varphi_Y = \text{isMaxIn}(d, m, C \cap \mathcal{C}_Y)) \\
& \forall \exists \alpha \in \Phi \bullet (\varphi = \text{not}(\alpha) \implies \varphi_Y = \text{not}(\alpha_Y)) \\
& \forall \exists \alpha, \beta \in \Phi \bullet (\varphi = \alpha \text{ and } \beta \implies \varphi_Y = \alpha_Y \text{ and } \beta_Y) \\
& \quad \vee (\varphi = \alpha \text{ or } \beta \implies \varphi_Y = \alpha_Y \text{ or } \beta_Y) \\
& \quad \vee (\varphi = \alpha \text{ since } \beta \implies \varphi_Y = \alpha_Y \text{ since } \beta_Y) \\
& \forall \exists \alpha \in \Phi, j \in \mathbb{N} \bullet (\varphi = \alpha \text{ before } j \implies \varphi_Y = \alpha_Y \text{ before } j) \\
& \forall \exists j, m \in \mathbb{N}, e \in \mathcal{E} \bullet (\varphi = \text{repmIn}(j, m, e) \implies \varphi_Y = \text{repmIn}(j, m, e))
\end{aligned}$$

Using these policy projections, it is possible to define those situations in which no coordination between the PDPs and PIPs of different systems is needed. Those situations are formalized by predicate $Sat \subseteq \prod \mathcal{T} \times \mathbb{P}(\mathcal{Y}) \times \mathbb{N} \times \Phi$. In correspondence with the intuitive motivation provided above, $Sat(\tau, Y, i, \varphi)$ will be defined to hold *true* iff for the tuple of executing traces $\tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y$ and the set of systems $Y \subseteq \mathcal{Y}$, trace $t_Y^\tau \in \mathcal{T}_Y$ satisfies φ_Y at time $i \in \mathbb{N}$ ($(t_Y^\tau, i) \models \varphi_Y$) and if this implies global satisfaction of formula $\varphi \in \Phi$ at the same point in time ($(t_Y^\tau, i) \models \varphi$). Again, the same arguments hold for the violation of formula φ , which is expressed by negating φ :

$$\begin{aligned}
& \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, Y \subseteq \mathcal{Y}, i \in \mathbb{N}, \varphi \in \Phi : \\
& ((t_Y^\tau, i) \models \varphi_Y \wedge Sat(\tau, Y, i, \varphi) \implies (t_Y^\tau, i) \models \varphi) \\
& \wedge ((t_Y^\tau, i) \not\models \varphi_Y \wedge Sat(\tau, Y, i, \text{not}(\varphi)) \implies (t_Y^\tau, i) \not\models \varphi)
\end{aligned}$$

In the following, $\varphi \in \Phi$ is demanded to be given in disjunctive normal form (DNF). The reason is that for operators $o \in \{\mathcal{E}, \text{isCombined}, \text{isNotIn}, \text{isMaxIn}, \text{repmIn}\}$ it turns out that the results of Sat should be different depending on whether o is ‘encapsulated’ in an even or an odd number of negations. When demanding φ to be in DNF, then

negations only occur next to operators o . This renders counting of the negations within φ unnecessary. However, the cases $\varphi = o$ and $\varphi = \underline{not}(o)$ must be considered individually. Finally, $Sat \subseteq \prod \mathcal{T} \times \mathbb{P}(\mathcal{Y}) \times \mathbb{N} \times \Phi$ is defined as follows. Note that this definition only ‘enumerates’ cases for which $Sat = true$. Cases not mentioned yield $Sat = false$. Proofs of correctness, i.e. that the above implications hold, are provided in Appendix B.

$$\begin{aligned}
& \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, Y \subseteq \mathcal{Y}, i \in \mathbb{N}, \varphi \in \Phi : Sat(\tau, Y, i, \varphi) \\
& \iff \varphi = \underline{true} \vee \varphi = \underline{false} \\
& \quad \vee \text{relevant}(\varphi, i, \tau) \subseteq Y \\
& \quad \vee \exists e \in \mathcal{E} \bullet (\varphi = e) \\
& \quad \vee \exists d \in \mathcal{D}, C \subseteq \mathcal{C} \bullet (\varphi = \underline{not}(\underline{isNotIn}(d, C))) \\
& \quad \vee \exists d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C} \bullet (\varphi = \underline{isCombined}(d_1, d_2, C)) \\
& \quad \vee \exists d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C} \bullet (\varphi = \underline{not}(\underline{isMaxIn}(d, m, C))) \\
& \quad \vee \exists \alpha, \beta \in \Phi \bullet ((\varphi = \alpha \underline{and} \beta \wedge Sat(\tau, Y, i, \alpha) \wedge Sat(\tau, Y, i, \beta)) \\
& \quad \quad \vee (\varphi = \alpha \underline{or} \beta \wedge ((t_Y^\tau, i) \models \alpha_Y \wedge Sat(\tau, Y, i, \alpha) \\
& \quad \quad \vee (t_Y^\tau, i) \models \beta_M \wedge Sat(\tau, Y, i, \beta))) \\
& \quad \vee (\varphi = \alpha \underline{since} \beta \\
& \quad \quad \wedge ((\exists j \in [0, i] : (t_Y^\tau, j) \models \beta_Y \wedge Sat(\tau, Y, j, \beta) \\
& \quad \quad \wedge \forall k \in (j, i] : (t_Y^\tau, k) \models \alpha_Y \wedge Sat(\tau, Y, k, \alpha)) \\
& \quad \quad \vee (\forall k \in [0, i] : (t_Y^\tau, k) \models \alpha_Y \wedge Sat(\tau, Y, k, \alpha)))))) \\
& \quad \vee (\varphi = \underline{not}(\alpha \underline{since} \beta) \\
& \quad \quad \wedge ((\forall j \in [0, i] : (t_Y^\tau, j) \not\models \beta_Y \wedge Sat(\tau, Y, j, \underline{not}(\beta)) \\
& \quad \quad \wedge \exists k \in (j, i] : (t_Y^\tau, k) \not\models \alpha_Y \wedge Sat(\tau, Y, k, \underline{not}(\alpha))) \\
& \quad \quad \vee (\exists k \in [0, i] : (t_Y^\tau, k) \not\models \alpha_Y \wedge Sat(\tau, Y, k, \underline{not}(\alpha)))))) \\
& \quad \vee \exists \alpha \in \Phi, j \in \mathbb{N} \bullet (\varphi = \alpha \underline{before} j \wedge Sat(\tau, Y, i - j, \alpha) \\
& \quad \quad \vee \varphi = \underline{not}(\alpha \underline{before} j) \wedge Sat(\tau, Y, i - j, \underline{not}(\alpha))) \\
& \quad \vee \exists e \in \mathcal{E}, j, m \in \mathbb{N} \bullet (\varphi = \underline{repmIn}(j, m, e))
\end{aligned}$$

In summary, predicate Sat identifies situations in which no coordination between systems for policy enforcement is necessary despite the fact that $|\text{relevant}(\varphi, i, \tau)| > 1$. This fact will be leveraged in Chapter 4 in order to reduce the communication overhead between systems when enforcing global policies.

Summary. Section 3.1 introduced an extended data usage control model that allows for the explicit distinction of multiple systems, their individual behaviors, as well as their interplay. By further allowing to combine the system traces and system states of multiple concurrently executing systems, the model provides a tool to analyze and

compare the behavior of single systems, as well as the behavior of an interrelated distributed system.

On the basis of this model, Section 3.2 addressed **RQ1** by describing a generic mechanism for cross-system data flow tracking. This mechanism allows the usage control infrastructure to be aware of all representations of all usage controlled data items throughout the entire distributed system. Only the availability of such knowledge enables the enforcement of global data usage policies on *all* representations of the protected data. By instantiating these mechanisms for TCP/IP, it becomes possible to perform cross-system data flow tracking for a multitude of applications (e.g. web browsing, file transfer, email) and application protocols (e.g. HTTP, FTP, SMTP) in a transparent manner, i.e. without the need for any modifications to applications and/or the operating system.

Based on the above model and mechanisms, Section 3.3 addressed **RQ2** by providing methods to coordinate decisions about *global policies* across multiple distributed PDPs. To this end, the first step is to identify which systems are potentially relevant for evaluating a given policy at a given point in time, thus limiting the amount of systems between which coordination of policy decisions is required. Knowing this set of potentially relevant systems, it becomes possible to identify situations in which still no coordination between those systems is required without compromising policy enforcement. In the best case it is thus possible for local PDPs to conclusively evaluate global policies, thus reducing communication and performance overheads (cf. Section 5.3).

4

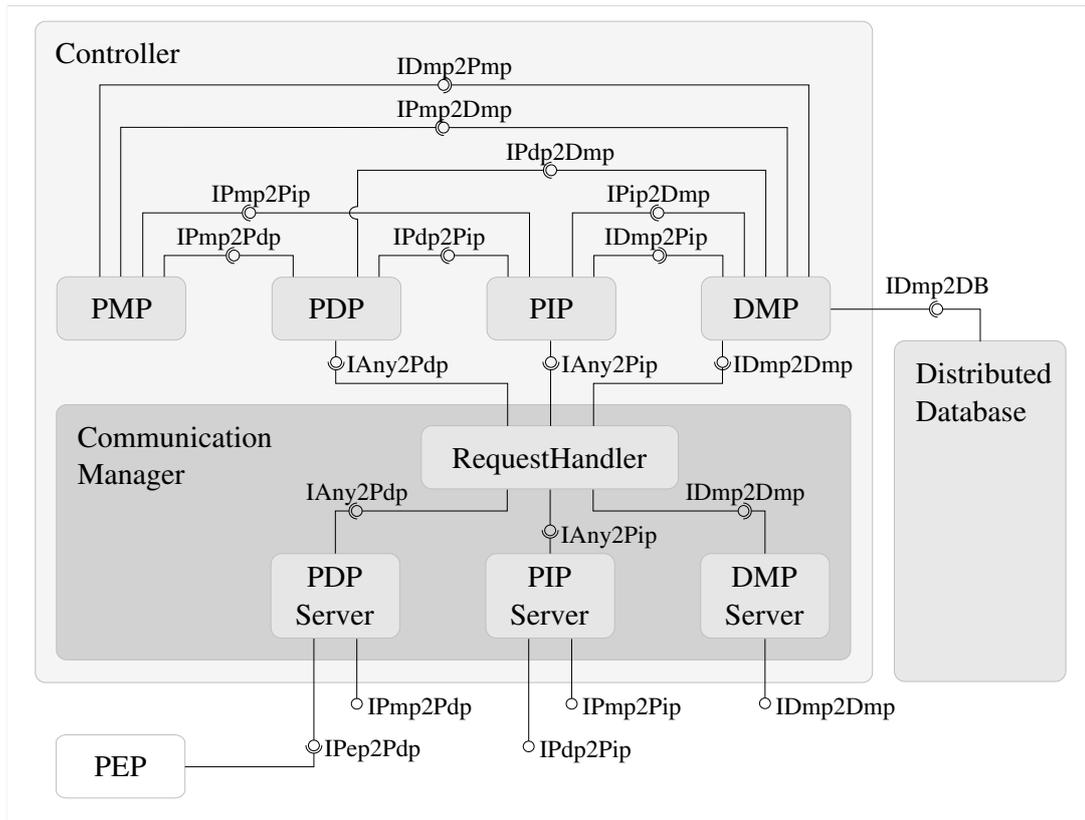
Architecture and Implementation

In order to show the feasibility of the concepts described and developed in Chapters 2 and 3, this chapter presents a technical architecture and its implementation, fulfilling the following functional system requirements:

- R1:** The infrastructure must allow for the deployment of data usage control policies in the form of ECA rules (cf. Sections 2.1.3 and 2.2.5).
- R2:** The infrastructure must be able to receive system events from PEPs, take policy decisions in correspondence with the deployed data usage control policies, and signal the decision back to the PEP (cf. Sections 2.2.2 and 2.2.3).
- R3:** The system's internal data flow state must be recorded and kept in a state that is consistent with the events happening (cf. Sections 2.1.2 and 2.2.4).
- R4:** The system must be able to track data flows across systems and to propagate the corresponding data usage control policies to the corresponding decision points. This requirement is in line with research question **RQ1** posed in Section 1.1.1 and the conceptual work presented in Section 3.2.
- R5:** The system must be able to consistently enforce data usage control policies if the events or the data addressed therein are distributed across systems. This requirement is in line with research question **RQ2** posed in Section 1.1.2 and the conceptual work presented in Section 3.3.

Note that requirements **R1**, **R2** and **R3** are not at the core of this thesis and hence the corresponding conceptual ideas are also *not* a contribution of this thesis. Even though components implementing these requirements have already been described at a high-level in Section 2.2, this chapter also details these local aspects of the architecture for two reasons: First, the distributed aspects and requirements (i.e. requirements **R4** and **R5**) build upon these local components. Second, large parts of these local aspects

Figure 4.1: High-level component diagram and its most important interfaces.



of the enforcement infrastructure have been (re-)designed and (re-)implemented within this thesis.

In the following, Section 4.1 first provides a high-level view of the infrastructure, also catering to requirements **R1**, **R2** and **R3**. Then, Section 4.2 describes how cross-system data flow tracking and policy propagation is achieved in practice (requirement **R4**), while Section 4.3 describes how global policies are coordinated across distributed PDPs (requirement **R5**).

4.1 High-level Architecture

By definition (Section 3.1), a system consists of exactly one PDP/PIP and multiple PEPs which use this PDP/PIP for taking usage control decisions and tracking local data flows. On the grounds of this definition, Figure 4.1 depicts the infrastructure's main components and interfaces, essentially depicting one system as defined in Section 3.1. Note that the architecture described in the following is simplified. Components and interfaces that are not relevant to the main contribution of this thesis are not shown and discussed. The subsequent sections describe the architecture's most important components as well as their interplay.

4.1.1 Component Overview

At a very coarse level, the architecture is composed of three independent but interconnected components: PEPs, a Controller and a distributed database. As further detailed in Section 4.1.4, PEPs are integrated into technical system layers (cp. Section 2.2.2) and it is their task to signal system events to the PDP and to enforce the PDP's decisions. The Controller features the PDP, PIP, and PMP, which have already been described in Section 2.2. Thereby, the Controller coordinates both the internal and external communication of those components. Further, the Controller features a Distribution Management Point (DMP) which is responsible for managing all tasks related to the distributed aspects discussed in this thesis such as cross-system data flow tracking and policy propagation (requirement **R4**, Section 4.2) as well as coordinating distributed policy decisions (requirement **R5**, Section 4.3). For this latter task the DMP interfaces with a distributed database which will be further detailed in Section 4.3. Further crucial components are the CommunicationManager and the RequestHandler.

CommunicationManager. The CommunicationManager manages all external communication of the Controller, for which it runs three RPC (remote procedure call) servers. It is the task of those servers, namely PdpServer, PipServer and DmpServer, to exhibit the functionality of the corresponding components to the outside. Most importantly, the PdpServer provides the interface needed for PEPs to request data usage control decisions in the presence of intercepted system events. By providing further interfaces to the outside, it is possible to deploy PDP, PIP, and PMP remotely from one another, thus catering to complex deployment requirements, e.g. if several PDPs ought to share the same PIP or PMP. This is further detailed in Section 4.1.3. The DmpServer provides interfaces that allow for the communication with remote DMPs, thus enabling cross-system data flow tracking and policy propagation (requirement **R4**). Note that all communication from/to the Controller, in particular when communicating with other remote Controllers as well as the distributed database, is secured using the Transport Layer Security (TLS) protocol [46]. In addition, the integrity and mutual authentication of remote Controllers is assumed at this point; these assumptions are discussed in Section 5.1.1.

RequestHandler. The task of the RequestHandler is to preprocess the requests that have arrived at the RPC servers and to queue them. Requests are queued for three reasons: (1) If many requests arrive at the CommunicationManager within short time intervals, the requests are buffered for later processing instead of being rejected due to busy components, i.e. PDP, PIP, PMP, and DMP; (2) It is ensured that requests are processed in the same order in which they arrive at the CommunicationManager; (3) It is ensured that only one request to PDP, PIP, PMP, or DMP is processed at each point in time, thus avoiding race conditions and other potential conflicts. Once a request from the RequestHandler's queue has been processed, the RequestHandler immediately triggers processing of the subsequent request.

Separation of the architecture into the above components is intentional as it organizes different functionalities into different components. Thereby the fundamental policy management and decision logics are provided by the PDP, PIP and PMP which follow the standard security architecture described in [49, 104, 160]. As detailed in Section 2.2.1, the PMP is in charge of policy management tasks (requirement **R1**), while the PDP takes policy decisions (requirement **R2**) and the PIP maintains and provides additional information about the data flow state (requirement **R3**) that is required for the PDP's decision process. In order to keep changes to existing local enforcement infrastructures as small as possible, tasks related to the distributed aspects of data usage control are outsourced to the new DMP component. Thereby, cross-system data flow tracking and usage control policy propagation (requirement **R4**) is organized in a peer-to-peer manner between each pair of remote DMPs whenever appropriate (cf. Section 4.2). For the coordination of distributed policy decisions (requirement **R5**), the DMPs leverage a distributed database (cf. Section 4.3). The purpose of the `CommunicationManager` is to separate the infrastructure's external communication from the `Controller`'s internal decision logics. This separation allows to easily change the communication technology without affecting any internal components.

Technically, the above infrastructure has been implemented in Java, which allows to deploy the infrastructure on a multitude of different operating systems. The infrastructure's internal components, i.e. PDP, PIP, PMP and DMP, are able to communicate both via function calls as well as via RPC. The latter has been implemented on the basis of Apache Thrift [192, 198]—a cross-language RPC framework originally developed at Facebook and now maintained by the Apache Software Foundation. The benefit of Thrift is that it abstracts from any underlying technologies, such as the operating system and the programming language being used. It allows for the convenient generation of client and server stubs for many different languages (including Java, C++, Python, OCaml, Delphi and others) from specifications written in the Thrift interface definition language. Consequently, Thrift allows communicating components to be written in different languages and to be run on different platforms. This is particularly useful because PEPs are integrated into many different layers of the system and thus written in a diverse set of languages. In addition, Thrift provides TLS support including the possibility to perform client-side authentication.

By enabling the internal components to communicate both via function calls and Thrift, the infrastructure can be flexibly deployed: If all components are deployed locally, fast function calls can be leveraged for inter-component communication. However, if necessary or beneficial in a given scenario, components can also be deployed remotely and communicate via Thrift. For RPC communication, corresponding Thrift services run on well-known ports. Possible deployment scenarios are further detailed in Section 4.1.3.

Listing 4.1: Interfaces provided by the PDP.

```

interface IPmp2Pdp {
    // Deploys policy p at the PDP and starts to enforce it.
    void deployPolicy(Policy p);

    // Revokes policy p from the PDP and stops enforcing it.
    void revokePolicy(Policy p);

    // Returns all policies which are currently deployed at the PDP.
    Set<Policy> listPolicies();
}

interface IPep2Pdp {
    // Signals system event e to the PDP. The PDP will evaluate the
    // event against all deployed policies and return a decision.
    Decision signal(SysEvent e);
}

interface IAny2Pdp extends IPmp2Pdp, IPep2Pdp;

```

4.1.2 The Interfaces provided by PDP, PIP, PMP, and DMP

As can be seen in Figure 4.1, several interfaces are used to coordinate the communication between the infrastructure's components. In the following, an overview over those interfaces and their most important methods are given. How these interfaces are used is described throughout the remainder of Chapter 4.

Interfaces provided by the PDP. The PDP provides two original interfaces, `IPmp2Pdp` and `IPep2Pdp`, detailed in Listing 4.1, which provide essential services to the PMP and PEPs, respectively. The most important methods provided by those interfaces are deployment, retrieval, and revocation of policies (requirement **R1**), as well as the possibility to signal system events and await a corresponding policy decision (requirement **R2**). Besides, `IAny2Pdp` is a convenience interface which unifies the methods provided by `IPmp2Pdp` and `IPep2Pdp`.

Interfaces provided by the PIP. The PIP provides two original interfaces, `IPdp2Pip` and `IPmp2Pip`, which are detailed in Listing 4.2. Using interface `IPdp2Pip`, the PDP is able to leverage the PIP for policy evaluation purposes (requirement **R2**). For this, `IPdp2Pip` provides methods to evaluate state-based operators and to retrieve all data within a given container. Further, the interface allows the PDP to signal data flow system events to the PIP, upon which it will update its data flow state (requirement **R3**). Interface `IPmp2Pip` allows the PMP to inform the PIP about the initial representations of some data as further detailed in Section 4.2.2. This is important when the PMP is about to deploy a new policy (requirement **R1**).

The additional interface `IDmp2Pip` is semantically equivalent to `IPmp2Pip` and is supposed to be used by the DMP in case cross-system data flows occur (requirement

Listing 4.2: Interfaces provided by the PIP.

```

interface IPdp2Pip {
    // Evaluates the specified state-based operator s
    // and returns the resulting truth value.
    boolean evaluate(StateBasedOperator s);

    // Retrieves all data items within the
    // container which is identified by i.
    Set<Data> getDataInContainer(Identifier i);

    // Signals the specified event e, upon which the PIP will update the
    // data flow state in correspondence with e's event semantics.
    void signal(SysEvent e);
}

interface IPmp2Pip {
    // Informs the PIP about a new representation for the
    // specified data item d. The initial representation
    // is the container which is identified by i.
    void initialRepresentation(Identifier i, Data d);
}

interface IDmp2Pip extends IPmp2Pip;

interface IAny2Pip extends IPdp2Pip, IPmp2Pip;

```

Listing 4.3: Interfaces provided by the PMP.

```

interface IDmp2Pmp extends IPmp2Pdp;

```

R4). Note that convenience interface `IAny2Pip` only unifies interfaces `IPdp2Pip` and `IPmp2Pip`, since the local PIP is not supposed to be queried by remote DMPs.

Interfaces provided by the PMP. The PMP provides only one interface, `IDmp2Pmp` (cf. Listing 4.3), which is used by the local DMP. The provided methods allow to deploy and revoke policies (requirement **R1**), as well as to retrieve the set of currently deployed policies. Interface `IDmp2Pmp` is equivalent to interface `IPmp2Pdp` which is provided by the PDP. The reason for this redundancy is that the PMP performs additional administrative tasks before forwarding the corresponding requests to the PDP.

Interfaces provided by the DMP. The DMP provides four interfaces which are detailed in Listing 4.4: (1) Interface `IPip2Dmp` allows the PIP to inform the DMP about data flows to containers residing on remote systems (requirement **R4**). (2) Interface `IPdp2Dmp` provides functionalities for the PDP that are necessary for coordinating policy decisions with other PDPs (requirement **R5**). This includes a method to notify to the DMP that the state of some local operator has changed (cf. Section 4.3), as well as several methods to query whether the state of some operator has changed at remote PDPs within or since a given timestep. Further, the interface provides methods to synchronize the points in time in which policy evaluation needs to take place. All

Listing 4.4: Interfaces provided by the DMP.

```

interface IPip2Dmp {
    // Instructs the DMP to perform remote data
    // flow tracking of the set of data d to the
    // remote container which is identified by i.
    void doRemoteTransfer(Identifier i, Set<Data> d);
}

interface IPdp2Dmp {
    // Notifies the DMP that the state of operator o has
    // changed within the PDP. The DMP will make this state
    // change available to other interested remote PDPs.
    void notify(Operator o);

    // Returns true if the specified operator o was true
    // at remote locations at the specified timestep.
    boolean wasTrueAt(Operator o, long timestep);

    // Returns how often the specified operator was true
    // at remote locations at the specified timestep.
    int howOftenTrueAt(Operator o, long timestep);

    // Returns how often the specified operator was true
    // at remote locations since the specified timestep.
    int howOftenTrueSince(Operator o, long timestep);

    // Notifies to the DMP the very first point in time in
    // which policy p was evaluated by the very first PDP.
    void setFirstEvaluation(Policy p, long timestamp);

    // Retrieves from the DMP the very first point in time in
    // which policy p was evaluated by the very first PDP.
    long getFirstEvaluation(Policy p);
}

interface IPmp2Dmp {
    // Registers the specified policy p at the DMP for
    // further distributed administration of the policy.
    void register(Policy p);

    // Unregisters the specified policy p.
    void unregister(Policy p);
}

interface IDmp2Dmp {
    // Informs a remote DMP that the set of data items d is about
    // to be transferred to a container under its administration.
    // The corresponding container is identified by i. Policies p
    // must from now on be enforced by the remote infrastructure.
    // Address a is a contact point at the invoking system.
    void remoteTransfer(Identifier i, Set<Data> d,
        Set<Policy> p, Address a);
}

```

information necessary for coordinated decision taking is exchanged between different remote DMPs via a distributed database. This is further detailed in Section 4.3. (3) Interface `IPmp2Dmp` allows the PMP to register and unregister policies at/from the DMP (requirement **R1**). (4) Interface `IDmp2Dmp` provides functionalities for cross-system data flow tracking and policy propagation between remote DMPs (requirement **R4**). This is further detailed in Section 4.2. Note that out of those four interfaces only interface `IDmp2Dmp` is exhibited to the outside. The other interfaces may only be used by the corresponding local components.

For all of the above interfaces it is assumed that they can only be utilized after a mutual authentication between the two communicating components has been performed, e.g. on the basis of a certificate infrastructure. In addition, the integrity of any remote Controller ought to be ensured before communication. These assumptions are discussed in Section 5.1.1. Further, deployment of new policies as well as policy revocation must only be performed by authorized parties that have the explicit permission to do so for the data addressed by the corresponding policy. Usually the legal owner of some data is such an authorized party. While this problem is out of the scope of this work, corresponding related work is discussed in Section 7.2.

Finally, it remains to clarify how the above interfaces and the datatypes being used therein relate to the models described in Chapters 2 and 3: Data type `Policy` represents ECA rules (cf. Section 2.1.3), data type `SysEvent` represents system events \mathcal{S} , data type `StateBasedOperator` represents state-based operators Ω , data type `Data` represents the data items to be protected \mathcal{D} , data type `Identifier` represents identifiers \mathcal{I} , data type `Address` represents addresses \mathcal{A} , and, finally, data type `Operator` represents the set of all policy operators such as *true*, *false*, *and*, *not*, *since*, *before*, *repmIn*, *isCombined*, and *isNotIn*.

4.1.3 Deployment Strategies

While the Controller provides all functionalities for both intra-system and cross-system usage control enforcement, its components and interfaces have been designed with many different requirements and scenarios in mind. As such, the entire infrastructure can be flexibly deployed. This section gives an overview of possible deployment strategies.

First of all, the above architecture allows to switch off single components of a Controller and to query corresponding remote components instead. This is the reason for exhibiting interfaces such as `IPmp2Pdp` and `IPdp2Pip` to the outside. For example, consider a scenario in which the PDPs of two Controllers ought to share the same PIP. In such a case, it is possible to switch off the PIP and `PipServer` within one of the Controllers and to have the corresponding PDP query the other remote PIP via interface `IPdp2Pip` instead. Similarly, it would be possible to deploy three Controllers, one of which only runs a PIP and the corresponding `PipServer`,

which would then be queried by the PDPs of the two other Controllers. While such deployment scenarios might be beneficial in certain application scenarios, they might also be used for load balancing.

The most straightforward deployment strategy is to deploy one single fully functional Controller for *all* global distributed PEPs. This way, it is possible to mimic the behavior of a purely centralized enforcement infrastructure, operating one central PDP/PIP instance. In this case, all PEPs must be configured to signal all system events to this central component.

However, the rationale behind developing the above infrastructure was to deploy PDPs and PIPs locally in order to improve on communication and performance overheads. Hence, another possibility is to deploy one fully functional Controller per organizational unit, department, or physical or virtual machine. In this case, the PEPs will always query the local PDP for decision making purposes, which, in turn, will query the local PIP for the local data flow state. Similarly, the local PMP manages all policies being enforced by the local PDP. As such, all components of each Controller keep a local state in correspondence with the events that have been signaled by the local PEPs. It is the responsibility of the DMP to perform tasks related to distributed usage control enforcement as further detailed in Sections 4.2 and 4.3.

4.1.4 Policy Enforcement Points

While the architecture described above is generic, PEPs must be integrated into different layers of the computing system for which they are expected to intercept relevant events and to signal them to the PDP/PIP for the purpose of decision making and data flow tracking. While PEPs have been built for many different system layers and applications (e.g. Android [55, 180], ChromiumOS [214], Java [61, 63], JavaScript [168], Mozilla Firefox [110], Mozilla Thunderbird [125], MS Office [188], MS Windows [216], MySQL [119], OpenBSD [74], OpenNebula [115], X11 [170]), this thesis leverages a PEP at the operating system layer for Unix-like systems. This PEP, a brief overview of which is given in the following, is a contribution of this thesis.

Based on the conceptual ideas proposed in [74], the developed PEP is based on the strace tool [102], the original purpose of which is to print out the “trace of system calls made by another process/program” [102]. Strace itself builds upon the ptrace system call [120], which allows to intercept, observe, modify, and prohibit system calls both before and after their execution by the kernel. Hence, no modifications to the operating system itself nor the monitored programs are needed. By building upon strace, most of the existing system call interpositioning framework can be reused and the actual PEP mainly consists of preprocessing the system calls for the PDP and PIP. For the most part, this preprocessing consists of collecting information required by the PDP and/or PIP, such as additional system call parameters not provided by strace. Eventually, the preprocessed system calls are signalled to the PDP via method `signal(SysEvent e)` (interface `IPep2Pdp`) and the corresponding policy decision is received. By building

upon `strace`, the PEP can be run on all systems supported by `strace`, which include Linux 32/64bit, BSD, and Android [151].

Since `strace` intercepts system calls both before and after their execution by the kernel, the PEP is capable of signaling corresponding intended and actual system events to the PDP. However, as detailed in Section 2.2.3, the PDP's and PIP's state only evolve in a persistent manner in the presence of *actual* system events. This behavior, however, may lead to race conditions: Consider a process that writes to a pipe or socket using system call `write`. Then, the PEP intercepts the processes' invocation of system call `write` (i.e. the invocation of the corresponding kernel space functionality) at time t_1 and the return of the same `write` system call after its execution by the kernel at time t_2 , $t_1 < t_2$. In correspondence with the above explanations, PDP and PIP would only evolve their internal states in a persistent manner at time t_2 . If, however, the PEP observes another processes' *actual read* system call on the same pipe/socket at time t_3 , $t_1 < t_3 < t_2$, then this might result in data flows that are not reflected within the PIP's data flow state. The current implementation mitigates such race conditions by conservative modeling: Whenever the PDP is about to allow an *intended write* system call (or any equivalent), both the PDP's and PIP's internal states are evolved instantaneously, i.e. without waiting for the PEP's signaling of the corresponding *actual* system call. Due to this instantaneous modeling of actual `write` system calls, the PEP is absolved from the need to signal actual `write` system calls (or any equivalents) to the PDP. Note that this behavior is of further importance in Section 5.2.

Since a high-level overview over the interactions between local PDPs, PIPs, and PMPs has been given in Section 2.2 and because these functionalities do not constitute a contribution of this thesis, they will not be further detailed in the following. Interested readers are pointed to [106, 126]. The following sections describe how cross-system data flow tracking and policy propagation (Section 4.2), as well as taking distributed policy decisions (Section 4.3) is achieved in practice. Hence, in the following it is assumed that one fully functional Controller is deployed per system. These Controllers are responsible for local data flow tracking and policy evaluation as well as intra-system data flow tracking. It is assumed that all Controllers are up and running and have not been tampered with. These and other assumptions are discussed in Section 5.1.

4.2 Cross-System Data Flow Tracking and Policy Propagation

A preliminary version of the content of this section has been published in [94].

Addressing RQ1, Section 3.2.1 introduced a generic model capable of capturing which data takes which representations within the entire distributed system. This model was instantiated for TCP/IP in Section 3.2.2, thus allowing to track data flows across systems independent of the applications and application-protocols being used. Building

upon these concepts, this section describes how TCP/IP-based cross-system data flow tracking is achieved in practice. In addition, the fact that usage controlled data is disseminated to different systems necessitates that the corresponding data usage policies are available to the PDPs which are in charge of enforcing those policies. Hence, also the latter aspect of shipping policies between systems is considered.

Note that by its very nature TCP/IP communication is limited to two communication partners. Thus, all considerations in the following sections are limited to a client and a server process. For scenarios with a larger number of systems and processes the presented features apply transitively.

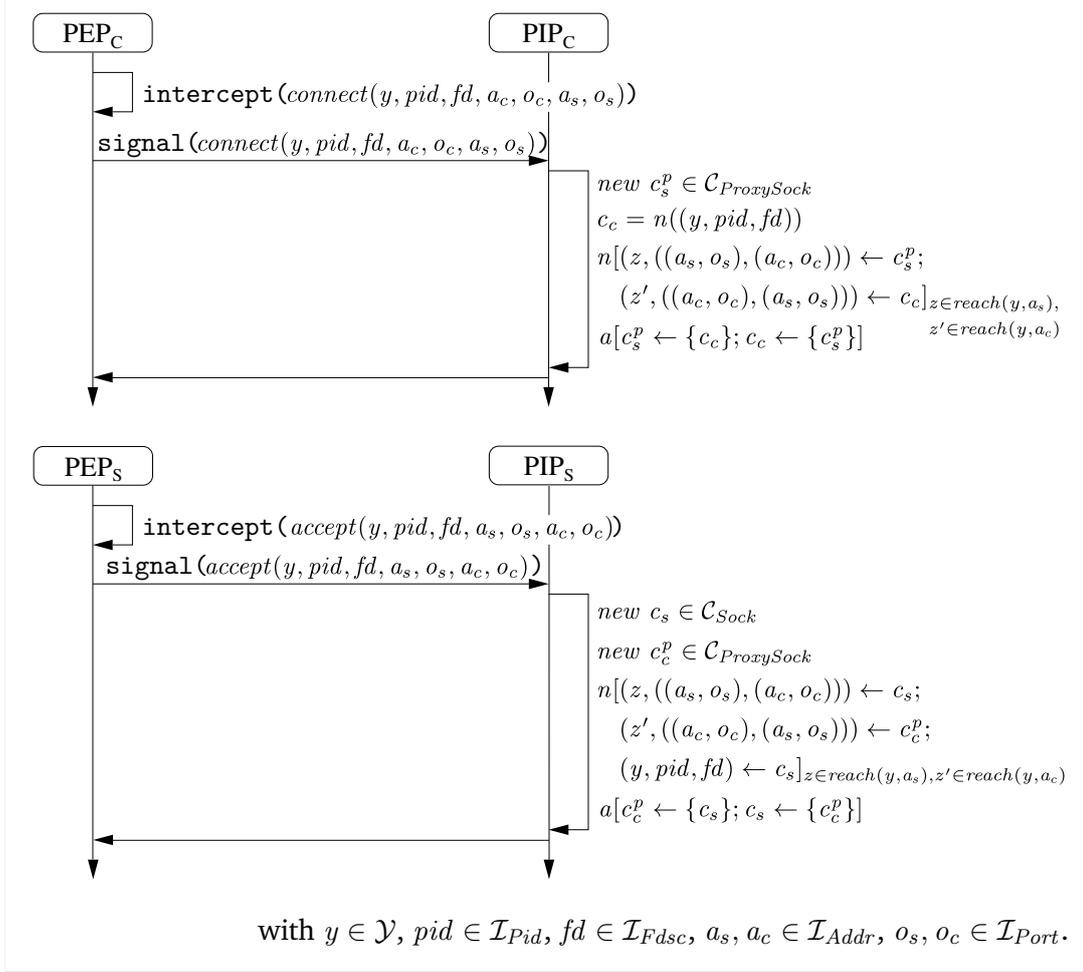
4.2.1 Connection Establishment

As described in Section 3.2.2, the data flow model bidirectionally aliases the server's and the client's communicating socket as soon as a TCP connection is established between two processes. However, it is nondeterministic whether the server's *accept* system call or the client's *connect* system call returns first. Further, there exists a cyclic dependency between the two system calls since all necessary information for modeling the connection establishment is only available once the second system call returns. For this reason, Section 3.2.2 assumed for each of those two system calls that it returns second. In the following, it is explained how this nondeterminism as well as the bidirectional alias is catered to by the implementation in case the TCP connection is established between two processes that run (i) on two remote systems, (ii) on the same system in the sense that the same PIP is responsible for tracking the processes' data flows.

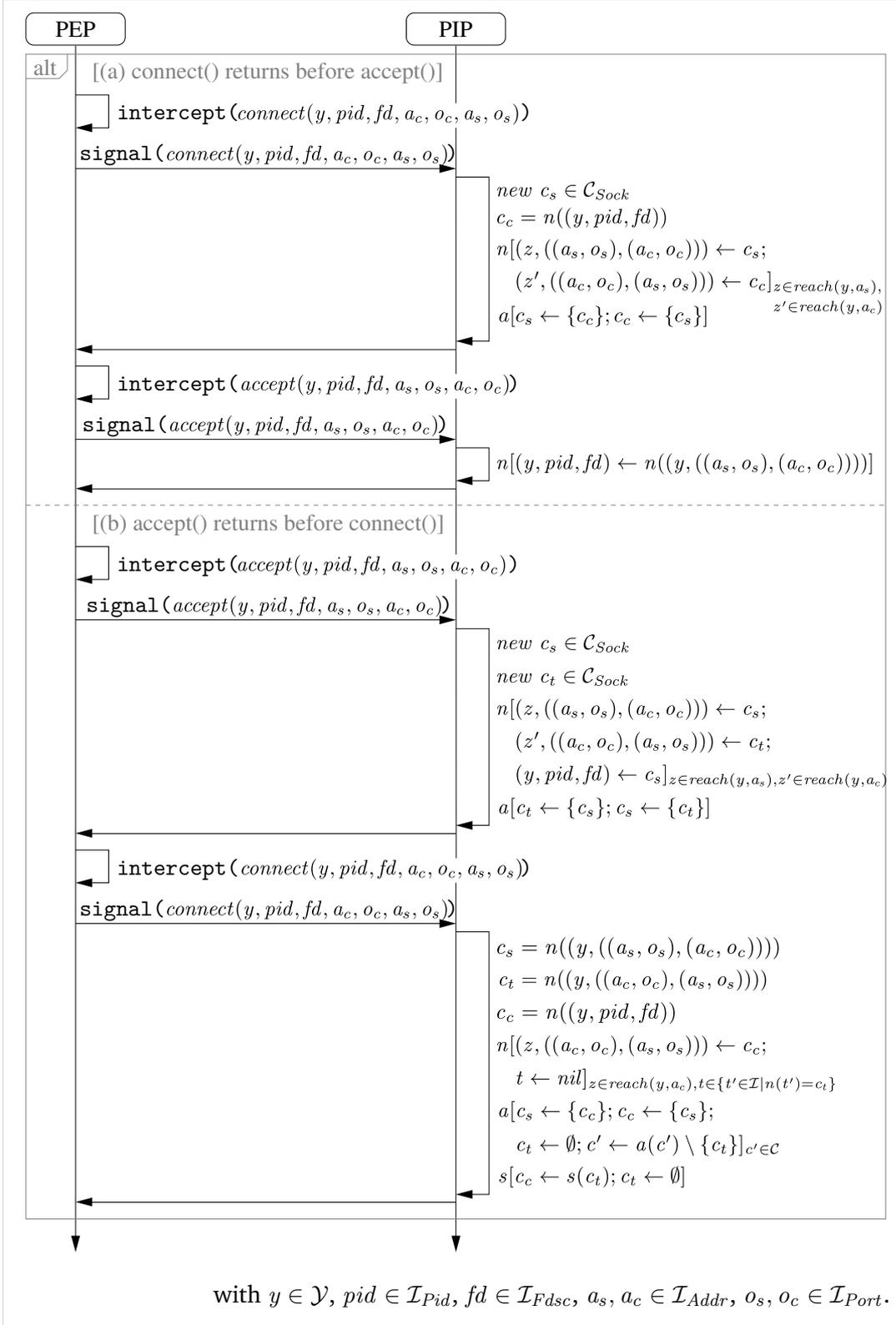
(i) If the two processes establishing a connection are remote, then the implementation creates two additional *proxy containers* ($\mathcal{C}_{ProxySock} \subseteq \mathcal{C}_{Sock}$)—one at the server side, $c_c^p \in \mathcal{C}_{ProxySock}$, and one at the client side, $c_s^p \in \mathcal{C}_{ProxySock}$. These proxy containers constitute local replacements for the corresponding actual remote socket containers and they are identified using the same identifier. The purpose of using these proxy containers is to avoid remote communication by the infrastructure upon connection establishment. At the server side c_c^p represents the remote container c_c , while at the client side c_s^p represents the remote container c_s . Consequently, the server's PIP, PIP_S , bidirectionally aliases c_s with c_c^p , while the client's PIP, PIP_C , bidirectionally aliases c_c with c_s^p . These two independent procedures on the server and the client side are depicted in Figure 4.2.

(ii) If the two processes establishing a connection are local, i.e. if their data flows are tracked by the same PIP, then the implementation needs to differentiate whether *accept* or *connect* returns first. The case in which *connect* returns *before* *accept* was

¹Signaling of the event to the PIP is actually mediated through the PDP: The PEP invokes method `signal(SysEvent e)` on the PDP (interface `IPep2Pdp`), which, in turn, invokes method `signal(SysEvent e)` on the PIP (interface `IPdp2Pip`). This is simplified in Figures 4.2 to 4.4. Further, the parameter values of events *accept*, *connect*, and *write* correspond to the ones used in Section 3.2.2.

Figure 4.2: Modeling the establishment of a *remote* TCP channel.¹

easier to implement as explained in the following. A corresponding sequence diagram is depicted in Figure 4.3: (a) If `connect` returns *before* `accept`, then the PIP performs most necessary tasks upon observing the returning `connect`. These tasks consists of creating and naming the new socket container c_s , which represents the, yet inexistent, socket which will be created upon return of the corresponding `accept`, and bidirectionally aliasing c_s with c_c . Upon return of `accept`, it then only remains to assign a new identifier to c_s , namely the file descriptor returned by `accept`. (b) If `accept` returns *before* `connect`, creation of the bidirectional alias between c_s and c_c necessitates several steps. First, upon return of `accept` the server's connected socket container c_s and its corresponding identifiers are created. Further, an additional temporary socket container $c_t \in \mathcal{C}_{Sock}$ is created, which temporarily represents the client's socket c_c which attempted connection establishment. Then, c_s and c_t are bidirectionally aliased. The reason for aliasing c_s with c_t rather than the original client's socket c_c is that upon `accept` there is no performant way of determining the remote socket c_c which attempted connection establishment. Further, the server process might write to the established connection *before* the client's returning `connect` is observed, in which case the data written to the

Figure 4.3: Modeling the establishment of a *local* TCP channel.¹

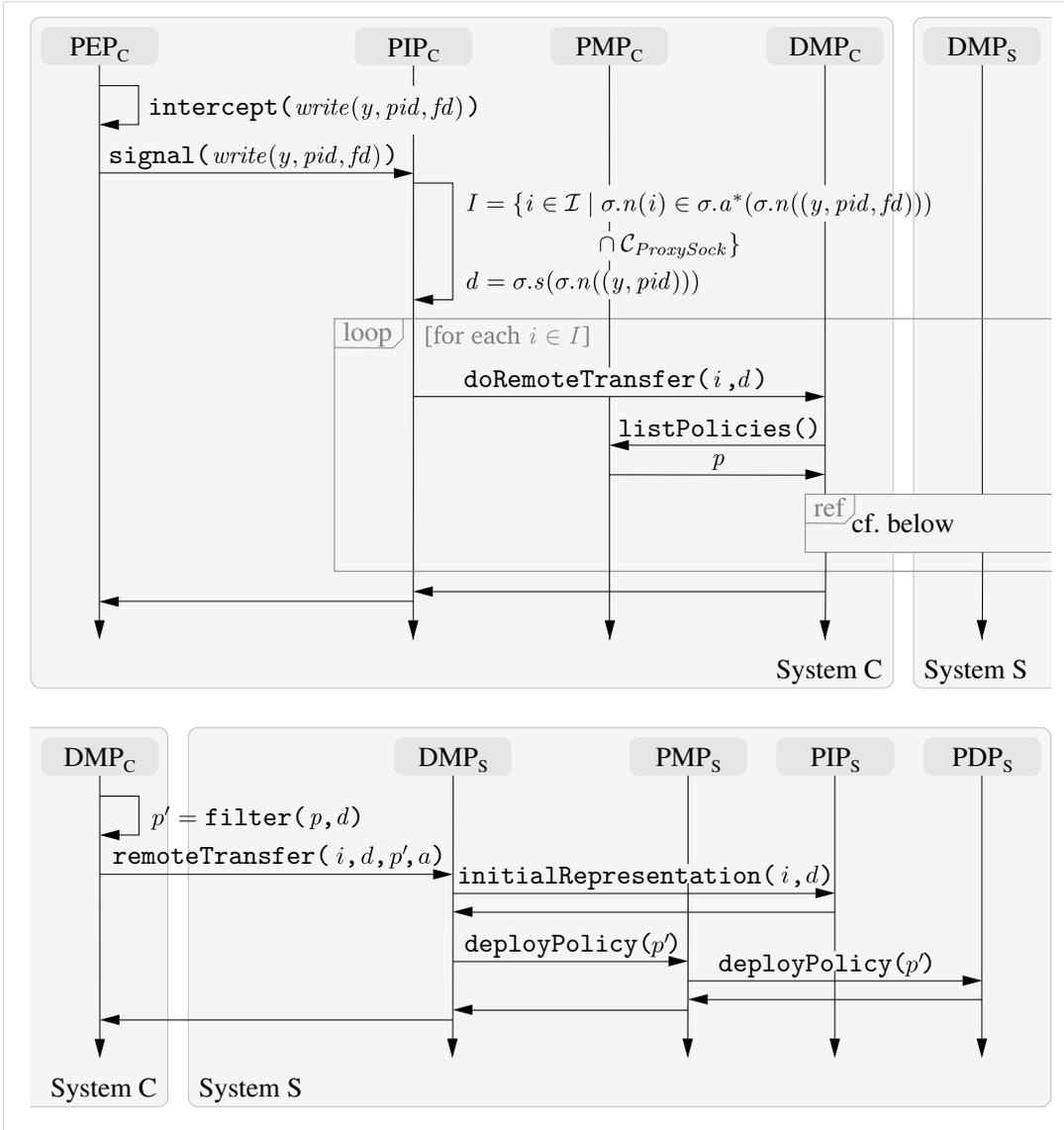
communication channel is not ‘lost’, but propagated to c_t . To be able to identify c_t later on, c_c ’s socket name (i.e. the local socket name and the remote socket name) is used as an identifier for c_t . Upon observing the returning *connect*, also the client’s connected socket c_c can be trivially identified via its file descriptor. Hence, the PIP bidirectionally aliases c_c with c_s and copies all data from c_t to c_c . In addition, c_c ’s socket name is now used to identify c_c rather than c_t . Finally, all storage, alias, and naming information related to c_t is deleted.

4.2.2 Data Transmission and Policy Propagation

After connection establishment, the client and the server process may exchange any kind of data by executing system call *write* (or any equivalent, cf. Section 3.2.2) on a file descriptor referring to the established TCP communication channel. In case the client and the server process are local, tracking of the corresponding data flows is trivially accomplished by the local PIP. If, however, the client and the server process reside on two different systems and are thus within the responsibility of two different PIPs, the usage control infrastructure must explicitly track the corresponding cross-system data flows. In addition, the infrastructure must ensure that also the corresponding policies are transferred together with usage controlled data and that they are deployed at the receiving side’s PDP. These functionalities are achieved by the DMP as explained in the following and depicted in Figure 4.4. Note that a prerequisite is that the PDP decided to allow the *write* system call on the TCP channel; otherwise no cross-system data flow (tracking) would happen in the first place. The remainder of this section describes how the implementation reflects a data transfer from a client process on system C to a server process on system S. Data transfer from the server process to the client process is analogous.

Once the client process’ *intended write*(y, pid, fd) system call to a TCP channel is intercepted and temporarily blocked by PEP_C , it is signaled to PDP_C and PIP_C for decision making and data flow tracking purposes. Naturally, PIP_C is capable of *detecting* the process’ *attempted* cross-system data flow: A cross-system data flow occurs if PIP_C propagates data to a container of type $C_{ProxySock}$, i.e. a proxy container representing a remote socket container. Formally, this is the case if for PIP_C ’s current state $\sigma \in \Sigma$ it holds that $\sigma.a^*(\sigma.n((y, pid, fd))) \cap C_{ProxySock} \neq \emptyset$. Upon detection of such an attempted remote data flow, PIP_C informs DMP_C about this fact, leveraging method `doRemoteTransfer(Identifier i, Set<Data> d)` (interface `IPip2Dmp`, Listing 4.4). Thereby $i \in (\mathcal{I}_{Sys} \times ((\mathcal{I}_{Addr} \times \mathcal{I}_{Port}) \times (\mathcal{I}_{Addr} \times \mathcal{I}_{Port}))) \subseteq \mathcal{I}$ refers to the destination container’s socket identifier and d to the set of data that is being transferred to this container.

DMP_C then retrieves all currently enforced policies from PMP_C using method `listPolicies()` (interface `IDmp2Pmp`, Listing 4.3), and filters the list of policies for those that constrain the usage of any data items within set d . Subsequently, DMP_C establishes a RPC connection to the remote DMP_S and performs remote data flow tracking

Figure 4.4: Cross-system data flow tracking and policy propagation.¹

and policy propagation via method `remoteTransfer(Identifier i, Set<Data> d, Set<Policy> p, Address a)` (interface `IDmp2Dmp`, Listing 4.4). Essentially, **DMP_C** informs **DMP_S** that the set of data items d is about to flow into the container identified by identifier i and that the set of policies p must consequently be enforced. The purpose of address parameter a is explained in Section 4.3.5.

Upon receiving the incoming RPC call, **DMP_S** informs **PIP_S** about the incoming data flow by invoking method `initialRepresentation(Identifier i, Set<Data> d)` (interface `IDmp2Pip`, Listing 4.2). **PIP_S** then updates its data flow state accordingly, essentially updating the storage function of the container identified by i with the provided data items d . Further, **DMP_S** informs **PMP_S** about the new policies to enforce by invoking method `deployPolicy(Policy p)` (interface `IDmp2Pmp`, Listing 4.3) for each transferred policy. In turn, **PMP_S** deploys those policies at **PDP_S**, provided they

have not been deployed before, e.g. due to some previous data flow between those systems.

Once this RPC call to DMP_S has succeeded, DMP_C returns and PIP_C continues with its execution. Eventually, the intended *write* system call which attempted the cross-system data flow is unblocked and the actual payload data flows to the receiving socket via the established communication channel. Once the receiving process reads from this socket, the corresponding data usage control infrastructure, in particular PDP_S and PIP_S , are already aware of the cross-system data flow and the corresponding policies, therefore extending usage control enforcement of the transferred data items to the receiving system.

Note that the data flow state of a process writing to a TCP channel might change in between two *write* system calls, e.g. if the process reads additional data. In this case the remote communication (i.e. `remoteTransfer()`) must be repeated upon the next *write* system call.

4.2.3 Connection Teardown

If a TCP connection between two processes on two remote systems is torn down, the corresponding local PIPs perform some cleanup by deleting the socket containers, the associated proxy containers, as well as their aliases and identifiers which were assigned during connection establishment.

Having described how data is tracked across systems and how policies are propagated accordingly (**RQ1**), the following section describes how global data usage policies, i.e. policies referring to data and events that are distributed across several systems, are enforced in practice.

4.3 Taking Distributed Policy Decisions

Contents of this section have been published in [93].

As stated by **RQ2**, global policies that have been disseminated to multiple systems are expected to be consistently enforced across all those systems at all times. To this end, Section 3.3 provided methods to coordinate decisions about global policies across multiple distributed PDPs. Building upon these results, this section focuses on the practical evaluation of conditions $\varphi_p \in \Phi$, which constitute the most complex and interesting part of ECA rules.

To explain how consistent enforcement of policies across systems is achieved in practice, the following considerations take the view of the PDP within a system A, PDP_A , which enforces ECA rule p with trigger event $e_p \in \mathcal{E}$, condition $\varphi_p \in \Phi$, and action a_p . As described in Section 2.2.3, any event signaled to PDP_A potentially changes the state of leaves within the expression tree of φ_p . Since such state changes are of potential importance for other PDPs enforcing p , PDP_A must make any such state

changes available to all corresponding other PDPs. As described in Section 3.3.2, this set of PDPs is overapproximated by function $relevant(\varphi_p, i, \tau)$ for each point in time i and the set of currently executing traces τ . The functionality to make state changes available to other PDPs is provided by the DMP. Hence, whenever state changes occur at PDP_A , it informs its DMP, i.e. DMP_A , about this fact via method `notify(Operator o)` (interface `IPdp2Dmp`, Listing 4.4). DMP_A is then responsible for exchanging this information with other DMPs, which in turn make this information available to their PDPs via interface `IPdp2Dmp`.

Section 4.3.1 describes at a high level how policy decisions are coordinated between PDPs, leveraging the interface `IPdp2Dmp` provided by the DMPs. Thereby, Section 4.3.1 intentionally abstracts from the technical synchronization between DMPs and assumes that all relevant remote DMPs are consistently, reliably, and timely informed about any state changes that are notified by the PDP. Sections 4.3.2 to 4.3.5 will then describe how remote DMPs synchronize in practice.

4.3.1 Coordinating Distributed Policies

As described in Section 2.2.3, ECA rule p must be evaluated whenever a timestep has passed or whenever a signaled event matches p 's trigger event e_p . In any of those cases each PDP enforcing p first evaluates φ_p locally, $eval(\varphi_p)$, according to Section 2.2.3.

Recap that $Sat(\tau, Y, i, \varphi_p)$, as defined in Section 3.3.2, states for each formula φ_p , given the tuple of executing traces τ , set of systems Y , and point in time i , whether φ_p 's local satisfaction implies its global satisfaction ($Sat(\tau, Y, i, \varphi_p) = true$) or whether its local violation implies its global violation ($Sat(\tau, Y, i, \underline{not}(\varphi_p)) = true$). Further recap that $eval(\varphi_p)$ reflects whether φ_p is satisfied given the currently executing trace t_Y^τ at the current point in time i (i.e. $(t_Y^\tau, i) \models \varphi_p$).

Hence, if local evaluation of φ_p yields $eval(\varphi_p) = true$ and if $Sat(\tau, Y, i, \varphi_p) = true$, i.e. if local satisfaction of φ_p implies its global satisfaction, then no further coordination with other PDPs is necessary: action a_p will be executed. Similarly, if $eval(\varphi_p) = false$ and if $Sat(\tau, Y, i, \varphi_p) = false$, then local violation of φ_p implies the global violation of φ_p and no coordination is required; action a_p will not be executed. In summary, no further coordination with other PDPs is required if $eval(\varphi_p) = Sat(\tau, Y, i, \varphi_p)$.

If, however, $eval(\varphi_p) \neq Sat(\tau, Y, i, \varphi_p)$, then it might be the case that the evaluation result of φ_p changes when considering other PDPs' state changes. Consequently, PDP_A re-evaluates φ_p , which is demanded to be given in DNF (cf. Section 3.3.2), with the help of DMP_A . For each leaf operator $o \in \{\mathcal{E}, \underline{isCombined}, \underline{repmIn}\}$ of φ_p , PDP_A queries DMP_A if o is not negated and if $eval(o) = false$, or if it is negated ($\underline{not}(o)$) and if $eval(\underline{not}(o)) = true$. For each leaf operator $o' \in \{\underline{isNotIn}, \underline{isMaxIn}\}$ of φ_p , PDP_A queries DMP_A if o' is not negated and if $eval(o') = true$, or if it is negated ($\underline{not}(o')$) and if $eval(\underline{not}(o')) = false$. This query strategy reflects the definition of predicate Sat in Section 3.3.2. Technically, PDP_A queries DMP_A using methods `wasTrueAt(Operator o, long timestep)` (for state-based operators $\underline{isCombined}$,

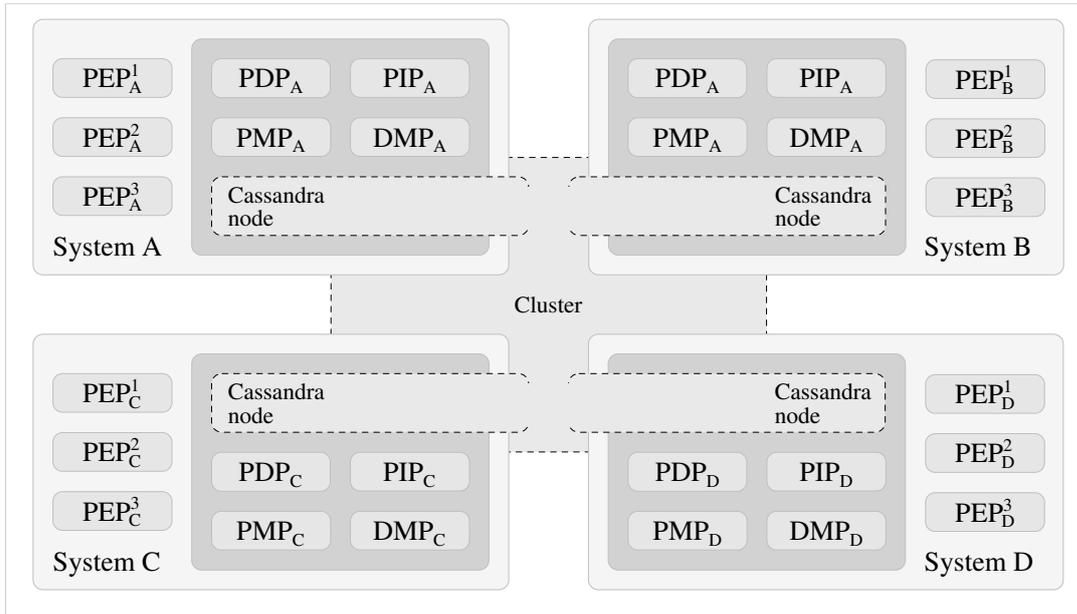
isNotIn, *isMaxIn*), *howOftenTrueAt*(Operator *o*, long *timestep*) (for events \mathcal{E}), and *howOftenTrueSince*(Operator *o*, long *timestep*) (for operator *repmIn*) of interface *IPdp2Dmp*, cf. Listing 4.4. If any of those queries yields a result different from the earlier local evaluation result, then the entire expression tree of φ_p is recursively re-evaluated starting from the root node. This re-evaluation then considers this newly available information.

As an example consider condition

$$\varphi_p = ((\text{requestOffer}, \{(obj, d)\}) \text{ before } 30) \text{ and } \text{repmIn}(30, 0, (\text{sendOffer}, \{(obj, d)\}))$$

of ECA rule 1a as described in Section 2.1.3, and a situation in which at system A event $(\text{sendOffer}, \{(obj, d)\})$ happened at the previous timestep, $i - 1$, while at system B the event $(\text{requestOffer}, \{(obj, d)\})$ did happen exactly 30 timesteps ago. Besides, no further events happen or have happened. PDP_A and PDP_B exchange the fact that these two events happened via DMP_A and DMP_B , using method *void notify*(Operator *o*) of interface *IPdp2Dmp*. For ease of writing φ_p is also written as $\varphi_p = \varphi_{p_1}$ *and* φ_{p_2} with $\varphi_{p_1} = ((\text{requestOffer}, \{(obj, d)\}) \text{ before } 30)$ and $\varphi_{p_2} = \text{repmIn}(30, 0, (\text{sendOffer}, \{(obj, d)\}))$. Note that according to the definition of *Sat*, it holds that $\text{Sat}(\tau, Y, i, \varphi_{p_1}) = \text{true}$ and $\text{Sat}(\tau, Y, i, \varphi_{p_2}) = \text{false}$ for $\tau = (t_A, t_B)$ and $Y = \{A, B\}$. Consequently, $\text{Sat}(\tau, Y, i, \varphi_p) = \text{Sat}(\tau, Y, i, \varphi_{p_1}) \wedge \text{Sat}(\tau, Y, i, \varphi_{p_2}) = \text{false}$. First, PDP_A and PDP_B evaluate φ_p locally: PDP_A 's local evaluation yields false, $\text{eval}_A(\varphi_p) = \text{false}$, since from PDP_A 's local point of view the event $(\text{requestOffer}, \{(obj, d)\})$ has *not* happened 30 timesteps ago. However, local evaluation of φ_p by PDP_B yields true, $\text{eval}_B(\varphi_p) = \text{true}$, since from PDP_B 's local point of view the event $(\text{requestOffer}, \{(obj, d)\})$ *did* happen 30 timesteps ago, while the event $(\text{sendOffer}, \{(obj, d)\})$ did *not* happen within the last 30 timesteps. At this point PDP_A is aware that the locally observed violation of φ_p implies the global violation of φ_p since $\text{eval}_A(\varphi_p) = \text{Sat}(\tau, Y, i, \varphi_p) = \text{false}$. Consequently, PDP_A can conclude $\text{eval}_A^g(\varphi_p) = \text{false}$. At the same time, however, PDP_B can not conclude locally, since $\text{eval}_B(\varphi_p) = \text{true} \neq \text{false} = \text{Sat}(\tau, Y, i, \varphi_p)$. Looking more closely at PDP_B 's evaluation of φ_p , it turns out that $\text{eval}_B(\varphi_{p_1}) = \text{Sat}(\tau, Y, i, \varphi_{p_1}) = \text{true}$, hence $\text{eval}_B^g(\varphi_{p_1}) = \text{Sat}(\tau, Y, i, \varphi_{p_1}) = \text{true}$, and $\text{eval}_B(\varphi_{p_2}) = \text{true} \neq \text{false} = \text{Sat}(\tau, Y, i, \varphi_{p_2})$. Hence, PDP_B queries DMP_B for φ_{p_2} . Since PDP_A leveraged DMP_A to publish the fact that event $(\text{sendOffer}, \{(obj, d)\})$ did happen in the previous timestep, distributed evaluation of φ_{p_2} by PDP_B results in $\text{eval}_B^g(\varphi_{p_2}) = \text{false}$. Since $\text{eval}_B^g(\varphi_{p_1}) = \text{true}$, PDP_B can conclude $\text{eval}_B^g(\varphi_p) = \text{eval}_B^g(\varphi_{p_1}) \wedge \text{eval}_B^g(\varphi_{p_2}) = \text{false} = \text{eval}_A^g(\varphi_p)$.

It is important to note that time-based policy evaluations must consistently happen at the same time across all PDPs. Otherwise, the PDPs might come to different conclusions when evaluating the same policy. Consider once again the above condition φ_p , a point in time i , a timestep interval of two days, and a situation in which event $(\text{requestOffer}, \{(obj, d)\})$ happens at time i , while event $(\text{sendOffer}, \{(obj, d)\})$ never happens. Further assume that PDP_A evaluates φ_p at times $i, i + 2, i + 4, i + 6, \dots$,

Figure 4.5: Four systems connected via the Cassandra database.

while PDP_B evaluates φ_p at times $i + 1, i + 3, i + 5, \dots$. Then, PDP_B 's evaluation at times $i + 29$ and $i + 31$ yields *false*, while PDP_A 's evaluation at time $i + 30$ yields *true*. In order to avoid such inconsistent evaluation results, the decentral PDPs always evaluate at the same time. The corresponding evaluation times are coordinated via the DMPs using methods `setFirstEvaluation(Policy p, long timestamp)` and `getFirstEvaluation()` (interface `IPdp2Dmp`, Listing 4.4), as further explained in Section 4.3.3.

Synchronizing the times of policy evaluations across systems as explained above is subject to scheduling and clock synchronization issues. Even more, as [131] states, “it is very difficult to obtain a global, consistent view of all components in a distributed system”. The presented implementation (as further detailed in the following sections) made use of the Network Time Protocol (NTP) [142] to synchronize system clocks.

4.3.2 Using Cassandra as a Distributed Database

As indicated in Figure 4.5, the current implementation leverages a distributed database, namely Cassandra, to synchronize all information provided by the PDPs to their corresponding DMPs. Cassandra is a distributed database originally developed at Facebook [112] and now maintained and supported by The Apache Software Foundation [199] and DataStax, Inc. [41]. Its purpose is to provide a “highly available service with no single point of failure” being run “on top of [...] hundreds of nodes” [112]. As such, Cassandra has been designed to achieve high scalability, availability, and performance.

Data Replication. In Cassandra, the entire set of nodes forming the distributed database is called a *cluster*. Within such a cluster all nodes are equal, resulting in a distributed database without any master nodes. The cluster's data is organized via

cluster

keyspace *keyspaces*, and each *table* is associated with, or contained in, exactly one keyspace. *table* Keyspaces take a central role, since each keyspace's *replication strategy* defines among which nodes of the cluster its associated tables are replicated. Essentially a keyspace's *replication strategy* thus defines a set of nodes that participate in the cluster. For this reason data with the same replication requirements should be organized within the same keyspace. Within the context of enforcing global data usage policies, each PDP might need to enforce several policies at the same time and for each the set of remote PDPs with which coordination is required might differ. Hence, the implementation represents each policy by exactly one keyspace. E.g., consider policy p constraining the usage of data d which has representations in systems A and B. Then, in the implementation there exists keyspace k_p with replication strategy $k_p^{rep} = \{A, B\}$. Thus, if PDP_A notifies a state change of p to DMP_A, then DMP_A will make this information available within keyspace k_p . Cassandra will then take care of replicating this information to exactly those DMPs for the PDPs of which it is of interest, i.e. DMP_B.

Data Consistency. With the CAP theorem [21, 68] stating that consistency, availability, and partition-tolerance can not all be achieved at the same time, many eventually consistent databases have emerged in recent years [11, 212]. In this respect, Cassandra is flexible by allowing to trade consistency with performance. For the time being, strong data consistency is assumed, i.e. that reads by the DMPs on the database always return the most recently written value [42]. Section 4.3.4 describes how this is efficiently achieved in practice. In case strong consistency is not sufficient, Cassandra provides *lightweight transactions* which implement linearizable consistency, i.e. the possibility to perform operations in a sequence that must not be interrupted by others. While such transactions in a distributed (database) system reflect the well-known consensus problem [56, 210], Cassandra provides a corresponding solution on the basis of the Paxos consensus protocol [42, 113, 114].

Deployment. As indicated in Figure 4.5, the idea is to bundle each PDP with one DMP and consequently with one Cassandra instance. Once the Cassandra instance is executed, it operates as a node within the cluster. Effectively this results in each DMP forming a node within that cluster. While the implemented infrastructure also allows to run the DMP, and consequently the Cassandra node, on any system remote from the PDP, running them on the same host is beneficial in terms of runtime performance. For the time being it is assumed that all Cassandra nodes participate in the cluster as soon as they are started. How this is actually achieved in practice is explained in Section 4.3.5.

Clock Synchronization. While system clocks were synchronized using NTP, the implementation also leveraged Cassandra's capability of using server-side timestamps which are guaranteed to be unique among the entire Cassandra cluster [43]. As such, the performed coordinated policy evaluations did not reveal evaluation inconsistencies. If, however, the accuracy of NTP, which is up to one millisecond, would turn out to

be insufficient, then more accurate clock synchronization techniques could be used. Examples are the Precision Time Protocol (PTP) [72], which provides accuracy up to the microsecond-level, or the Global Positioning System (GPS) [117], which provides accuracy up to the nanosecond-level.

4.3.3 Bootstrapping and Reflecting Cross-System Data Flows

Consider a set of systems with their corresponding usage control infrastructure as depicted in Figures 4.1 and 4.5. Further assume that no data usage policy has yet been deployed. Then, at some point in time the first policy p , protecting data d , is deployed at system A via PMP_A as described in [106], e.g. if an end user deploys a policy via a dedicated policy editor tool. First, PMP_A parses the policy p and identifies the data protected by the policy, $d \in \mathcal{D}$, as well as d 's initial representations, say some container $c \in \mathcal{C}_A$ which is identified by identifier $i \in \mathcal{I}_A$ within system A. Using method `initialRepresentation(Identifier i, Data d)` (interface `IPmp2Pip`, Listing 4.2), PMP_A informs PIP_A about this initial representation of data d . Further, PMP_A registers policy p at the DMP_A using method `register(Policy p)` (interface `IPmp2Dmp`, Listing 4.4), which prepares keyspace k_p with replication strategy $k_p^{\text{rep}} = \{A\}$ for potential later coordination of policy p with other DMPs/PDPs. Finally, PMP_A deploys policy p at PDP_A , which then starts to enforce it (method `deployPolicy(Policy p)`, interface `IPmp2Pdp`, Listing 4.1). Once PDP_A performs the first time-based evaluation of p , the corresponding point in time is communicated to DMP_A using method `setFirstEvaluation(Policy p, long timestamp)` (interface `IPdp2Dmp`, Listing 4.4). DMP_A stores this information within keyspace k_p as it will be required for the future consistent enforcement of p across multiple PDPs. However, as of now p and d are only known to $\text{PDP}_A/\text{PIP}_A$, which is why PDP_A can independently take all decisions about p as described in Section 2.2.3.

Now, consider that system A shares data d with system B via the network. From then on, also system B might influence the evaluation of p . How the implementation handles such cross-system data flows has been described in Section 4.2.2. Recall that method `remoteTransfer(Identifier i, Set<Data> d, Set<Policy> p, Address a)` of interface `IDmp2Dmp` was used for that purpose. Once this method is called, DMP_B triggers the adaptation of the existing keyspace k_p , effectively adapting the keyspace's replication strategy to incorporate system B's Cassandra node, $k_p^{\text{rep}} \leftarrow k_p^{\text{rep}} \cup \{B\} = \{A, B\}$. Subsequently, all data written to k_p is immediately replicated to Cassandra nodes A and B and is thus immediately available to both DMP_A and DMP_B , and thus to PDP_A and PDP_B . As motivated earlier, once policy p has been deployed at both PDP_A and PDP_B , time-based policy evaluations at those two PDPs are expected to happen at the same times. Hence, upon deployment of policy p , PDP_B queries DMP_B for the very first point in time at which p was ever evaluated using method `long getFirstEvaluation(Policy p)` (interface `IPdp2Dmp`, Listing 4.4). This information is available to DMP_B within keyspace k_p . PDP_B then synchronizes its local time-based

evaluations with the returned value. Note that in the current implementation a keyspace's replication strategy reflects the set of systems being returned by function *knowD* as defined in Section 3.3.1. Thus, a keyspace's replication strategy already provides a good approximation of the systems being relevant for evaluating a given policy. In other words, for policy p keyspace k_p is a good approximation for function $relevant(\varphi_p, i, \tau)$ as defined in Section 3.3.1.

Now, system B might further share data d with system C. DMP_C will further adapt the existing keyspace to also incorporate system C's Cassandra node, $k_p^{rep} \leftarrow k_p^{rep} \cup \{C\} = \{A, B, C\}$. Notably, the keyspace's adaption is immediately perceived by nodes A and B, such that from now on all data written to k_p will be replicated to nodes A, B and C. In order to prevent conflicts and lost updates, these adaptations of a keyspace's replication strategy must be atomic. Hence, a corresponding locking mechanisms was implemented on top of the keyspace being updated. For atomic acquiring of the lock, Cassandra's lightweight transactions are used.

4.3.4 Cassandra Consistency

Up to now, strong data consistency was assumed. In Cassandra, each single read and write operation to a keyspace can be configured with a *consistency level* (CL). Consistency levels define how many nodes of the keyspace must acknowledge to have received and obeyed an operation. Among others, Cassandra provides the consistency levels *One*, *Two*, *Three* and *All*, respectively defining that one, two, three and all nodes of the keyspace must acknowledge the corresponding operation. While using $CL=All$ for all read and write operations guarantees strong data consistency, it comes at the cost of performance and the requirement that all of the keyspace's nodes must be always online and reachable by all other nodes. By providing consistency level *Quorum*, Cassandra allows to achieve strong consistency without such drawbacks: If $CL=Quorum$, then operations must be acknowledged by at least half of the nodes. Consequently, strong consistency can be achieved by using $CL=Quorum$ for all reads and writes. Note that strong consistency could also be achieved by using $CL=All$ for all writes and $CL=One$ for all reads, or vice versa. This, however, imposes the drawbacks for consistency level *All* mentioned above.

Whenever a consistency level different from *One* is used, reads and writes to a keyspace might fail. If $CL=All$, then it is sufficient that only one of the keyspace's nodes is not available in order to make queries to the keyspace fail. Since failing of a node or some network link is not unlikely in practice, a consistency level of *All* can be considered impractical. If $CL=Quorum$, read and write operations might fail if half of the nodes of a keyspace are not available. While such situations are not impossible, e.g. if network partitions occur, they are much more unlikely in practice. Considering the Cassandra cluster from the point of view of a single node, any query to a keyspace with $CL \neq One$ fails in case the considered node is offline. While configurable, by default the implementation uses $CL=Quorum$ for all reads and writes.

The implementation tackles the aforementioned problems by two means: First, it is configurable how often and in which intervals failed queries are retried. Second, if queries still fail after the predefined amount of tries, the PDP takes a fallback decision. Clearly, such a fallback decision depends on the policy being enforced, the scenario, and the attacker model. Hence, the policies can be configured accordingly.

4.3.5 Connecting Cassandra Nodes

In Cassandra nodes might join and leave the cluster at all times. To join, a new node needs some way to discover the cluster it ought to participate in. For this, it is sufficient for the new node to know *any* other node that is already part of the cluster. Once contacted, the new node learns about the cluster using a peer-to-peer gossip protocol [98]. For this startup phase Cassandra defines seed nodes, a set of fixed nodes which are highly available. Since one original goal of this thesis was to develop a fully decentral infrastructure, this section provides solutions to the problem of integrating new nodes into an existing cluster without any well-known seed nodes. Unfortunately, Cassandra does not provide an API to explicitly trigger the above gossip protocol on one specific node with the task to explore the cluster for yet unknown nodes. Having in mind that such a functionality would simplify the following solutions, the following workarounds are provided.

Recap the scenario described in Section 4.3.3, in which the very first policy p , protecting data d , is deployed at PDP_A , while PDP_B is not yet enforcing any policies. At some point in time, d , and subsequently policy p , is transferred to system B as described in Section 4.2. In Section 4.3.3 it was assumed that system B's Cassandra node did already participate in the cluster *before* the transfer of d from system A to system B. If this is not the case, however, then there must be a way of making the Cassandra node on system B join the cluster without any well-known seed nodes. The solution to this problem is to *not* start the Cassandra node together with the corresponding Controller, but only once the first global policy ought to be enforced: Once DMP_B receives policy p via remote procedure call `remoteTransfer(Identifier i, Set<Data> d, Set<Policy> p, Address a)` from DMP_A (cf. Section 4.2.2), DMP_B also gets to know the address a of system A's Cassandra node. Knowing this address, system B's Controller starts its Cassandra node, using the given address a as a seed node. Further, DMP_B adds address a to its private list of known Cassandra nodes, which is used to reconnect to the cluster in case a network or power outage occurred.

Now, consider an extended scenario in which systems A and B, i.e. PDP_A and PDP_B , enforce policy p , protecting data d , while PDP_C enforces policy p' which protects data d' . Since the sets of systems enforcing p and p' are disjoint, the overall cluster can be considered to be partitioned, while the single partitions are not aware of any other partitions. Once data d is transferred from system A to system C, these two partitions must be merged because in the following systems A, B and C must coordinate their decisions w.r.t. policy p . Since an explicit command to trigger the gossip protocol as

described above is missing, one way of technically resolving that problem is as follows: Once d is transferred from system A to system C, a temporary Cassandra node, which uses both A's Cassandra node as well as C's Cassandra node as seed nodes, is started. Gossiping through this temporary node, the previously autonomous parts of the cluster will get to know about each other. Once this has happened, the temporary node can be taken down again. While this functionality has not been implemented within the described usage control enforcement infrastructure, preliminary experiments showed that the described approach does work in practice.

Discussion. While most state-of-the-art distributed databases support the features that were required for synchronizing policy decisions (e.g. data replication and synchronization, fault and network partition tolerance, data consistency), the implementation leverages Cassandra for several reasons: (i) Keyspaces allow to flexibly define which database nodes replicate which data; (ii) the absence of any master nodes leads to a system without any single point of failure; (iii) consistency can be traded for performance and vice versa, allowing to 'tune' the performance in case (minor) inconsistencies in the policy decisions are acceptable, (iv) TLS including client authentication (i.e. authentication of the Controller) is readily available; (v) it is based on Java technology and can thus be run on any system on which the Controller can be run; (vi) convenient Java drivers are available. However, as the evaluation will show (Section 5.3), Cassandra introduces some non-negligible performance and communication overheads. It stands to reason that a solution tailored to the particular requirements of taking distributed data usage control decisions would improve upon these overheads.

5

Evaluation

This chapter provides a comprehensive evaluation of the concepts and the infrastructure developed within this thesis. The main purpose is to understand (i) which security guarantees are provided by the developed infrastructure (Section 5.1), and (ii) which communication and performance overheads are introduced (Sections 5.2 and 5.3). Finally, Section 5.4 analyzes the performed experiments w.r.t. certain threats to validity.

5.1 Security Evaluation

This section addresses **RQ3** posed in Section 1.1.3 by performing a security analysis of the infrastructure developed within this thesis. To this end, Section 5.1.1 starts with a discussion of security-relevant assumptions taken throughout this thesis. Section 5.1.2 then performs the actual security analysis. Finally, Section 5.1.3 summarizes by detailing in which situations, i.e. in the presence of which attacker models and within which kind of environment, which guarantees are provided when deploying the infrastructure developed within this thesis.

5.1.1 Security-relevant Assumptions

No Vulnerabilities. Since the usage control infrastructure developed in this thesis runs as a process within the operating system, both the usage control infrastructure and the operating system are assumed to be free of vulnerabilities. Otherwise, an attacker, both from the inside or the outside, might be able to gain administrative privileges and switch off or tamper with the usage control infrastructure. The same assumptions are expected to hold for state-of-the-art cryptographic methods and access control mechanisms, since the developed usage control infrastructure depends on such techniques as discussed in the following paragraphs.

Access Control. The usage control infrastructure leverages state-of-the-art access control mechanisms built into the operating system and applications. These mechanisms are expected to enforce standard access control policies such as constraining users in

accessing certain parts of the file system, e.g. other users' home directories or system files and directories. In particular, access control mechanisms must prevent regular users from gaining administrative privileges and from escalating their privileges in any other way.

Confidentiality and Integrity of Data. The usage control infrastructure assumes that Data at Rest and Data in Motion is handled confidentially and that its integrity is assured. In terms of Data in Motion, the implemented infrastructures builds upon TLS (Transport Layer Security) [46] to assure the confidentiality and integrity of all information exchanged between remote components of the infrastructure. Further, all information exchanged between the different nodes of the distributed database is secured using TLS. In terms of Data at Rest, the presented infrastructure does currently not implement any corresponding technologies. However, disk-encryption technologies such as EncFS [70], Microsoft BitLocker [139], and CipherShed [36], a fork of the discontinued TrueCrypt project [207], are fully transparent for application software and can thus be trivially integrated. Consequently, corresponding technologies are assumed to be in place. Note that the proper operation of such technologies requires access control mechanisms, as described above, to be in place.

Certificate Infrastructure. Due to the usage of TLS, the usage control infrastructures must be equipped with corresponding certificates, and, implicitly, public/private key pairs. For the evaluation in Sections 5.2 and 5.3, key pairs were decentrally created and the corresponding certificates were signed by a trusted Certificate Authority. Different instantiations of the infrastructure exchange and verify those certificates in a peer-to-peer manner whenever required. For this, the built-in TLS capabilities of Thrift and Cassandra were used. Using central Certificate Authorities might be discouraged when deploying an otherwise fully distributed infrastructure. In such a case, it is possible to organize the certificates in the manner of a Web of Trust, in which the concept of trust between parties is organized in a decentral rather than a central manner. Such considerations are orthogonal to the solution provided in this thesis. Along similar lines, it must be assumed that there do not exist ways for any entity to obtain invalid or faked certificates. If private keys are lost or stolen, the infrastructure should provide means to handle such situations, e.g. by maintaining and complying with corresponding revocation lists.

Correctness of Models and Implementation. The usage control infrastructure can only operate as expected if all of its aspects, such as data flow tracking and taking distributed policy decisions, have been modeled and implemented correctly. If in doubt about how to 'correctly' model a specific system behavior, the most conservative approach should be taken. Since the correctness of models, instantiations and implementations can not in all cases be formally proven, it is essential to review corresponding artifacts, e.g. instantiations of the generic data flow model described in Sections 2.3 and 3.2.2 (including state transitions \mathcal{R}) or the implementation described

in Chapter 4. In addition, it is advisable to digitally sign the infrastructure's code/executables before shipment and consequently to be able to verify its integrity upon deployment.

Robustness of Implementation. There are many reasons for which the usage control infrastructure or any of the underlying components, such as the operating system or the hardware, might fail. E.g., if the machine runs out of memory, if the system is under attack, or if power outages occur. Despite such situations usage controlled data must not be used without respecting the corresponding data usage policy. Hence, it is assumed that usage controlled data is exclusively stored on encrypted file systems that can only be decrypted and accessed by the usage control infrastructure. This way, users are not able to circumvent policy enforcement, even if components such as the operating system and/or its access control mechanisms fail. Along similar lines, the internal states of the infrastructure's central components, i.e. PDP, PIP, PMP, and DMP, must be kept in persistent memory. The reason is that this enables the recovery of those components' internal states, e.g. after a system crash. Without such persistency previous data flows, deployed policies, as well as policy decisions would be lost, potentially resulting in the uncontrolled usage of usage controlled data.

Integrity of the Usage Control Infrastructure. This thesis assumes the integrity of all deployed data usage control infrastructures, as well as the underlying operating systems and any other security-relevant mechanisms upon which the implementation builds. The extent to which this assumption is valid, however, depends on the considered scenario and attacker model. For example, such an assumption might be adequate in a business scenario such as the insurance company introduced in Section 1.5. In such scenarios, users are usually provided with hardware by the employer and do not have administrative privileges on the corresponding systems. If the usage control infrastructure is embedded within the operating system, run with administrative privileges, and if it is not possible for the user to tamper with that infrastructure, then the above assumption may be adequate. Similar scenarios are omnipresent when considering most of today's smartphones and tablet PCs, where users rarely have full administrative privileges. While it is technically possible to jailbreak those systems or to install alternative operating systems, such procedures are not an option for non-experts. If it comes to more open environments with many different heterogeneous distributed systems and end users having administrative privileges, however, the integrity of all deployed usage control infrastructures can surely no longer be assumed. While trusted computing technologies such as Trusted Platform Module (TPM) [208] and Next-Generation Secure Computing Base (NGSCB) [51, 140] provide technical solutions to such problems, there was no breakthrough of such technologies in over ten years of their development. Due to many disputes around such technology, a further discussion on this topic is provided in Chapter 7.

Omnipresence of the Usage Control Infrastructure. This thesis assumed a usage control infrastructure to be deployed on each system that maintains copies of usage controlled data. In this scenario a strict usage control implementation would by no means allow usage controlled data to be sent to systems that can not reliably state that a corresponding infrastructure is in place. This, however, might be unrealistic in practice. While solutions to this problem are likely to be highly dependent on concrete scenarios, one possible solution is to allow usage controlled data to be sent to non-usage controlled systems in a controlled manner, e.g. after critical parts of the protected data have been anonymized [60].

5.1.2 Security Analysis

A preliminary version of the content of this section has been published in [94].

While a security analysis can hardly be exhausting, the purpose of this section is to understand the attack surface and vulnerabilities present in the developed infrastructure. Consequently, this section implicitly describes potential future work by pointing to vulnerabilities and potential countermeasures. In the following, two main attacker models are considered: non-privileged end users as omnipresent in many business scenarios such as in the example introduced in Section 1.5, as well as a man-in-the-middle between different components of the distributed usage control infrastructure. Since the central goal of the data usage control infrastructure is to enforce compliance with data usage policies, the most interesting goal for attackers is to circumvent this enforcement, i.e. to be able to use data without respecting the corresponding policy. Other attacks considered in the following address the availability of usage controlled systems.

Creation of Unmonitored Data Copies. One possibility to achieve the goal of unmonitored data usage is to create copies of the protected data that are no longer monitored by the infrastructure. Such attacks are described in the following. If possible, corresponding countermeasures are described.

Since the cross-system data flow tracking instantiation (Section 3.2) and its implementation (Section 4.2) are limited to TCP/IP, it is possible to circumvent the tracking of cross-system data flows by using other protocols at the transport layer. User Datagram Protocol (UDP) is the second most prominent transport layer protocol. It is mainly used by time-critical applications such as voice, video and media streaming, and online games, as well as lightweight protocols that exchange only small amounts of data such as Network Time Protocol (NTP), Domain Name System (DNS), and Simple Network Management Protocol (SNMP). Consequently, any transfer of usage controlled data via any of those applications or protocols results in undetected and untracked cross-system data flows. The data could thus be used in an uncontrolled manner at the receiving system. The above arguments also hold for any other transport layer protocol different from TCP. There exist several possibilities to mitigate such attacks.

First of all, it is possible to disallow any cross-system data flows that do not build upon TCP/IP. This could either be achieved by having the system's firewall block all non-TCP traffic, or by having the usage control infrastructure inhibit the creation of any non-TCP sockets. In practice, however, such a solution would disallow essential network services such as DNS and NTP and is thus not acceptable in real-world production systems. Alternatively, the generic cross-system data flow model (Section 3.2) and its implementation (Section 4.2) could be extended to also incorporate non-TCP data flows, e.g. by monitoring cross-system data flows at the underlying IP layer. Note that equivalent attacks and solutions also apply to other data exchange technology such as Bluetooth or Wireless USB.

Neither the model nor the implementation described in this thesis catered to the possibility that the actual TCP communication channel over which the payload data is transferred between systems might be insecure. E.g., when using network protocols such as HTTP or FTP, data is usually transferred in plain text. This implies that any attacker that is able to observe the network and/or the input/output of the corresponding network interfaces can read the payload data. Consequently, a copy of the transferred data can be created and used without complying to the corresponding policy. One countermeasure is to only allow the transmission of usage controlled data via secure network channels, e.g. via HTTPS, FTPS (i.e. HTTP/FTP with TLS support), or SSH. Alternatively, insecure protocols without support for TLS could be tunneled over protocols ensuring confidentiality and integrity of data, such as Virtual Private Networks, SSH, IPsec [99], or `tcpcrypt` [20].

End users might save usage controlled data to portable media (e.g. USB drives, CD-ROM), physically removable hard disks, or the like. By unmounting these data media and using them at non-usage controlled systems, unmonitored copies of the usage controlled data can be created. Solutions to this attack are to disallow all attempts to save usage controlled data to portable media or to only allow usage controlled data to be saved to encrypted media that can only be decrypted by the usage control infrastructure. Alternatively, the usage control infrastructure could take the effort to transparently encrypt all usage controlled data that is stored to persistent memory, e.g. by mediating all corresponding events and their payload data through encryption tools such as GNU Privacy Guard (GnuPG) [201]. For either solution, the question remains how attackers can be kept from reengineering the encryption/decryption algorithms and, most importantly, how the corresponding keys can be kept secret.

Another possibility for attackers is to create unmonitored copies of the protected data at system layers not monitored by the usage control infrastructure. While generally the possibility of such attacks can be considered a limitation of the usage control infrastructure and its implementation, there exist system layers at which the monitoring of data usage and data copies is hard if not impossible. E.g., although encrypted in persistent memory and in-flight, usage controlled data must ultimately be decrypted in order to perform operations on it. As such, unencrypted copies of the data or the

corresponding decryption key remain in registers at the CPU level and within the main memory, giving rise to different attacks, e.g. [71, 184]. While possible countermeasures have been described in the literature, e.g. [77, 150, 169], to date such solutions are not implemented in commodity systems.

One further possibility to circumvent data usage control infrastructures is to make use of media breaks, e.g. by taking pictures or videos of the screen or by sending sensitive documents to physical printers. Once usage controlled data has left the technical infrastructure in such a way, there hardly exist any means to still control the data's usage. However, once this data is re-digitized, e.g. by scanning a printout, it may become possible to detect the equivalence of certain data/documents and to reapply the corresponding policies. Yet, such procedures are far from being implemented and integrated with data usage control infrastructures and they would only be useful once the corresponding data re-enters a usage controlled system. Further, also legal and ethical concerns remain: it might very well be the case that some data classified as 'equivalent' to some usage controlled data was created by some other party that did not intend to usage control the data.

Render Usage Control Infrastructure Unusable. Another incentive for an attacker might be to render the usage control infrastructure or the usage controlled systems unusable, meaning that no further (satisfactory) usage of those systems is possible.

To achieve this goal, a first possibility is to deliberately 'leverage' the usage control infrastructure's overapproximations when performing data flow tracking. This way, it would be possible to taint large parts of the system, such that usage control policies apply to most data containers within the system. If the corresponding usage control policies are of such a kind that they entirely inhibit usage of the corresponding data, no further usage of the tainted data containers are possible. Notably, such an attack is usually limited to the resources for which the attacking user has write permissions, since only in this case he is actually able to propagate usage controlled data to those containers. Hence, proper access control can counteract this problem in most situations. However, when using shared file systems or if a non-privileged process communicates with a privileged process, the corresponding taints could indirectly be propagated to system resources to which the attacking user does actually not have access. E.g., if the attacking user's non-privileged process, such as a web browser, communicates with a privileged system service, such as a web server, which might later write to system files. From this examples it gets clear that corresponding system files, such as libraries or the operating system kernel, should be secured with particular care, e.g. by making those files read only for all processes.

Another possibility is to mount a (distributed) denial of service attack, which might be caused by both local and remote users. Either way, such an attack would issue massive amounts of system events and have them evaluated by the usage control infrastructure. Alternatively, it is possible to flood the infrastructure's publicly running RPC services with requests. Such attacks may lead to high system response times

and render the system unusable. In the worst case, too many events and/or requests might overflow the usage control infrastructure's internal request processing queue (cf. Section 4.1) or make the system run out of memory, essentially resulting in major system/infrastructure failures. Since in such a case no data usage is possible at all, soundness of the infrastructure (i.e., no uncontrolled usage of data) is not compromised by such attacks. However, availability of the data can then no longer be guaranteed. To counteract such attacks, it is useful to mutually authenticate all of the infrastructure's (remote) components whenever remote communication is happening. While such mechanisms might not entirely prevent such attacks, they introduce non-repudiation and consequently liabilities. Another possibility is to block corresponding entities entirely, e.g. by blocking their access at the operating system layer or by having the system's firewall block all requests from certain remote endpoints. However, certain problems remain. E.g., massive amounts of requests might in fact be legitimate in certain situations; sophisticated denial of service attacks are hard to counteract to date.

Lastly, it remains to note that further attacks not specifically targeted to data usage control infrastructures may be possible. Mostly, such attacks target assumptions (cf. Section 5.1.1) or technologies upon which this thesis builds. E.g., misbehaving Certificate Authorities (CAs), insecure operating systems, improper (physical) access control to usage controlled systems, and social engineering might pose significant threats.

5.1.3 Summary: Provided Guarantees

Under consideration of the assumptions presented in Section 5.1.1 as well as the security evaluation performed in Section 5.1.2, this section summarizes in which situations which guarantees are provided by the presented infrastructure. For this, in the following it is assumed that the assumptions from Section 5.1.1 hold and that the countermeasures proposed in Section 5.1.2 have in fact been implemented. Since the protection of usage controlled data is at the core of the developed infrastructure, the following considerations focus on this aspect.

Unintentional Attacks within Closed Environments. Considering a closed environment, e.g. regular end users without administrative privileges as in the insurance company use case presented in Section 1.5, and an unintentional attacker model, i.e. end users that do not actively try to circumvent the usage control infrastructure, it can be summarized that the presented solution detects and/or prevents (depending on the deployed policies) any policy violations in a reliable manner. For example, the presented infrastructure can prevent end users from accidentally sharing sensitive documents with unauthorized colleagues or business partners. While it is possible that some critical assumptions or possible attacks were missed in Sections 5.1.1 and 5.1.2, in this scenario users are not expected to intentionally try to circumvent the enforcement infrastructure. By considering regular users without administrative privileges it is

also very unlikely that these users unintentionally misconfigure system aspects which would open up further attacker vectors. If audits would reveal that some unintentional circumvention of the usage control infrastructure did indeed happen, then fixing the corresponding flaws and rolling out a new release of the infrastructure should not face any major organizational or technical issues. In sum, the presented infrastructure is able to reliably enforce usage control policies in the presence of an unintentional attacker model within closed environments. Lastly, it should be noted that such scenarios are omnipresent in business contexts in which employees are equipped with ready-to-use computing devices by the employer's IT department.

Unintentional Attacks within Open Environments. Considering an unintentional attacker model within open environments, e.g. if the usage control infrastructure is rolled out to open systems on which end users do have administrative privileges, the usage control infrastructure is still able to reliably detect and/or prevent illegitimate data usages. While in this scenario users are also not expected to intentionally circumvent the infrastructure, the fact that the environment is open and that the user has administrative privileges poses additional threats. E.g., the user might unintentionally misconfigure the system by messing with access control settings, by trusting invalid certificates, or by installing vulnerable versions of the usage control infrastructure, the operating system, or other essential software. In such a case it might be possible for external attackers to gain access to the system and to protected data. Further, negligent users or administrators might use insecure channels to exchange usage controlled data or use insecure on-disk encryption mechanisms, thus opening up further attack vectors for external attackers. Nevertheless, even within open environments the usage control infrastructure as presented in this thesis is able to reliably enforce data usage control policies as long as end users and system administrators do not act negligently, e.g. by misconfiguring the system or by not keeping their systems updated.

Intentional Attacks within Closed Environments. Intentional attackers are attackers that deliberately try to circumvent the usage control infrastructure with the goal to use usage controlled data in an uncontrolled manner. A scenario featuring an intentional attacker within a closed environment might be the insurance company scenario from Section 1.5 in which a disgruntled employee tries to leak sensitive data from its workstation to the public. Further, such scenarios are omnipresent when considering today's computing devices, such as tablets, mobile phones and integrated systems, on which regular end users seldom have administrative privileges. In such a case it must be assumed that the attacker actively tries to circumvent the usage control infrastructure. Consequently, any shortcomings, limitations, or bugs within the infrastructure might be actively looked for and, if present, be intentionally exploited. Since software is likely never to be free of flaws, motivated attackers will likely always be able achieve their goal one way or another. Thereby, in particular media breaks as described in Section 5.1.2 constitute a significant threat, as it remains open whether

there will ever exist reliable technical solutions to counteract this attack vector. In sum, the deployment of a distributed data usage control solution within closed environments can pose significant barriers for intentional attackers. While these barriers might never be able to provide absolute guarantees that usage controlled data is only used in compliance with its policies, they might at least make their circumvention uneconomical for attackers.

Intentional Attacks within Open Environments. Considering completely open environments and intentional attackers, it is questionable whether usage control solutions as the one presented in this thesis will ever be able to provide any guarantees. In such a scenario intentional attackers are in full control of their system and may thus switch off the usage control infrastructure, tamper with it, or reengineer it with the goal to use sensitive data without respecting the corresponding policies. While technical solutions such as TPM, NGSCB and SmartCards are capable of measuring the integrity of the operating systems as well as usage control infrastructures, it can be doubted that such technologies will be in place on systems on which attackers do have administrative privileges and intentionally try to circumvent data usage control technologies.

5.2 Cross-System Data Flow Tracking and Policy Propagation

A preliminary version of the content of this section has been published in [94].

The goal of this section is to understand which performance overheads are introduced when tracking data flows across systems and propagating the corresponding data usage policies (**RQ1**) as conceptualized in Section 3.2 and implemented in Section 4.2 (requirement **R4**).

For this, recap that whenever networking-related system calls (e.g. *socket*, *accept*, *connect*, *write*, *read*) are observed, the PIP determines whether usage controlled data is transferred to a remote system. If this is the case, then the DMPs of the two involved systems (i.e. the sending and the receiving system) communicate in order to coordinate this fact of data exchange. In addition, the data usage control policies of the exchanged data is sent to the receiving system for further enforcement.

In order to measure the performance overhead introduced by this procedure, several case studies along the running example from Section 1.5 have been performed: several usage controlled files of different sizes were transferred between two systems using different protocols, applications and bandwidths. The motivation was to simulate multiple real-world data transfer scenarios (e.g., transfer 128MB via HTTP on a fast network of 300Mbps, transfer 1KB via FTP on a slow network of 10Mbps) and to understand which parameter influences the imposed performance overhead to which extent. In order to measure performance overheads, a client and a server application transferred those files and the execution times of those file transfers were measured.

Further, the same experiments were performed with different stripped-down versions of the infrastructure as well as for native file transfers. The motivation for this

was twofold: First, measuring the performance when cross-system data flow tracking and policy propagation is disabled serves as a baseline for comparing the performance overheads in the presence of remote data flows. Second, the PEP for Unix-like systems (cf. Section 4.1.4) as well as major parts of the local usage control infrastructure (i.e. PDP, PIP, PMP, cf. Section 4.1), upon which this evaluation is based, have been developed within this thesis.

After detailing the system setup, identifying relevant parameters, and describing how the experiments were executed, the actual evaluation results are presented.

System Setup. For this set of experiments, two machines, also called client and server, were set up. Each machine featured a 2x2.6GHz CPU and 4GB RAM. The machines run Linux Mint 17.1 64 bit, Kernel 3.13.0; Cassandra was used in version 2.1.2. Both the client and the server machine run one fully functional Controller as described in Section 4.1.3. The PEP described in Section 4.1.4 monitored the applications performing the file transfer and consequently all observed system calls were signaled to the corresponding local Controller. For the communication between the PEP and the Controller Thrift was used in version 0.9.2. All cross-system communication was encrypted using TLS. Cassandra used a consistency level of *Quorum*.

Parameters. When running and evaluating the experiments, the following parameters turned out to influence the performance overheads:

- (i) The *kind* of monitoring and/or data flow tracking being performed (k). $k = cross$ indicates that data was tracked across systems and that the corresponding policies were propagated accordingly. To understand which parts of the infrastructure are to which extent responsible for the measured overheads, further measurements with a stripped-down version of the infrastructure were performed: $k = local$ indicates that data was only tracked locally and no data and no policies were ever tracked/propagated across systems. Hence, all functionality related to cross-system data flow tracking and policy propagation was disabled. $k = signal$ indicates that all system calls were intercepted, preprocessed and signaled from the PEP to the Controller; no further processing of the system calls by the PDP and/or PIP was performed. In other words, essentially all of the Controller's functionality was disabled. Finally, $k = native$ denotes the native execution of the remote file transfer without any monitoring. In summary:

k	Description
<i>native</i>	Native Execution
<i>signal</i>	Monitor and signal to Controller
<i>local</i>	Monitor and signal to Controller, track local data flows
<i>cross</i>	Monitor and signal to Controller, track local and remote data flows

- (ii) The *size* of the files being transferred across systems. File sizes of 1KB, 1MB, 128MB and 512MB were used.
- (iii) The *bit rate* of the underlying network (r). Bit rates of 10Mbps, 50Mbps, 100Mbps and 300Mbps were used.
- (iv) The *policy* being attached to the tracked data and consequently being enforced and evaluated. Since policy evaluation is further evaluated in Section 5.3, the policy being used for this set of experiments had a trivial condition of $\varphi = \textit{false}$ and did thus immediately allow all events.
- (v) The protocols and *applications* being used for the file transfer. The files were transferred using protocols HTTPS, FTPS and SSH. The following client and server applications were used:

Protocol	Server application(s)	Client applications(s)
HTTPS	Apache2 HTTP Server 2.4.7 [200]	wget 1.15 [59]
	nginx 1.4.6 [157]	cURL 7.35.0 [38]
		aria2c 1.18.1 [209]
FTPS	ProFTPD 1.3.5 [145]	cURL 7.35.0
	vsftpd 3.0.2 [52]	LFTP 4.4.13 [129]
SSH	OpenSSH 6.6.1 [202]	SCP 6.6.1 [121]

As described in Section 4.1.4, the PEP intercepts all system calls within those applications and signals them to the Controller. For this reason the amount of system calls being issued by those applications is of particular importance. In particular, each *write* system call observed within the server application might trigger the transfer of usage controlled data to the client application, thus resulting in additional remote communication between the server's and client's DMP. Hence, Table 5.1 shows for each of those applications and for all file sizes how many system calls are *signaled* by the PEP to the Controller (c_s^Σ), as well as the amount of system calls that are *processed* by the PDP/PIP (c_p)¹. Because the policy deployed for these experiments allows all signaled events, the total amount of events being processed by the PDP/PIP (c_p) corresponds to the amount of system calls being signaled by the PEP (c_s^Σ) *plus* the amount of signaled *write* system calls (c_s^w), $c_p = c_s^\Sigma + c_s^w$.

Experiment Execution. For each measurement all of the above parameters were fixed. For $k \in \{\textit{signal}, \textit{local}, \textit{cross}\}$, one fully functional Controller was started on both the client and the server machine. After startup of the usage control infrastructure, the server process was started on the server machine, offering four different files of sizes 1KB, 1MB, 128MB, and 512MB for download. Once the server process finished initialization, a compatible client process was started on the client machine with the

¹Note that $c_s^\Sigma \leq c_p$ because for system call *write* only intended system calls are signaled by the PEP (cf. Section 4.1.4). However, as described in Section 4.1.4, the PDP/PIP also process the corresponding actual system call as soon as the intended system call is allowed.

Table 5.1: Amount of intercepted and signaled system calls.

File size	Program	Events signaled to PDP/PIP				Events processed by PDP/PIP (c_p)
		<i>read</i> (c_s^r)	<i>write</i> (c_s^w)	others (c_s^o)	total (c_s^Σ)	
1KB	Apache2	8	4	19	31	35
	nginx	14	4	7	25	29
	vsftpd	414	191	235	840	1031
	ProFTPD	203	131	324	658	789
	OpenSSH	402	118	2865	3385	3503
	wget	38	4	57	99	103
	cURL	34	5	71	110	115
	aria2c	53	11	77	141	152
	LFTP	130	30	120	280	310
	SCP	177	20	204	401	421
1MB	Apache2	8	67	19	94	161
	nginx	76	68	7	151	219
	vsftpd	564	314	234	1112	1426
	ProFTPD	267	263	324	854	1117
	OpenSSH	598	298	2860	3756	4054
	wget	230	259	57	546	805
	cURL	291	132	71	494	626
	aria2c	245	74	77	396	470
	LFTP	384	93	120	597	690
	SCP	441	203	208	852	1055
128MB	Apache2	7	8195	139	8341	16536
	nginx	8204	8196	7	16407	24603
	vsftpd	20884	16570	234	37688	54258
	ProFTPD	8396	9240	330	17966	27206
	OpenSSH	27315	24293	2867	54475	78768
	wget	24614	32771	57	57442	90213
	cURL	32803	16388	71	49262	65650
	aria2c	24629	8202	77	32908	41110
	LFTP	32890	8221	120	41231	49452
	SCP	39435	20562	208	60205	80767
512MB	Apache2	7	32771	523	33301	66072
	nginx	32780	32794	7	65581	98375
	vsftpd	82324	65722	235	148281	214003
	ProFTPD	32977	36273	360	69610	105883
	OpenSSH	108025	96541	2852	207418	303959
	wget	98342	131075	57	229474	360549
	cURL	131107	65540	71	196718	262258
	aria2c	98357	32781	103	131241	164022
	LFTP	131193	32797	120	164110	196907
	SCP	157170	81396	208	238774	320170

task to download one of the provided files. The total time needed for downloading the file was measured by invoking the client process with the Linux command line tool `time` [123]. Note that for each measurement the same value for k was used on the client and the server side. Further note that all data transfers made use of secure communication channels by using protocols HTTPS, FTPS and SSH. For $r = 300Mbps$ each measurement was repeated 40 times; for $r \in \{50Mbps, 100Mbps\}$ each measurement was repeated 30 times; for $r = 10Mbps$ each measurement was repeated 5 times.

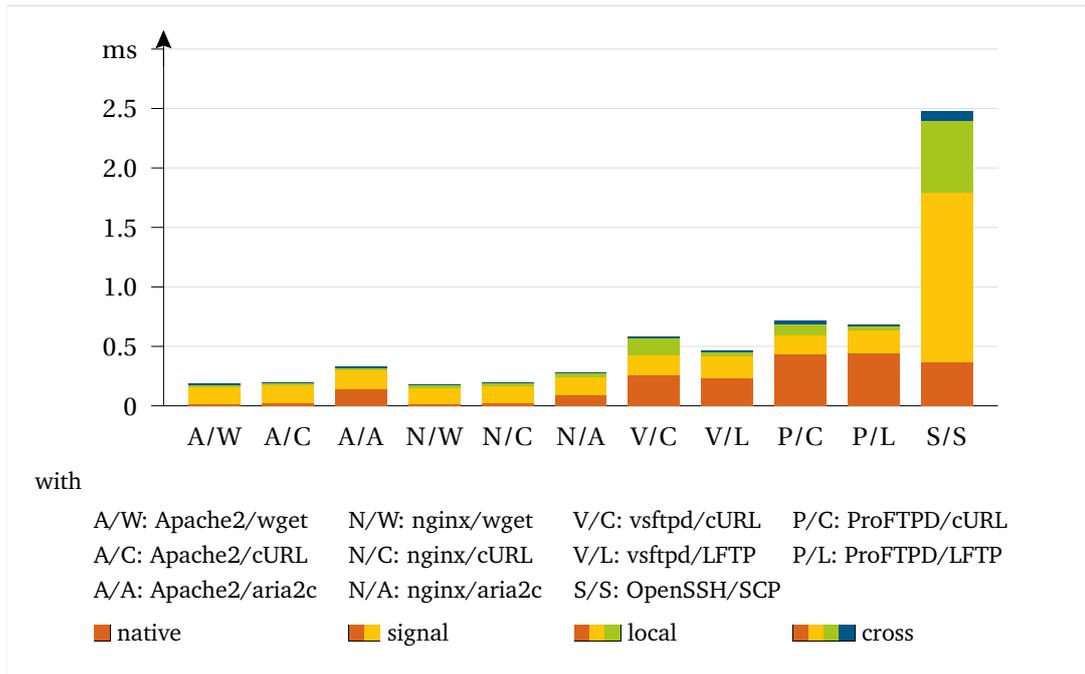
Results in a Nutshell. The evaluation results reveal that the imposed relative performance overheads span an enormous range between 0.4% and 1000%, depending on concrete parameter values being considered. That said, most of this overhead is attributed to local aspects of the infrastructure, i.e. signaling of system events to the Controller and evaluation of these events by the PDP/PIP. The overheads imposed by cross-system data flow tracking and policy propagation were in most cases negligible. Whether any such overheads are acceptable in practice clearly depends on the concrete scenario being considered. This is further discussed in Section 5.2.6.

The subsequent sections present the evaluation results in more detail. The results of cross-system data flow tracking and policy propagation are compared to different stripped-down versions of the infrastructure as explained above, as well as to native execution. This way, it is possible to analyze which parts of the infrastructure cause which parts of the overall performance overhead. More detailed evaluation results are provided in Appendix C.

5.2.1 Transferring Files of Size 1KB

In the first set of experiments, files of size 1KB were transferred between two systems and the corresponding file transfer times were measured. Table 5.2 provides the median file transfer times and the corresponding standard deviation in milliseconds (ms) for different client/server combinations as well as for native file transfers and different versions of the usage control infrastructure. Figure 5.1 visualizes the median values of Table 5.2 as bar charts. In addition, Figure C.2 (Appendix C.1) provides some more details by plotting all measurement results as boxplots.

When transferring 1KB files, the first observation is that the network bit rate does not have any influence on any file transfer times: Given a client and a server application the time needed to transfer a 1KB file is the same for network bit rates of $r = \{10Mbps, 50Mbps, 100Mbps, 300Mbps\}$. For this reason Table 5.2 and Figure 5.1 do not further differentiate between different bit rates. This fact has the additional consequence that each presented number is based on 105 individual measurements: 5 for $r = 10Mbps$, 30 for $r \in \{50Mbps, 100Mbps\}$, and 40 for $r = 300Mbps$.

Figure 5.1: Transfer times for a 1KB file.**Table 5.2: Time and standard deviation [ms] to transfer a 1KB file.**

Bit rate	Protocol	Server	Client	native (■)	signal (■)	local (■)	cross (■)
any	HTTPS	Apache2	wget	17 ± 2	152 ± 18	171 ± 19	182 ± 22
			cURL	25 ± 2	166 ± 22	190 ± 20	193 ± 20
			aria2c	140 ± 22	296 ± 41	318 ± 38	324 ± 38
		nginx	wget	17 ± 1	143 ± 18	166 ± 18	171 ± 20
			cURL	24 ± 2	164 ± 20	179 ± 23	188 ± 22
			aria2c	90 ± 16	244 ± 32	264 ± 37	273 ± 34
	FTPS	vsftpd	cURL	259 ± 2	431 ± 33	566 ± 35	576 ± 31
			LFTP	232 ± 14	415 ± 34	450 ± 32	464 ± 32
		ProFTPD	cURL	432 ± 3	594 ± 30	692 ± 28	708 ± 33
			LFTP	444 ± 25	630 ± 47	664 ± 47	681 ± 47
	SSH	OpenSSH	SCP	368 ± 11	1786 ± 112	2398 ± 63	2478 ± 67

In terms of *native* file transfer times (■), i.e. when no monitoring by the usage control infrastructure is involved, it turns out that HTTPS transfers small files much faster than FTPS and SSH. There are two—related—explanations for this: First, FTPS and SSH are more complex than HTTPS. For FTPS the reason is that two TCP communication channels are established—one for payload data and one for management/command data. For SSH the reason is that it constitutes a complex general purpose protocol that also performs client-side user authentication. This additional complexity of FTPS/SSH w.r.t. HTTPS is also emphasized by the amount of system calls being issued by the corresponding client and server applications: as Table 5.1 shows, vsftpd, ProFTPD (FTPS) and OpenSSH (SSH) issue much more system calls for transferring a 1KB file

than Apache2 and nginx (HTTPS); e.g. while Apache2 issues 35 system calls, OpenSSH issues 3503 system calls to transfer a 1KB file. Further, the measurements reveal that aria2c performs badly when compared to wget and cURL, indicating that initiation of a file transfer is expensive for this application. Along the same lines, Table 5.2 and Figure 5.1 reveal that initiation of a file transfer is more expensive when using the ProFTPD rather than the vsftpd FTP server.

While SSH and the FTPS client/server applications perform approximately similar when executed natively, Table 5.2 and Figure 5.1 show that monitoring of those file transfers by the usage control infrastructure (■) imposes much more overhead for SSH than for FTPS. Again, one reason is the significant difference in the amount of system calls that are signaled to the Controller and consequently processed by the PDP and the PIP. Interestingly, the absolute overhead for signaling system calls to the Controller (■) is similar for all client/server combinations despite SSH and averages at $156ms \pm 30ms$. In contrast, for SSH the absolute overhead to signal events is $\sim 1418ms$. In terms of performing local data flow tracking (■), the additional absolute overhead for most client/server combinations is around $25ms \pm 10ms$. Exceptions to this can be observed when cURL is used as a client application ($116ms \pm 19ms$), as well as for SSH ($\sim 612ms$). Finally, additional overhead imposed by cross-system data flow tracking (■) as described in Section 4.2 is present but hardly observable. This additional overhead averages at $\sim 10ms$, which is, however, within the standard deviation.

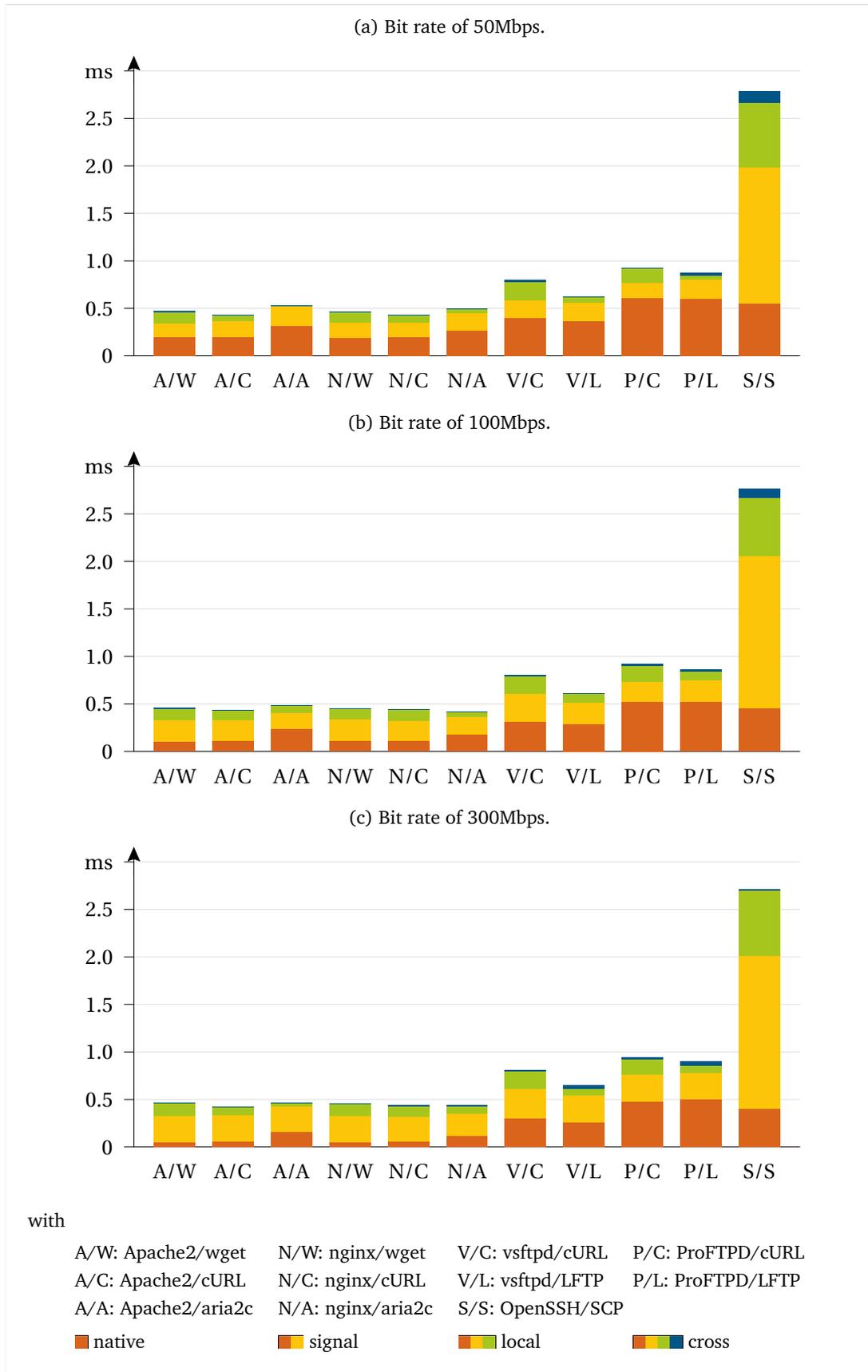
In conclusion, transferring a very small file of size 1KB imposes significant initial management overhead which can not be amortized during the very short file transfer time. As an overall result, transferring a file of size 1KB results in an overall absolute overhead (■) ranging from 154ms (nginx/wget) to 2110ms (OpenSSH/SCP), and an overall relative overhead ranging from 53% (ProFTPD/LFTP) to 971% (Apache2/wget). Those numbers are subject to further discussion in Section 5.2.6. All overall absolute and relative overheads for all client/server combinations are provided in Table C.1 (Appendix C).

5.2.2 Transferring Files of Size 1MB

At a first glance the performance measurement results for transferring files of size 1MB (Table 5.3 and Figure 5.2, as well as Appendix C.2 for additional details) look similar to the results of transferring 1KB files as discussed above. However, some remarkable differences can be observed.

First of all, for files of 1MB the influence of different network bit rates is observable: when comparing the *native* file transfer times (■) for different bit rates (Table 5.3 and Figures 5.2a to 5.2c), it turns out that the performance increases for higher network bit rates. However, it depends on the protocol and the client/server applications to which extent the native file transfer time is accelerated: For FTPS and SSH the impact of the increased bit rate is not as significant as for HTTPS. The reason is that the

Figure 5.2: Transfer times for a 1MB file.



expensive startup phase of those protocols can still not be amortized when transferring files of size 1MB. This argument is emphasized by the amount of issued system calls: As Table 5.1 shows, for a file size of 1MB the absolute difference in the amount of system calls between applications is still significant, however, in comparison with the 1KB case, their relative difference is decreasing.

Despite this speedup of the native file transfer times, file transfer times do *not* improve with higher bit rates if the applications are monitored by the usage control infrastructure (■ ■ ■ ■). For example, independent of the network's bit rate vsftpd/cURL takes approximately 0.8 seconds to transfer a file of size 1MB, while OpenSSH/SCP takes approximately 2.8 seconds. This surprising observation leads to the fact that both absolute and relative performance overheads imposed by the usage control infrastructure (■ ■ ■ ■) *increase* with higher network bit rates. Considering all client/server combinations but the exceptionally expensive OpenSSH/SCP case, it *seems* that the overall time needed to signal events from the PEP to the Controller (■) increases by approximately 40ms if the bit rate is increased from 50Mbps to 100Mbps, and by another 40ms if the bit rate is increased from 100Mbps to 300Mbps. At the same time, the overhead imposed by local data flow tracking (■) remains constant despite the change in the network bit rate.

Considering these observations, the question remains why signaling events from the PEP to the Controller is slower in case the network bit rate increases. The answer is that it is not. However, for slower bit rates the client's and server's system calls to the TCP communication channel block more often and longer due to the channel's bad I/O performance. Since the PEP intercepts those intended system calls right before their execution by the kernel, the signaling of the system calls to the Controller, as well as their processing by the PDP/PIP, happens at the time when the process would be waiting for the TCP channel to become ready to read/write. This way, the usage control infrastructure is able to utilize those periods in which the actual processes idle.

In terms of tracking data flows and policies across systems (■), Table 5.3 shows that the additional imposed overhead is hardly measurable.

In sum, for transferring files of size 1MB the overall absolute overhead imposed by the usage control infrastructure (■ ■ ■ ■) ranges from 212ms (nginx/aria2c/50Mbps) to 2317ms (OpenSSH/SCP/300Mbps), whereas the relative overhead ranges from 45% (ProFTPD/LFTP/50Mbps) to 816% (Apache2/wget/300Mbps). All overall absolute and relative for all client/server/bit rate combinations are provided in Table C.2.

Table 5.3: Time and standard deviation [ms] to transfer a 1MB file.

Bit rate	Server	Client	native (■)	signal (■)	local (■)	cross (■)	
50Mbps	Apache2	wget	194 ± 2	347 ± 30	450 ± 42	450 ± 47	
		cURL	202 ± 2	355 ± 29	416 ± 41	422 ± 38	
		aria2c	319 ± 21	504 ± 51	532 ± 31	536 ± 36	
	nginx	wget	194 ± 2	344 ± 33	449 ± 37	456 ± 41	
		cURL	201 ± 2	344 ± 27	423 ± 41	427 ± 37	
		aria2c	270 ± 16	443 ± 33	486 ± 40	482 ± 27	
	vsftpd	cURL	397 ± 2	589 ± 35	769 ± 45	804 ± 39	
		LFTP	369 ± 10	555 ± 32	609 ± 40	617 ± 42	
	ProFTPD	cURL	610 ± 2	767 ± 39	908 ± 56	915 ± 48	
		LFTP	602 ± 18	792 ± 33	838 ± 54	872 ± 50	
	OpenSSH	SCP	550 ± 13	1988 ± 118	2667 ± 83	2807 ± 65	
	100Mbps	Apache2	wget	107 ± 1	333 ± 48	428 ± 43	442 ± 48
			cURL	115 ± 2	337 ± 47	430 ± 38	426 ± 33
			aria2c	234 ± 24	417 ± 32	478 ± 55	477 ± 53
		nginx	wget	107 ± 2	355 ± 55	428 ± 45	448 ± 40
cURL			114 ± 2	316 ± 35	434 ± 41	421 ± 41	
aria2c			183 ± 17	361 ± 44	411 ± 31	409 ± 39	
vsftpd		cURL	312 ± 2	614 ± 44	778 ± 44	787 ± 40	
		LFTP	286 ± 17	514 ± 55	595 ± 44	596 ± 40	
ProFTPD		cURL	520 ± 2	728 ± 46	907 ± 48	925 ± 46	
		LFTP	524 ± 21	746 ± 57	844 ± 47	854 ± 52	
OpenSSH		SCP	458 ± 12	2042 ± 110	2678 ± 82	2764 ± 52	
300Mbps		Apache2	wget	49 ± 9	337 ± 49	451 ± 52	449 ± 59
			cURL	58 ± 8	342 ± 45	427 ± 46	402 ± 48
			aria2c	163 ± 27	433 ± 49	467 ± 51	467 ± 50
		nginx	wget	47 ± 9	330 ± 48	458 ± 45	420 ± 54
	cURL		57 ± 8	317 ± 43	425 ± 52	437 ± 51	
	aria2c		114 ± 23	355 ± 49	428 ± 58	431 ± 58	
	vsftpd	cURL	301 ± 10	612 ± 53	786 ± 50	804 ± 51	
		LFTP	258 ± 27	526 ± 72	592 ± 41	614 ± 65	
	ProFTPD	cURL	472 ± 7	771 ± 67	914 ± 50	928 ± 57	
		LFTP	500 ± 31	749 ± 60	829 ± 52	854 ± 73	
	OpenSSH	SCP	399 ± 11	2040 ± 121	2730 ± 93	2716 ± 76	

5.2.3 Transferring Files of Size 128MB

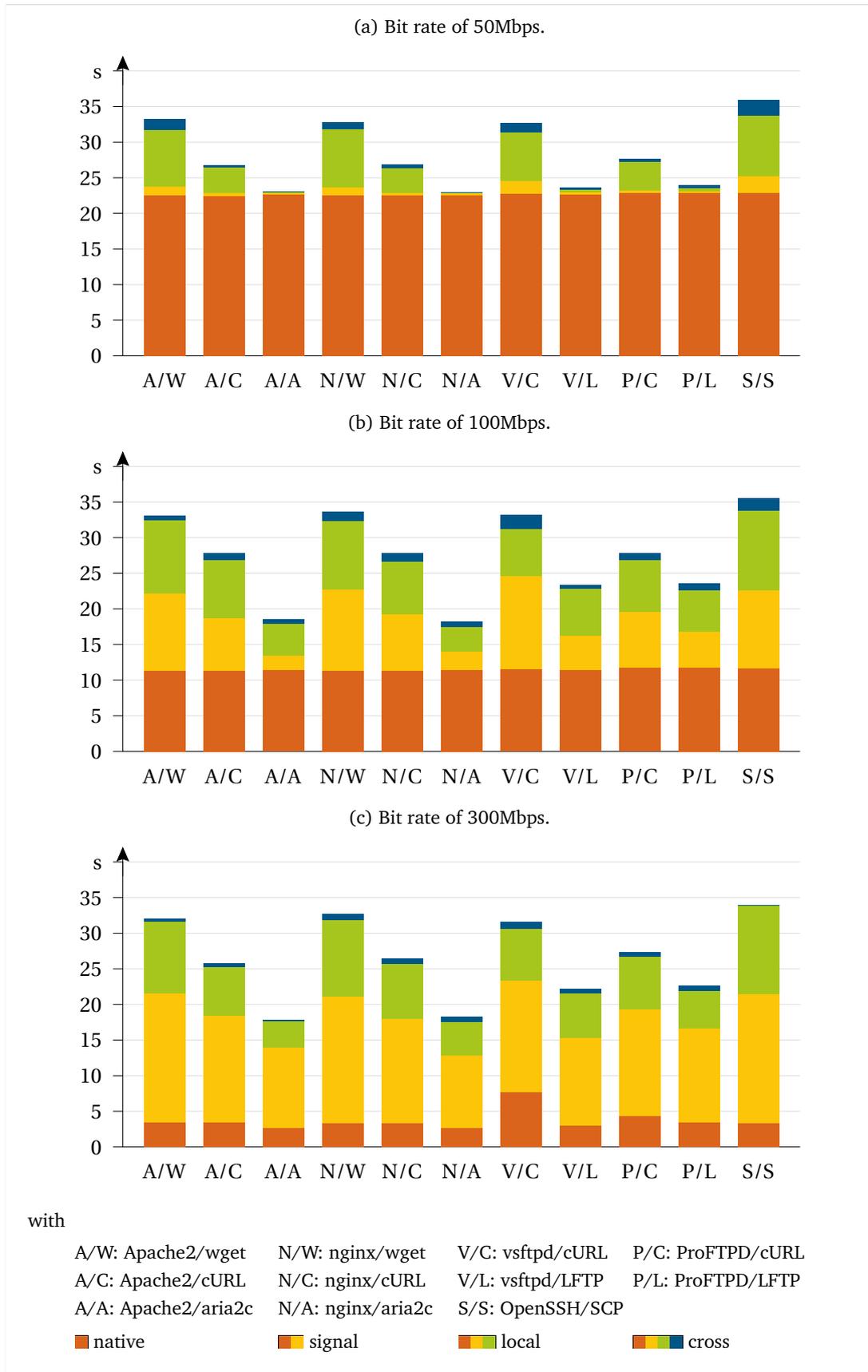
When transferring files of size 128MB (Figure 5.3 and Table 5.4 as well as Appendix C.3 for additional details), the first observation is that the network bit rate has a significant impact onto the *native* (■) file transfer times. In fact, in most of the considered scenarios the bit rate r is the limiting factor for native file transfer times, resulting in repeated measurement results of $\sim 22.7s$ for $r = 50Mbps$ (Figure 5.3a), $\sim 11.5s$ for $r = 100Mbps$ (Figure 5.3b) and $\sim 3.7s$ for $r = 300Mbps$ (Figure 5.3c). The most prominent exception is the combination of vsftpd/cURL, which seems not to be able to utilize the entire bit rate of 300Mbps, resulting in comparatively slow native file transfer times of $7.7s$. Interestingly, aria2c, which performed badly for files of size 1KB and 1MB, performs slightly faster than wget and cURL for a file size of 128MB and for a bit rate of 300Mbps.

Similar to the above case of 1MB, Figure 5.3 and Table 5.4 show that for each client/server combination the file transfer time is constant for different network bit rates if the usage control infrastructure (■) is *enabled*. There are two exceptions to this observation: For Apache2/aria2c and nginx/aria2c the usage controlled file transfer (■) for $r = 50Mbps$ takes $\sim 23s$, while the transfer times for $r = 100Mbps$ and $r = 300Mbps$ are $\sim 18.2s$. These exceptions are due to the bandwidth's physical limitation in the 50Mbps case, in which the file transfer can under no circumstances be faster than $20.48s (= 128MB/50Mbps)$.

Again, and similar to the 1MB case, it can be observed that the overhead for signaling system calls to the Controller increases with higher bit rates (Table 5.4 and Figure 5.3, ■). At the same time, and as discussed above, for each usage controlled client/server combination the total file transfer time (■) is independent of the network's bit rate. Again, the explanation for this phenomenon is that the signaling to the Controller is performed while the corresponding system calls block and wait for the communication channel to become ready to read/write.

Transferring larger files of 128MB allows to further investigate the overhead imposed by the usage control infrastructure, since exceptional operations performed at file transfer startup, which were predominant for the 1KB/1MB case, are negligible. This allows to calculate the *additional overhead per system call* for different scenarios. For these calculations the measured file transfer times (Table 5.4) and the amount of system calls signaled from the *client application* (Table 5.1) are taken as a basis. The reason for considering the client application's system calls rather than the server's or a combination thereof, is that additional experiments showed that for large files the client application is the bottleneck for the file transfer times. This argument is underpinned by two further facts: First, the used HTTPS/FTPS/SSH server applications are usually deployed on a large scale and serve many clients simultaneously. Therefore, these applications are much more tweaked for performance than the corresponding client applications. Second, looking at the signaling overheads (■ in Figures 5.3b and 5.3c) and

Figure 5.3: Transfer times for a 128MB file.



comparing them with the amount of system calls of the corresponding client applications (Table 5.1), it seems that there is a correlation between these numbers.

In the following, the additional overhead for signaling system calls to the Controller (■) is analyzed: Ranging over all client/server combinations, for a bit rate of 50Mbps the additional overhead *per system call* is $0.016ms \pm 0.012ms$; for a bit rate of 100Mbps it is $0.156ms \pm 0.10ms$; for a bit rate of 300Mbps it is $0.314ms \pm 0.02ms$. First, these numbers reveal that for a low bit rate of 50Mbps the signal time per system call is relatively low ($0.016ms$) and relatively fluctuant ($\pm 0.012ms$). The reason is that for some client applications (e.g., aria2c, LFTP) a relatively small amount of system calls (32908 and 41231, respectively) is signaled to the Controller, having the effect that they can for the most part be signaled while the client process is waiting for the busy communication channel (resulting in an overhead per system call of $0.016ms - 0.012ms = 0.004ms$). For other client applications (e.g., wget, SCP), however, more system calls must be signaled (57442 and 60205, respectively) and not all signaling can be performed while the communication channel is busy (resulting in an overhead per system call of $0.016ms + 0.012ms = 0.028ms$). Further, these numbers reveal that for a high bit rate of 300Mbps the signal time per system call is comparatively high ($0.314ms$) and relatively constant ($\pm 0.02ms$) for all considered client/server combinations. The reason for the high signaling time is that system calls on the communication channel only block for very short amounts of time due to the high network bit rate. The constancy of these calculation results across different client/server combinations confirms that calculating the signaling time per system call on the basis of the client applications' amount of system calls was a reasonable choice.

In the same manner the additional overhead *per system call* for local data flow tracking (■) can be computed. The obtained results follow a similar pattern as above: for a bit rate of 50Mbps the overhead is $0.060ms \pm 0.059ms$; for a bit rate of 100Mbps it is $0.120ms \pm 0.024ms$; for a bit rate of 300Mbps it is $0.136ms \pm 0.031ms$. The explanation of these results follows the same arguments as above: E.g., for a bit rate of 50Mbps some applications (e.g. aria2c) are able to leverage the waiting times for the communication channel to perform not only signaling of signals to the Controller, but also to have the corresponding system calls processed by the PDP and PIP, resulting in an overhead as small as $0.060ms - 0.059ms = 0.001ms$ per system call.

The additional absolute overhead imposed by cross-system data flow tracking (■) averages at 1s to 2s. However, this overhead is within the measurements' standard deviation and could thus not be reliably measured in all cases.

Concluding this discussion, the overall absolute overhead imposed by the usage control infrastructure to transfer a file of size 128MB ranges from 0.38s (Apache2/aria2c/50Mbps) to 30.8s (OpenSSH/SCP/300Mbps), whereas the overall relative overheads range from 2% (Apache2/aria2c/50Mbps) to 931% (OpenSSH/SCP/300Mbps). Again, these results are subject to discussion in Section 5.2.6. All overall absolute and relative for all client/server/bit rate combinations are provided in Table C.3.

Table 5.4: Time and standard deviation [s] to transfer a 128MB file.

Bit rate	Server	Client	native (■)	signal (■)	local (■)	cross (■)	
50Mbps	Apache2	wget	22.51 ± 0.00	23.30 ± 1.29	31.53 ± 1.23	33.20 ± 1.17	
		cURL	22.52 ± 0.00	22.72 ± 0.25	26.46 ± 0.94	26.98 ± 1.01	
		aria2c	22.68 ± 0.02	22.89 ± 0.04	22.96 ± 0.06	23.06 ± 0.05	
	nginx	wget	22.51 ± 0.00	23.11 ± 1.12	32.00 ± 1.01	33.01 ± 0.94	
		cURL	22.52 ± 0.00	22.74 ± 0.31	26.30 ± 0.87	27.05 ± 1.04	
		aria2c	22.63 ± 0.02	22.84 ± 0.05	22.91 ± 0.06	23.01 ± 0.05	
	vsftpd	cURL	22.73 ± 0.00	24.13 ± 1.14	31.30 ± 0.81	32.66 ± 1.20	
		LFTP	22.72 ± 0.01	22.93 ± 0.04	23.13 ± 0.56	23.42 ± 0.61	
	ProFTPD	cURL	22.93 ± 0.00	23.14 ± 0.27	26.82 ± 1.12	27.77 ± 1.14	
		LFTP	22.96 ± 0.02	23.16 ± 0.05	23.33 ± 0.59	23.73 ± 0.78	
	OpenSSH	SCP	22.86 ± 0.01	24.44 ± 1.38	34.04 ± 1.15	35.95 ± 1.33	
	100Mbps	Apache2	wget	11.30 ± 0.02	22.20 ± 0.89	32.49 ± 1.19	33.45 ± 1.59
			cURL	11.30 ± 0.01	18.50 ± 0.70	26.79 ± 1.19	27.80 ± 1.21
			aria2c	11.45 ± 0.03	13.27 ± 0.67	17.79 ± 0.73	18.37 ± 0.96
		nginx	wget	11.29 ± 0.02	22.39 ± 1.43	32.11 ± 1.72	33.61 ± 1.58
cURL			11.30 ± 0.01	19.19 ± 1.32	26.48 ± 0.95	27.75 ± 1.52	
aria2c			11.40 ± 0.02	13.54 ± 1.08	17.50 ± 0.60	18.23 ± 0.86	
vsftpd		cURL	11.49 ± 0.01	24.18 ± 1.79	31.24 ± 0.91	33.25 ± 1.37	
		LFTP	11.50 ± 0.02	16.17 ± 1.18	22.43 ± 1.06	23.40 ± 0.85	
ProFTPD		cURL	11.72 ± 0.01	19.37 ± 1.17	26.85 ± 0.90	28.06 ± 0.87	
		LFTP	11.74 ± 0.04	16.16 ± 1.57	22.57 ± 0.96	23.71 ± 0.83	
OpenSSH		SCP	11.63 ± 0.02	22.24 ± 1.13	33.90 ± 1.33	36.30 ± 2.05	
300Mbps		Apache2	wget	3.50 ± 0.89	21.62 ± 1.84	32.40 ± 1.58	32.53 ± 1.73
			cURL	3.45 ± 0.97	18.54 ± 1.76	25.81 ± 1.35	26.59 ± 1.46
			aria2c	2.69 ± 0.63	13.72 ± 1.50	18.02 ± 1.04	18.02 ± 0.88
		nginx	wget	3.33 ± 0.93	21.42 ± 1.25	32.68 ± 1.78	32.60 ± 2.40
	cURL		3.35 ± 0.81	18.22 ± 1.09	26.07 ± 1.28	26.62 ± 1.77	
	aria2c		2.61 ± 0.69	13.16 ± 0.81	17.65 ± 0.94	18.53 ± 1.08	
	vsftpd	cURL	7.77 ± 1.19	23.87 ± 1.78	31.29 ± 1.63	31.87 ± 1.62	
		LFTP	2.98 ± 0.78	15.72 ± 1.25	21.95 ± 1.01	22.66 ± 0.99	
	ProFTPD	cURL	4.40 ± 1.01	19.12 ± 1.83	26.99 ± 1.48	27.77 ± 1.72	
		LFTP	3.46 ± 0.68	16.32 ± 1.51	22.08 ± 1.22	22.69 ± 1.17	
	OpenSSH	SCP	3.31 ± 0.63	21.03 ± 1.54	34.58 ± 1.81	34.11 ± 1.55	

5.2.4 Transferring Files of Size 512MB

Transferring even larger files of 512MB (Figure 5.4 and Table 5.5 as well as Appendix C.4 for further details) confirms the above trends and observations. Again, native transfer times (■) are limited by the network bit rate and they are constant for each bit rate for all client/server combinations (50Mbps: $\sim 90s$, 100Mbps $\sim 45s$, 300Mbps: $\sim 13s$). As above, vsftpd/cURL on 300Mbps poses a significant exception. Again, the overall file transfer time if the usage control infrastructure is enabled (■) is constant for each client/server combination for different bit rates.

When calculating the times needed to signal (■) each single system call to the Controller, the results are insignificantly different to the 128MB case: for 50Mbps the overhead per system call is $0.017ms \pm 0.015ms$, for 100Mbps it is $0.161ms \pm 0.10ms$, and for 300Mbps it is $0.314ms \pm 0.02ms$. Also, the time that is needed to process the system calls by the PDP/PIP (■) is similar to the 128MB case: for 50Mbps the overhead per system call is $0.056ms \pm 0.054ms$, for 100Mbps it is $0.122ms \pm 0.015ms$, and for 300Mbps it is $0.126ms \pm 0.02ms$. The explanations for these results are equivalent to what has already been discussed.

Cross-system data flow tracking (■) imposed an additional overhead of up to 6s. Again, this overhead was within the standard deviation and could not be reliably measured; e.g. for nginx/wget/100Mbps no additional overhead was measurable.

Lastly, the overall absolute overhead imposed by the usage control infrastructure to transfer a file of size 512MB ranges from 0.85s (Apache2/aria2c/50Mbps) to 118.98s (OpenSSH/SCP/300Mbps), whereas the overall relative overheads range from 1% (Apache2/aria2c/50Mbps) to 1000% (OpenSSH/SCP/300Mbps).

Having analyzed the performance measurement results for different client/server combinations, file sizes and bit rates, it turned out that the overhead imposed by the usage control infrastructure is generally lower for slow network bit rates. As explained before, the reason is that in this case most of the usage control related tasks (i.e. signaling of the system calls to the Controller as well as local/remote data flow tracking and policy propagation) are performed while the actual client and server processes would be waiting for the TCP communication channel to become ready to read/write. To understand whether the above observations generalize, additional evaluations with a very low network bit rate of 10Mbps were performed. The corresponding results are presented in the subsequent section.

Figure 5.4: Transfer times for a 512MB file.



Table 5.5: Time and standard deviation [s] to transfer a 512MB file.

Bit rate	Server	Client	native (■)	signal (■)	local (■)	cross (■)	
50Mbps	Apache2	wget	90.01 ± 0.2	92.47 ± 5.1	130.08 ± 3.7	131.37 ± 6.0	
		cURL	90.04 ± 0.3	90.73 ± 1.1	106.08 ± 3.6	106.65 ± 2.5	
		aria2c	90.29 ± 0.2	90.66 ± 0.4	91.11 ± 0.3	91.26 ± 0.4	
	nginx	wget	90.02 ± 0.2	95.18 ± 5.2	130.63 ± 3.6	130.74 ± 2.7	
		cURL	90.02 ± 0.2	91.33 ± 1.2	105.87 ± 3.0	107.25 ± 3.2	
		aria2c	90.24 ± 0.3	90.63 ± 0.3	91.25 ± 0.4	91.09 ± 0.3	
	vsftpd	cURL	90.32 ± 0.2	96.69 ± 2.8	125.00 ± 3.4	126.80 ± 4.2	
		LFTP	90.33 ± 0.5	90.75 ± 0.4	92.22 ± 2.1	93.10 ± 2.1	
	ProFTPD	cURL	90.43 ± 0.5	91.40 ± 1.5	105.93 ± 2.4	107.07 ± 2.8	
		LFTP	90.57 ± 0.3	91.05 ± 0.3	92.37 ± 2.2	93.35 ± 1.9	
	OpenSSH	SCP	90.32 ± 0.5	92.62 ± 2.4	127.96 ± 5.1	133.71 ± 5.5	
	100Mbps	Apache2	wget	45.23 ± 0.3	89.18 ± 2.1	131.48 ± 5.1	132.94 ± 5.5
			cURL	45.18 ± 0.1	77.09 ± 2.5	107.77 ± 3.4	109.07 ± 2.8
			aria2c	45.43 ± 0.3	53.07 ± 2.1	72.72 ± 4.3	74.22 ± 4.4
		nginx	wget	45.18 ± 0.2	90.43 ± 3.7	131.26 ± 3.9	131.19 ± 4.6
cURL			45.18 ± 0.3	78.33 ± 2.5	106.36 ± 3.0	108.07 ± 3.3	
aria2c			45.37 ± 0.2	54.46 ± 2.3	73.00 ± 3.1	74.14 ± 3.6	
vsftpd		cURL	45.31 ± 0.2	97.29 ± 3.3	126.24 ± 5.6	128.93 ± 3.9	
		LFTP	45.46 ± 0.1	64.33 ± 6.0	89.36 ± 3.4	91.77 ± 3.0	
ProFTPD		cURL	45.64 ± 0.2	77.18 ± 2.4	107.28 ± 3.3	109.01 ± 3.4	
		LFTP	45.73 ± 0.4	65.15 ± 4.0	88.96 ± 2.6	92.68 ± 2.5	
OpenSSH		SCP	45.45 ± 0.1	85.14 ± 2.9	128.93 ± 7.0	134.77 ± 5.8	
300Mbps		Apache2	wget	13.51 ± 3.4	86.18 ± 5.6	131.71 ± 4.7	129.01 ± 5.9
			cURL	13.61 ± 2.9	74.41 ± 5.7	106.42 ± 3.6	105.22 ± 3.7
			aria2c	10.85 ± 2.4	55.01 ± 3.9	72.27 ± 3.1	72.92 ± 3.8
		nginx	wget	13.81 ± 3.1	86.95 ± 5.0	131.29 ± 5.2	131.16 ± 5.7
	cURL		13.32 ± 3.2	75.75 ± 6.0	105.81 ± 4.5	106.41 ± 3.7	
	aria2c		10.75 ± 2.6	54.88 ± 3.3	72.25 ± 3.5	74.21 ± 3.5	
	vsftpd	cURL	31.91 ± 3.6	95.35 ± 4.9	124.57 ± 5.4	126.78 ± 5.6	
		LFTP	13.27 ± 2.4	63.37 ± 5.2	89.67 ± 3.0	87.98 ± 3.3	
	ProFTPD	cURL	16.53 ± 3.1	77.00 ± 5.7	108.50 ± 3.5	106.49 ± 4.2	
		LFTP	13.05 ± 2.5	62.57 ± 4.2	88.08 ± 3.3	89.18 ± 3.4	
	OpenSSH	SCP	11.90 ± 2.1	81.36 ± 4.5	128.15 ± 6.1	130.88 ± 4.2	

5.2.5 Transferring Files with a Bit Rate of 10Mbps

Transferring files using a bit rate as low as 10Mbps confirmed the above observations for a bit rate of 50Mbps, i.e. that the usage control infrastructure's overhead decreases with lower bit rates. The performance measurement results for this set of experiments are provided in Table 5.6 and Figure 5.5. Notably, these results do not include the case of transferring files of size 1KB. As discussed in Section 5.2.1, the network bit rate did not have any significant influence when transferring such small files. This was also true for a 10Mbps bit rate, which is why this case is covered in Section 5.2.1.

Considering the transfer of larger files of size 1MB and comparing the evaluation results with the results for a bit rate of 50Mbps in Section 5.2.2, it turns out the *native* file transfer times (■) increase in correspondence with the slower bit rate. The additional overheads imposed by the usage control infrastructure turn out to be slightly smaller for a bit rate of 10Mbps: For all client/server combinations but OpenSSH/SCP, signaling of the system calls to the Controller (■) averages at $\sim 160ms \pm 32ms$ for 10Mbps and at $\sim 167ms \pm 24ms$ for 50Mbps; the additional overhead imposed by local data flow tracking (■) averages at $\sim 65ms \pm 52ms$ for 10Mbps and at $\sim 104ms \pm 76ms$ for 50Mbps; in terms of cross-system data flow tracking and policy propagation (■) the additional overhead is similarly negligible in both cases.

Evaluation of the measurement results for files of size 128MB and 512MB reveals that the overall overhead imposed by the usage control infrastructure (■) is quite small for low network bit rates. For a 128MB file the absolute overall overhead is $\sim 3.2s$ for OpenSSH/SCP and $\sim 1.3s$ for all other client/server combinations. Considering the overall file transfer time of $\sim 112s$, the relative overhead imposed by the usage control infrastructure is as low as 1% to 3%. For a file of size 512MB the overall absolute overhead averages $\sim 6.8s$ for OpenSSH/SCP and at $4.3s \pm 2.5s$ for all other client/server combinations. Again, considering the overall file transfer time of $\sim 450s$, the relative overall overhead ranges between 0.4% and 1.5%.

5.2.6 Summary

Summarizing the results presented in the previous sections, the relative overhead imposed by the usage control infrastructure when transferring files between systems ranges between 0.4% (512MB/nginx/aria2c/10Mbps) and 1000% (512MB/OpenSSH/SCP/300Mbps). Since this range is enormous, it is useful to further discuss and differentiate those results, since the overheads depend on many different parameters as follows.

File size. Naturally, the size of the file being transferred has a significant impact onto the file transfer times. Due to the expensive startup phase of all client/server applications used for the experiments, the usage controlled transfer of smaller files (1KB, 1MB) involved an overhead of at least 45% for decent bit rates of at least 50Mbps. This was different for larger files (128MB, 512MB), in which case the minimal relative

Figure 5.5: Transfer times for a bit rate of 10Mbps.



Table 5.6: Time and standard deviation to transfer files at a bit rate of 10Mbps.

File size	Server	Client	native (■)	signal (■)	local (■)	cross (■)
1MB	Apache2	wget	894 ± 1	1026 ± 11	1071 ± 13	1067 ± 9
		cURL	901 ± 1	1043 ± 9	1076 ± 13	1082 ± 19
		aria2c	1022 ± 6	1192 ± 16	1228 ± 10	1264 ± 20
	nginx	wget	895 ± 1	1029 ± 8	1062 ± 9	1071 ± 7
		cURL	904 ± 1	1040 ± 10	1081 ± 8	1099 ± 8
		aria2c	974 ± 3	1140 ± 12	1182 ± 18	1204 ± 12
	vsftpd	cURL	1100 ± 1	1267 ± 37	1384 ± 25	1378 ± 24
		LFTP	1066 ± 8	1252 ± 16	1265 ± 16	1305 ± 7
	ProFTPD	cURL	1337 ± 2	1465 ± 14	1572 ± 42	1603 ± 31
		LFTP	1325 ± 10	1517 ± 16	1556 ± 14	1560 ± 19
	OpenSSH	SCP	1243 ± 9	2729 ± 141	3221 ± 104	3527 ± 412
	128MB	Apache2	wget	112.49 ± 0.00	112.67 ± 1.12	112.69 ± 0.02
cURL			112.49 ± 0.00	112.67 ± 0.02	112.69 ± 0.17	113.73 ± 0.49
aria2c			112.77 ± 0.00	112.95 ± 0.43	113.06 ± 4.94	114.04 ± 0.03
nginx		wget	112.49 ± 0.00	112.63 ± 0.03	112.66 ± 0.27	113.77 ± 0.27
		cURL	112.50 ± 0.35	112.66 ± 0.02	112.69 ± 0.04	113.77 ± 1.42
		aria2c	112.73 ± 0.07	112.91 ± 0.03	112.98 ± 0.83	113.98 ± 0.12
vsftpd		cURL	114.60 ± 1.02	112.88 ± 0.02	113.04 ± 0.02	114.10 ± 0.05
		LFTP	112.84 ± 0.70	113.02 ± 0.02	113.06 ± 0.02	114.09 ± 0.77
ProFTPD		cURL	112.95 ± 2.21	113.12 ± 0.03	113.24 ± 0.03	114.22 ± 0.05
		LFTP	113.17 ± 0.65	113.34 ± 0.41	113.33 ± 0.01	114.40 ± 0.08
OpenSSH		SCP	112.76 ± 0.01	114.19 ± 0.09	114.79 ± 0.04	116.00 ± 0.13
512MB		Apache2	wget	452.31 ± 1.3	450.13 ± 1.0	450.32 ± 0.9
	cURL		449.91 ± 0.3	450.77 ± 0.9	450.65 ± 1.2	454.54 ± 0.2
	aria2c		450.69 ± 0.7	451.26 ± 0.7	451.21 ± 0.3	455.24 ± 0.5
	nginx	wget	449.92 ± 0.8	450.14 ± 0.8	450.40 ± 0.2	454.78 ± 1.1
		cURL	452.09 ± 4.3	450.47 ± 0.3	451.20 ± 0.4	454.34 ± 1.2
		aria2c	453.58 ± 1.5	451.04 ± 0.4	451.13 ± 0.1	455.39 ± 0.6
	vsftpd	cURL	450.73 ± 1.6	451.66 ± 0.6	451.13 ± 0.8	455.49 ± 0.5
		LFTP	451.10 ± 1.5	451.09 ± 0.0	452.11 ± 1.5	455.18 ± 0.2
	ProFTPD	cURL	450.45 ± 1.6	450.78 ± 0.3	451.07 ± 1.2	455.05 ± 0.8
		LFTP	451.14 ± 2.3	451.79 ± 2.6	451.71 ± 0.4	455.49 ± 0.1
	OpenSSH	SCP	449.96 ± 0.2	451.80 ± 0.1	452.78 ± 1.2	456.75 ± 0.1

overhead dropped to 1%. Both for smaller and larger files, however, the relative overhead might go up to 1000%—depending on the other parameters discussed in the following.

Application. In particular when transferring large files the choice of the application can heavily influence file transfer times. E.g., transferring a 512MB file using a bit rate of 300Mbps takes 72.92s when using Apache2/aria2c, but 132.94s when using Apache2/wget. For transferring small files, OpenSSH/SCP turned out to be extremely expensive, taking 2.5ms to transfer a 1KB file which only takes $0.5ms \pm 0.2ms$ for all other client/server combinations. As the experiments show, there is a clear correlation between those file transfer times and the amount of system calls being issued by the corresponding applications. As a consequence, a redesign of the applications to use less *read* and *write* system calls—which make up the majority of system calls when transferring larger files—could significantly improve the above measurement results. E.g., since version 2.2 the Linux kernel offers the system call *sendfile*, which is one single system call that allows to “transfer data between file descriptors [which] is more efficient than the combination of read and write” [122].

Bit rate. The overheads imposed by the usage control infrastructure are relatively low for smaller network bit rates. As discussed, this is because in this case most of the infrastructure’s tasks can be performed while the client and/or server process is waiting for the TCP communication channel to become ready to read or write. Consequently, for a bit rate of 10Mbps the experiments showed that the overall relative overhead can be as low as 0.4%. For a bit rate of 50Mbps the relative overhead was as low as 1% for some client/server/file size combinations. On the other hand, a network bit rate as high as 300Mbps results in enormous overheads of up to 1000%, effectively degrading the network’s throughput to 10% of its nominal value.

Considering this latter case, it is particularly beneficial to analyze which parts of the usage control infrastructure account for which parts of the overall overhead. In fact, for a bit rate of 300Mbps and a file size of at least 1MB signaling of the system calls from the PEP to the Controller (■) accounts for 57% to 89% of the overall imposed overhead (cf. Figures 5.2c, 5.3c and 5.4a and the corresponding tables). In comparison, local data flow tracking (■) accounts for ‘only’ 11% to 39% percent of the overall overhead. Cross-system data flow tracking and policy propagation (■) never accounts for more than 7% of the overall overhead and is usually rather around 2% to 3%. Signaling of the system calls to the Controller is so expensive is because the Controller, including PDP, PIP, and PMP, have been implemented in Java (cf. Section 4.1), while the PEP being used for these experiments was implemented in C (cf. Section 4.1.4). Hence, signaling of the system calls demands expensive inter-process communication which was implemented using the Apache Thrift RPC protocol. It stands to reason that much of the above signaling overhead could be avoided by tightly

integrating the usage control infrastructure with the PEP, e.g. by implementing the usage control infrastructure as a shared library and linking it to the PEP directly.

Which of the measured overheads are actually acceptable clearly depends on the given application scenario. E.g., if in an application scenario small files are *occasionally* transferred across systems, than a file transfer time of 171ms instead of 17ms (cf. Section 5.2.1, 1KB/nginx/wget) might be totally acceptable. On the other hand, if many such small files (e.g., 1000) would need to be transferred one after another in batch mode, than a transfer time of 171s instead of 17s is likely not acceptable.

For low bit rates (cf. Section 5.2.5) and larger files the overall overhead is around 1% for most cases. This seems to be absolutely acceptable for most scenarios when considering the overall transfer times of 130s for 128MB and 450s for 512MB. Given a bit rate of 300Mbps, however, an overall file transfer time of 130s instead of the native 13.5s (512MB/Apache2/wget) is likely to be considered unacceptable.

5.3 Distributed Policy Decisions

Contents of this section have been published in [93].

The goal of this section is to understand which communication and performance overheads are introduced when enforcing global policies in a decentralized manner (**RQ2**) as conceptualized in Section 3.3 and implemented in Section 4.3 (requirement **R5**). Further, the goal is to understand how these overheads compare to a centralized infrastructure. For this, several case studies along the running example from Section 1.5 have been performed: Several data usage policies were enforced by multiple systems simultaneously and the overheads in different situations were measured. After detailing the system setup, identifying relevant parameters, and describing how the experiments were executed, the subsequent sections present the actual evaluation results.

System Setup. The experiments presented in this section were run on twelve machines, y_0, \dots, y_{11} , which were configured with a 4x2.6GHz CPU. y_0 was configured with 16GB RAM, while y_1, \dots, y_{11} were configured with 4GB RAM each. The software being used was equivalent to what has been described in Section 5.2. For the central system setup, y_0 was hosting the central data usage control instance, consisting of one fully functional Controller as described in Section 4.1.3. This central Controller was responsible for policy evaluation and data flow tracking for several PEPs being run on systems y_1, \dots, y_{11} . In this case, Cassandra was *not* executed. For the decentral setup, systems y_1, \dots, y_{11} all run exactly one fully functional Controller; y_0 was not used. Again, all cross-system communication was encrypted using TLS, and Cassandra used a consistency level of *Quorum*.

Parameters. While running and evaluating the experiments, the following parameters turned out to influence the evaluation results when enforcing global data usage policies:

- (i) The *policy* being enforced. In the following experiments, policies 1, 2 and 3 from the running example were enforced. Note that for the performance evaluation (Sections 5.3.4 to 5.3.6) it was of importance whether the policy was evaluated upon actual and/or desired events. In order to get both best and worst case evaluation results, policies were only evaluated upon actual events. Due to the performed (distributed) policy evaluation, the evaluation times of actual events yielded worst case results. Since desired events did not trigger any policy evaluation, their evaluation times yielded best case results.
- (ii) The *total* number of systems being usage controlled ($t \in \mathbb{N}$), which was in between 3 and 11, $3 \leq t \leq 11$.
- (iii) The number of systems actually *enforcing* the policy ($e \in \mathbb{N}$). Given a number of t usage controlled systems and a policy p , only some of those usage controlled systems might in fact enforce policy p . E.g., if a system is not aware of any data being addressed by policy p , then there is no need for this system to enforce policy p . Hence, $e \leq t$.
- (iv) The *global event frequency* in events per second (Ev/s) ($f \in \mathbb{R}^{\geq 0}$), i.e. the amount of events happening within a certain amount of time within the entire distributed system. For the following experiments, event frequencies up to 167 Ev/s were used.
- (v) The percentage of events *relevant* for data flow tracking and/or policy evaluation (r), since usually not all events being observed by PEPs are of relevance for the PDP and/or PIP. Hence, $0\% \leq r \leq 100\%$.

Although those parameters and the range of their potential values impose a huge complexity on the performed experiments, the presented evaluation results provide first insights of the influence of those parameters on any overheads.

Experiment Execution. For each measurement all of the above parameters were fixed and an event trace was randomly generated. Each generated event trace matched the given global event frequency f and had a nominal total execution time of 30 seconds (e.g., given $f = 100\text{Ev/s}$ an event trace consisting of 3000 events was generated). Thereby, each event was randomly assigned to one of the t participating usage controlled systems. Then, the event trace was executed, whereby the policy was evaluated upon every trigger event as well as for a timestep interval of one second. In correspondence with Section 4.3.1, time-based policy evaluation happened consistently across all PDPs. After each run, the entire infrastructure was reset. Note, that the PEPs intercepted the system events both before and after their execution, resulting in a *desired event* and an *actual event* being sent to the PDP. Communication overhead was measured by dumping the network communication using `tcpdump`; times were measured using the C++ `chrono` `datetime` library.

Note that due to the vast input space it was not feasible to have multiple experiment executions for every possible combination of parameters as explained above. Never-

Figure 5.6: Recap of policy 1 and the corresponding ECA rules.

Policy 1: ‘Exactly one contract offer must be sent to the customer not later than 30 days after a request for a contract offer has been received.’	
	Event: $\langle any \rangle$
(a)	Condition: $((requestOffer, \{(obj, d)\}) \textit{before} 30)$ $\textit{and repmax}(30, 0, (sendOffer, \{(obj, d)\}))$
	Action: $(notifyManager, \{(obj, d)\})$
	Event: $(sendOffer, \{(obj, d)\})$
(b)	Condition: $repmax(30, 0, (requestOffer, \{(obj, d)\}))$ $\textit{or repmin}(30, 1, (sendOffer, \{(obj, d)\}))$
	Action: $inhibit$

theless, the regularity of the obtained evaluation results (cf. the following sections) indicates that the randomness in the evaluation process does not significantly influence the results. This claim is further confirmed by the twentyfold execution of selected parameter combinations and the corresponding minimal difference in the obtained results.

Results in a Nutshell. The results show that neither the decentralized nor the centralized approach performs inexorably better than the other. In many cases, however, the decentralized approach outperformed the centralized approach. The following sections present the obtained evaluation results in detail. Sections 5.3.1 to 5.3.3 deal with the measured communication overheads, while Sections 5.3.4 to 5.3.6 describe the imposed performance overheads. Section 5.3.7 summarizes the results.

5.3.1 Communication Overhead: Policy 1

Figures 5.7 to 5.9 show the global communication overhead when enforcing ECA rules 1a and 1b, which are recapped in Figure 5.6. For Figures 5.7 and 5.8 a total number of three usage controlled systems were monitored and all of them actually enforced ECA rule 1a and 1b, respectively ($e = 3 = t$). For Figure 5.9 a total number of seven usage controlled systems were monitored and three of them actually enforced ECA rule 1a ($e = 3 \leq 7 = t$).

For Figures 5.7a, 5.8a and 5.9a the depicted data points were obtained by fixing several percentages of relevant events ($r \in \{0\%, 10\%, 25\%, 50\%, 75\%, 100\%\}$) and by experimenting with the global event frequency $0.3\text{Ev/s} \leq f \leq 130\text{Ev/s}$. The x-axis shows the global event frequency f in Ev/s, while the y-axis shows the *global* amount of exchanged Bytes per second. Trends are visualized using linear regression. The same applies to Figures 5.11a, 5.13a and 5.14a within the upcoming sections.

For these and all future experiments the results produced by the central system setup (\blacklozenge) were of little surprise: On average, each event being observed by a PEP

caused 1170 Bytes to be exchanged between the PEP and the central PDP. For the central system setup the percentage of relevant events (r) did not have any influence on the communication overhead. This is of no surprise when recapping (cf. Section 2.2.2) that the PEP is stateless and that *every* event must be signaled to the central PDP.

Running the distributed infrastructure, the first observation is that Cassandra causes some base ‘noise’ in order to keep the distributed database in a consistent state. This implies that the centralized approach performs inexorably better for very low event frequencies as can be seen in Figures 5.7a, 5.8a and 5.9a for $f \lesssim 12\text{Ev/s}$. However, depending on the ECA rule being enforced, the event frequency f , the percentage of relevant events r , as well as the amount of involved systems (t, e), the decentralized approach is capable of outperforming the centralized approach.

While in general event traces with a low percentage of relevant events perform particularly well (Figures 5.7a, 5.8a and 5.9a,  (10% relevant events),  (25%)), some remarkable exceptions can be observed. First of all, aforementioned traces perform good for two reasons: (i) policies can in many cases be conclusively evaluated locally, avoiding costly lookups within the distributed database; (ii) a low percentage of relevant events implies that a small amount of state changes (cf. Section 4.3) must be notified to other PDPs and thus written to the distributed database. Secondly, traces with a high percentage of relevant events perform badly ( (75%),  (100%)). While in this case the PDPs can almost always decide locally, a high amount of state changes must be notified to other PDPs. Thus, the lion’s share of the communication overhead is due to events and state changes being written to the database. Thirdly, traces with 0% of relevant events () may perform particularly good (cf. Figure 5.8a) or particularly bad (cf. Figures 5.7a and 5.9a). The reason for this vast difference is that for ECA rule 1a the corresponding ECA rule’s trigger event is $\langle any \rangle$, while for ECA rule 1b the trigger event is one concrete event, namely $(sendOffer, \{(obj, d)\})$. Consequently, ECA rule 1a is evaluated upon *every* observed event and for each the ECA rule’s condition must be evaluated, resulting in expensive lookups within the database (Figures 5.7a and 5.9a, ). Since the percentage of relevant events is 0, the trigger event of ECA rule 1b never happens and thus the ECA rule’s condition is only evaluated once per timestep per PDP; consequently, for the most part only Cassandra’s base noise can be observed (Figure 5.8a, .

Comparing Figures 5.7a and 5.8a, it turns out that ECA rule 1a can generally be evaluated more efficiently than ECA rule 1b. The main reason is that the evaluation of operator *before* in ECA rule 1a necessitates at most one database lookup per PDP *per timestep*, while in the worst case each *repmIn* operator, which occurs twice in the condition of ECA rule 1b, necessitates one lookup upon *every* event.

Comparing Figures 5.7a and 5.9a, it turns out that the global communication overhead decreases if the total amount of usage controlled systems (t) increases and if the total amount of systems actually enforcing ECA rule 1a (e) remains constant (Figure 5.7a: $e = 3 = t$; Figure 5.9a: $e = 3 \leq 7 = t$). The reason is that if $e = 3 = t$,

Figure 5.7: Communication overhead when enforcing ECA rule 1a on three systems.

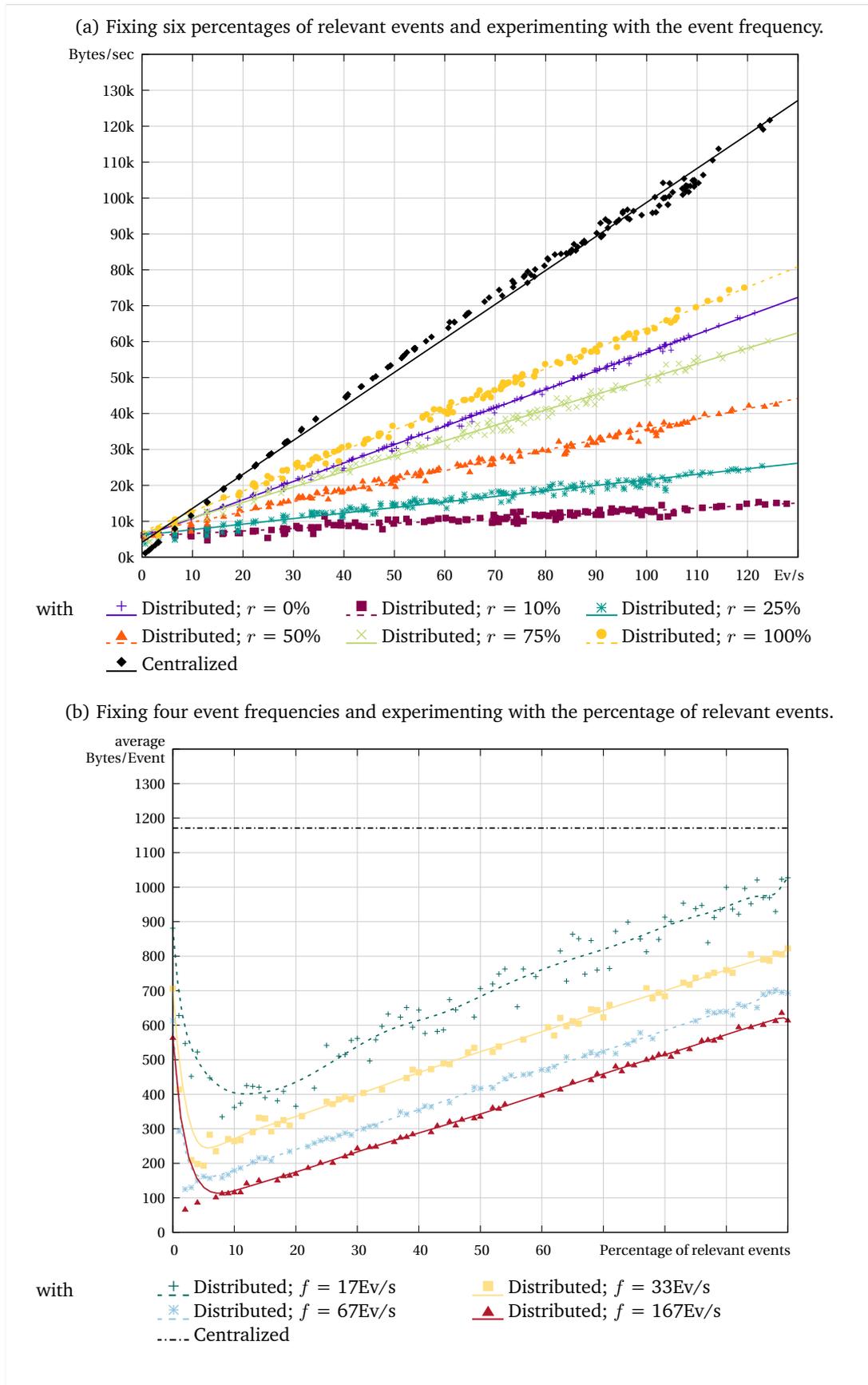


Figure 5.8: Communication overhead when enforcing ECA rule 1b on three systems.

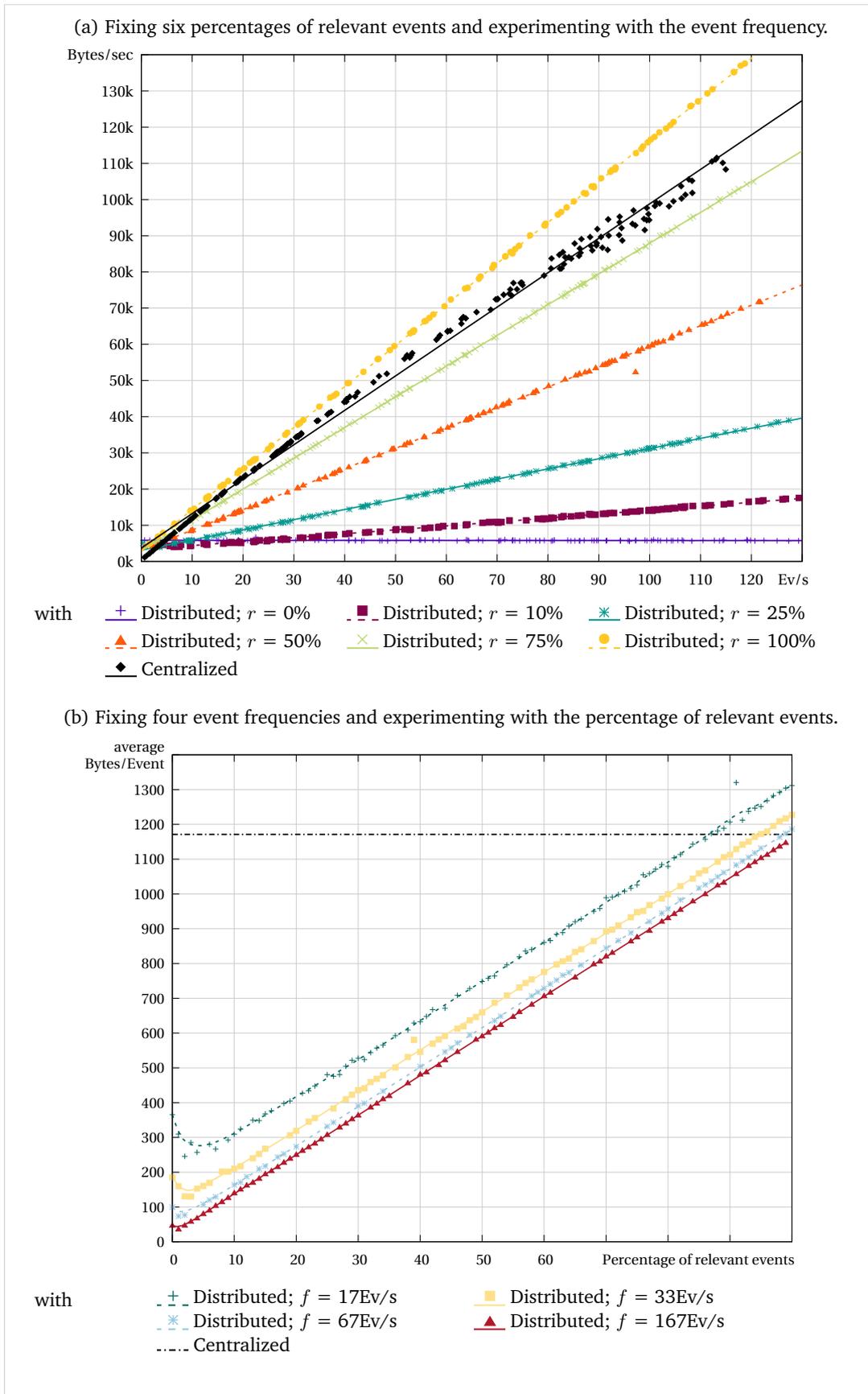
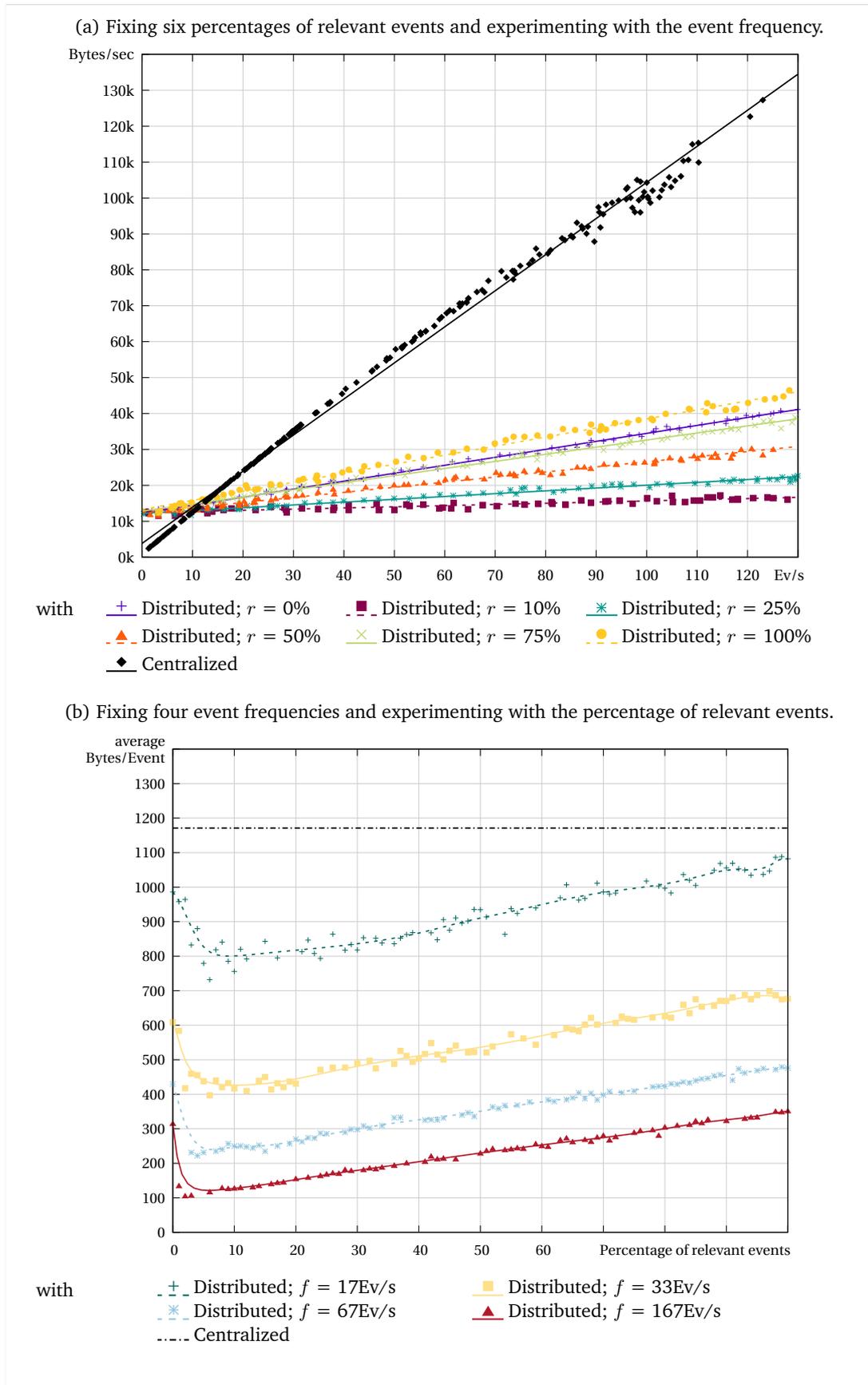


Figure 5.9: Communication overhead enforcing ECA rule 1a on three of seven systems.



then every single observed event (i.e. $e/t = 3/3 = 1 = 100\%$) causes the evaluation of ECA rule 1a's condition and thus potentially some reading or writing on the database. However, if only three out of the seven usage controlled systems do in fact enforce ECA rule 1a ($e = 3 \leq 7 = t$), then only 43% of all observed events (i.e. $e/t = 3/7 \approx 0.43$) cause the evaluation of ECA rule 1a's condition and thus potentially some reading or writing on the database.

In Figures 5.7a, 5.8a and 5.9a several percentages for relevant events were fixed, i.e. $r \in \{0\%, 10\%, 25\%, 50\%, 75\%, 100\%\}$. As described, these event frequencies do have a major influence on the measured communication overheads. To understand this influence better, Figures 5.7b, 5.8b and 5.9b fix several event frequencies (i.e. $f \in \{17\text{Ev/s}, 33\text{Ev/s}, 67\text{Ev/s}, 167\text{Ev/s}\}$) and show how the percentage of relevant events r ($0\% \leq r \leq 100\%$, x-axis) influences the total amount of Bytes being exchanged between all involved systems. To make those numbers comparable, the measurements are normalized by dividing the total amount of exchanged Bytes by the number of observed events, resulting in a number representing the *average* amount of Bytes exchanged *per event* (y-axis). Again, for the centralized approach (.....) the communication overhead is constant (1170 Bytes per event) and the percentage of relevant events does not influence the amount of bytes being exchanged. Note that this description also applies to Figures 5.11b, 5.13b and 5.14b.

For all scenarios it turns out that the decentralized approach performs best for high event frequencies (Figures 5.7b, 5.8b and 5.9b, \ast (67Ev/s), \blacktriangle (167Ev/s)) and if the percentage of relevant events is around 3% to 10%. Firstly, this is because higher event frequencies exploit better Cassandra's base noise which keeps the database consistent. Secondly, a low percentage of relevant events results in many situations in which the local PDPs can decide conclusively, while a low amount of state changes must be notified to other PDPs. However, if the amount of relevant events is too low ($r \lesssim 2\%$), then many lookups within the database are required, while the presence of many relevant events results in many writes to the database. Hence, the centralized approach outperforms the decentralized approach if the percentage of relevant events is very low or very high ($r \lesssim 2\% \vee r \gtrsim 85\%$) and if the global event frequency is very low ($\lesssim 11\text{Ev/s}$); concrete values depend on the ECA rule being enforced as well as on the amount of usage controlled systems (t, e).

Recap that the above evaluation results are based on event traces that were randomly generated. In order to show that the introduced randomness does not significantly influence the evaluation results, the experiments were repeated twenty times for selected parameter configurations. The obtained results, i.e. the median amount of bytes exchanged per second as well as the standard deviation, are presented in Tables 5.7 to 5.9 respectively for the enforcement ECA rule 1a within three systems, for the enforcement ECA rule 1b within three systems, as well as for the enforcement of ECA rule 1a within three out of seven systems. The presented numbers are in correspondence with the results depicted in Figures 5.7a, 5.8a and 5.9a.

Table 5.7: Median KB per second and standard deviation for ECA rule 1a on three systems.

Relevant events	Event frequency			
	16Ev/s	32Ev/s	62Ev/s	113Ev/s
0%	13.9 ± 0.47	21.7 ± 0.12	36.6 ± 0.12	59.7 ± 0.21
10%	6.8 ± 0.58	8.0 ± 0.66	10.5 ± 0.43	13.7 ± 0.43
25%	8.6 ± 0.47	11.5 ± 0.36	16.1 ± 0.44	23.1 ± 0.52
50%	11.4 ± 0.27	16.4 ± 0.35	24.2 ± 0.74	37.8 ± 0.72
75%	14.0 ± 0.53	20.7 ± 0.46	33.0 ± 0.55	53.0 ± 1.02
100%	16.3 ± 0.33	25.2 ± 0.70	41.5 ± 0.75	67.4 ± 0.61

Table 5.8: Median KB per second and standard deviation for ECA rule 1b on three systems.

Relevant events	Event frequency			
	16Ev/s	32Ev/s	62Ev/s	113Ev/s
0%	5.6 ± 0.08	5.6 ± 0.10	5.6 ± 0.13	5.7 ± 0.27
10%	4.8 ± 0.18	6.4 ± 0.10	9.7 ± 0.11	15.4 ± 0.07
25%	7.3 ± 0.13	11.7 ± 0.10	19.9 ± 0.10	33.9 ± 0.12
50%	11.7 ± 0.13	20.5 ± 0.09	37.2 ± 0.09	64.2 ± 0.08
75%	16.3 ± 0.10	29.4 ± 0.45	54.4 ± 0.10	93.7 ± 0.21
100%	20.7 ± 0.08	38.2 ± 0.12	71.5 ± 0.14	121.8 ± 0.15

Table 5.9: Median KB per second for ECA rule 1a on three out of seven systems.

Relevant events	Event frequency			
	16Ev/s	32Ev/s	62Ev/s	113Ev/s
0%	15.5 ± 0.23	18.9 ± 0.42	25.4 ± 0.48	35.8 ± 0.46
10%	12.8 ± 0.74	12.9 ± 0.50	14.1 ± 0.39	15.9 ± 0.51
25%	13.0 ± 0.36	14.4 ± 0.39	16.7 ± 0.29	20.4 ± 0.43
50%	14.2 ± 0.35	17.0 ± 0.39	21.0 ± 0.44	27.1 ± 0.60
75%	15.4 ± 0.29	19.1 ± 0.38	24.9 ± 0.64	33.4 ± 0.78
100%	16.9 ± 0.34	21.4 ± 0.51	28.6 ± 0.60	40.4 ± 0.88

Figure 5.10: Recap of policy 2 and the corresponding ECA rule.

Policy 2: <i>‘If the customer declines a contract offer, then all associated and derived data must not be used anymore.’</i>	
Event:	$(use, \{(obj, d)\})$
Condition:	$not(always(not(declineOffer, \{(obj, d)\})))$
Action:	<i>inhibit</i>

5.3.2 Communication Overhead: Policy 2

Figures 5.11a and 5.11b show the communication overheads when enforcing ECA rule 2, which is recapped in Figure 5.10, within a total of seven usage controlled systems ($e = 7 = t$). ECA rule 2 can be enforced particularly efficient using the decentralized infrastructure, in particular if the percentage of relevant events r is greater than 0, $r > 0\%$. The reason for this is the condition of ECA rule 2, which is satisfied if the event $(declineOffer, \{(obj, d)\})$ happened at least once in the past. Once this event is observed for the first time and notified to all other PDPs, no further coordination is ever needed, and only Cassandra’s base noise is observed (Figure 5.11, \blacksquare (10%), \ast (25%), \blacktriangle (50%), \times (75%), \bullet (100%)). If no relevant events are ever happening (Figure 5.11a, $+$ (0%)), the ECA rule’s condition is evaluated once per timestep per PDP. This necessitates a lookup in the database for each PDP in order to determine whether the condition was satisfied at some other PDP. Consequently this case results in a higher communication overhead. Again, the communication overhead caused by the centralized infrastructure is linear in the number of events—irrespective of the percentage of relevant events—and averages at 1170 Bytes/Event.

Similar to the enforcement of policies 1a and 1b in Section 5.3.1, Figure 5.11b shows that a very low percentage of relevant events ($r \lesssim 1\%$) causes relatively high communication overheads. However, different from to the previous policies, Figure 5.11b reveals that for ECA rule 2 the communication overhead for higher percentages of relevant events ($r > 1\%$) is constant for all event frequencies ($+$ (17Ev/s), \blacksquare (3Ev/s), \ast (67Ev/s), \blacktriangle (167Ev/s)).

In summary, when enforcing ECA rule 2 on seven usage controlled systems the decentralized approach outperforms the centralized approach if no relevant events happen ($r = 0\%$) and if the event frequency is greater than $\sim 16\text{Ev/s}$, as well as if relevant events *do* happen ($r \geq 1\%$) and if the event frequency is greater than $\sim 9\text{Ev/s}$.

Also for ECA rule 2 experiments for certain parameter configurations were executed twenty times in order to show the insignificance of the randomness within the evaluation process. The corresponding results, i.e. the median amount of exchanged bytes per second as well as the standard deviation, are presented in Table 5.10. Again, these numbers are in line with the results depicted in Figure 5.11a.

Figure 5.11: Communication overhead when enforcing ECA rule 2 on seven systems.

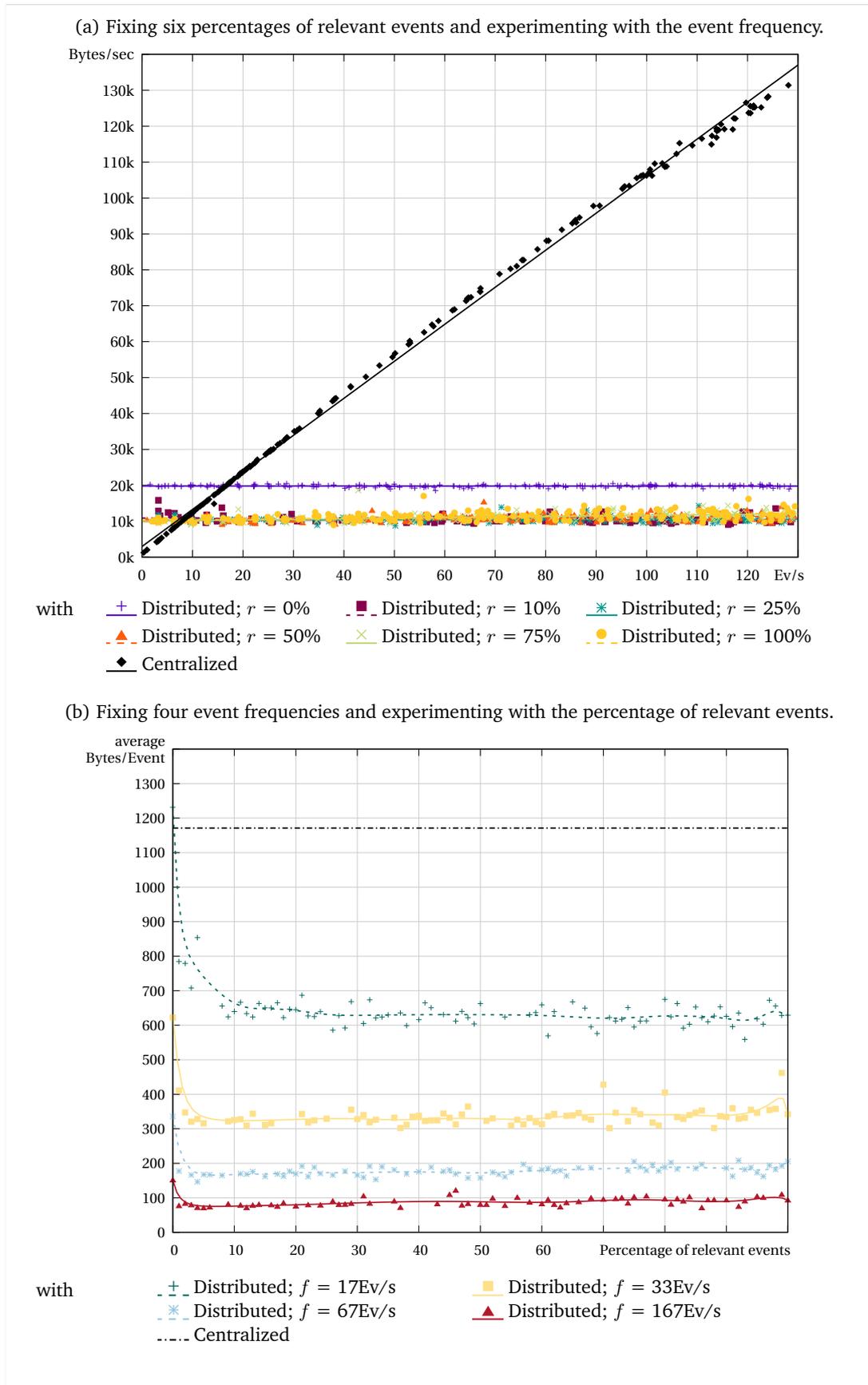


Table 5.10: Median KB per second and standard deviation for ECA rule 2 on seven systems.

Relevant events	Event frequency			
	16Ev/s	32Ev/s	62Ev/s	113Ev/s
0%	19.4 ± 0.25	19.4 ± 0.23	19.4 ± 0.22	19.4 ± 0.59
10%	9.9 ± 0.73	10.0 ± 0.27	9.9 ± 0.90	10.0 ± 0.97
25%	9.9 ± 0.37	10.0 ± 0.44	10.3 ± 0.55	10.5 ± 0.54
50%	9.9 ± 0.41	10.2 ± 0.44	10.7 ± 0.51	11.0 ± 0.90
75%	10.0 ± 0.36	10.4 ± 0.42	10.8 ± 0.71	11.9 ± 1.29
100%	10.3 ± 0.50	10.5 ± 1.02	11.2 ± 0.91	11.5 ± 3.19

Figure 5.12: Recap of policy 3 and the corresponding ECA rule.

Policy 3: ‘Each contract must be reviewed and approved by at least two clerks.’

Event: $(sendOffer, \{(obj, d)\})$

Condition: $\underline{repmax}(30, 1, (review, \{(obj, d)\}))$
 $or \underline{repmax}(30, 1, (approve, \{(obj, d)\}))$

Action: *inhibit*

5.3.3 Communication Overhead: Policy 3

Policy 3 (Figure 5.12) was enforced within two settings: Figures 5.13a and 5.13b show the communication overhead when enforcing ECA rule 3 within a set of eleven systems ($e = 11 = t$), while Figures 5.14a and 5.14b shows the overhead when monitoring a set of eleven systems out of which only five are in fact enforcing ECA rule 3 ($e = 5 < 11 = t$). Note that the scales of the y-axes of Figures 5.13a and 5.14a, as well as those of Figures 5.13b and 5.14b differ.

Technically ECA rule 3 is similar to ECA rule 1b: Both policies feature the same trigger event and their conditions are similar, essentially consisting of a disjunction of two *repmix* operators (recap that the *repmix* operator is a negated *repmix* operator, cf. Section 2.1.3). For this reason the following evaluation results are also compared to the results of enforcing ECA rule 1b within a set of three systems, providing some further insights into how the amount of involved systems influences the communication overhead.

Figure 5.13a reveals that the centralized approach outperforms the decentralized approach for many combinations of event frequency (f) and the percentage of relevant events (r). In particular for high percentages of relevant events ($r \gtrsim 50\%$) the decentralized infrastructure will always perform worse than the centralized approach which can be inferred from the steepness of the corresponding trendlines (cf. \blacklozenge (centralized) and \blacktriangle ($r = 50\%$) in Figure 5.13a). For lower amounts of relevant events (Figure 5.13a, \blackplus (0%), \blacksquare (10%), \blackstar (25%)) and higher event frequencies ($f \gtrsim 65\text{Ev/s}$), however, the decentralized approach performs better than the centralized one. Concrete numbers can be read from Figure 5.13a. These results are

confirmed by Figure 5.13b, which shows that the decentralized approach does under no circumstances perform better than the centralized approach if the event frequency drops below $\sim 67\text{Ev/s}$ (Figure 5.13b, $*$). At the same time, even for high event frequencies (Figure 5.13b, \blacktriangle (167Ev/s)) the percentage of relevant events must be lower than $\sim 25\%$ in order to perform better than the centralized approach.

Comparing these results with the ones obtained by enforcing ECA rule 1b within a total amount of three systems (cf. Section 5.3.1), it turns out that enforcing ECA rule 3 within a total amount of eleven systems is much more expensive. Since the two policies are very similar from a technical perspective as explained above, the root cause for this difference must lie in the amount systems being usage controlled and enforcing the corresponding ECA rule. This difference can be explained when recapping that the underlying Cassandra database is configured to maintain strong data consistency by using a consistency level of *Quorum* for all read and write operations. When enforcing an ECA rule with consistency level *Quorum* within a set of three systems (as was the case for ECA rule 1b in Section 5.3.1), then two Cassandra instances must acknowledge each read and write on the database. In contrast, when enforcing an ECA rule with the same consistency level within a set of eleven systems, then each read and write on the database must be acknowledged by six Cassandra instances. In fact, naively dividing these two numbers ($6/2$) results in a factor of three which explains the difference in the communication overheads in these two scenarios to a very large extent: The communication overhead for enforcing ECA rule 3 within eleven systems (Figure 5.13a) is in fact ~ 3.5 times the overhead for enforcing ECA rule 1b within three systems (Figure 5.8a). The remaining factor of ~ 0.5 can be attributed to the larger management overhead in order to keep a database of eleven rather than three Cassandra instances in a consistent state: the base noise of Cassandra for three systems is $\sim 6\text{KB/sec}$, while it is $\sim 70\text{KB/sec}$ for eleven systems ($+$ in Figures 5.8a and 5.13a).

Figures 5.14a and 5.14b show the results of another experiment in which a total of eleven systems were usage controlled, out of which five were enforcing ECA rule 3. As expected, the communication overhead dropped significantly when compared with the results described above. Comparing the communication overheads in Figures 5.13a and 5.14a, the overhead in the former is 5.8 times the overhead in the latter for percentages of relevant events $r \in \{50\%, 75\%, 100\%\}$. For $r = 25\%$ the difference in the overhead is a factor of 4.2; for $r \in \{0\%, 10\%\}$ the difference in the overheads is a factor of 2.7. There are two parameters that, in combination, explain the difference in the communication overhead in this order of magnitude. First, only five out of eleven systems are enforcing ECA rule 3, which implies that only the events issued by those five systems are of interest for the ECA rule's enforcement. In other words, only $5/11 \approx 45\%$ of all relevant events may cause any communication between the different PDPs. Second, by only coordinating five Cassandra instances, each read and write on the database must only be acknowledged by three Cassandra instances (instead

Figure 5.13: Communication overhead when enforcing ECA rule 3 on eleven systems.

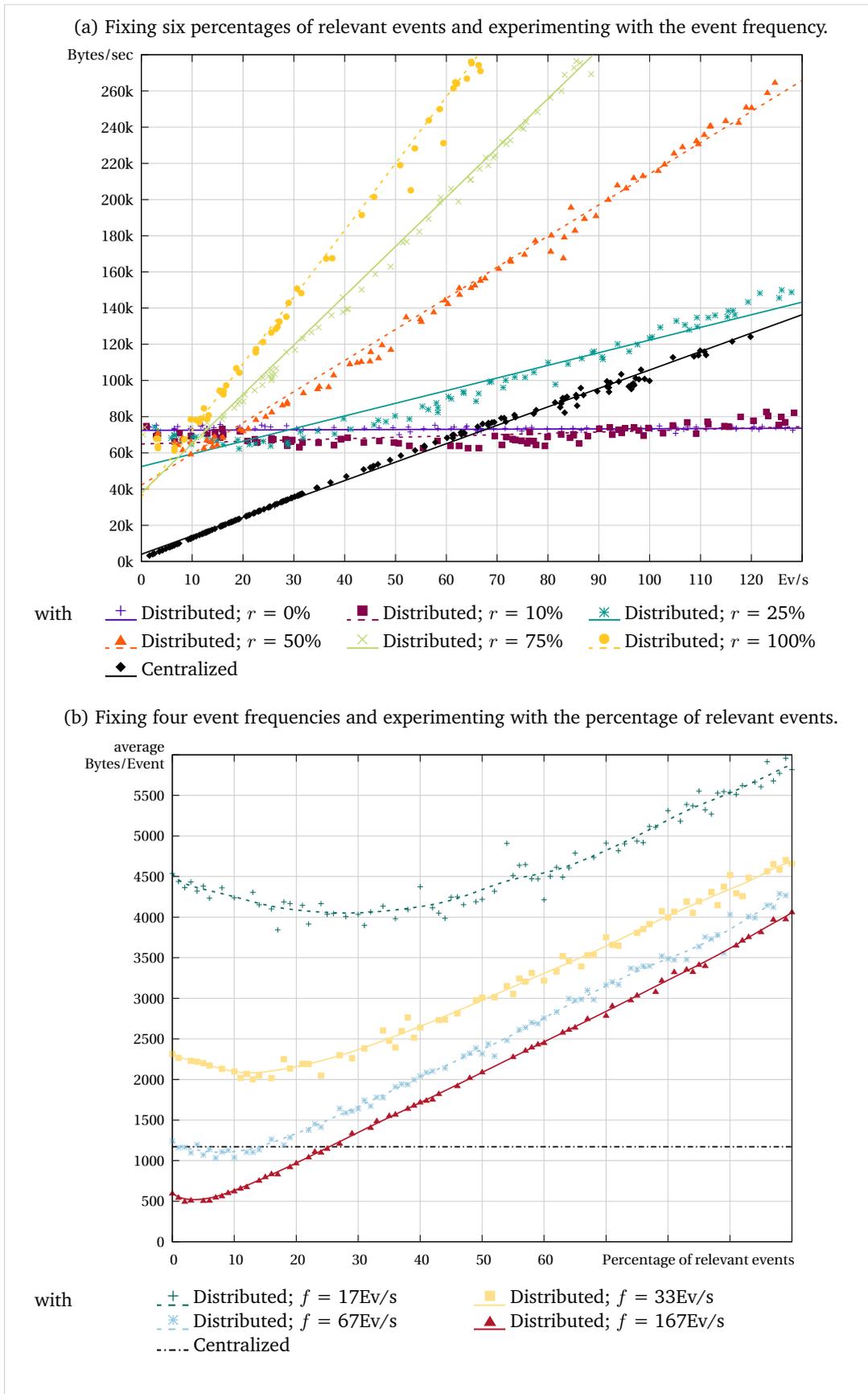


Figure 5.14: Communication overhead enforcing ECA rule 3 on five out of eleven systems.

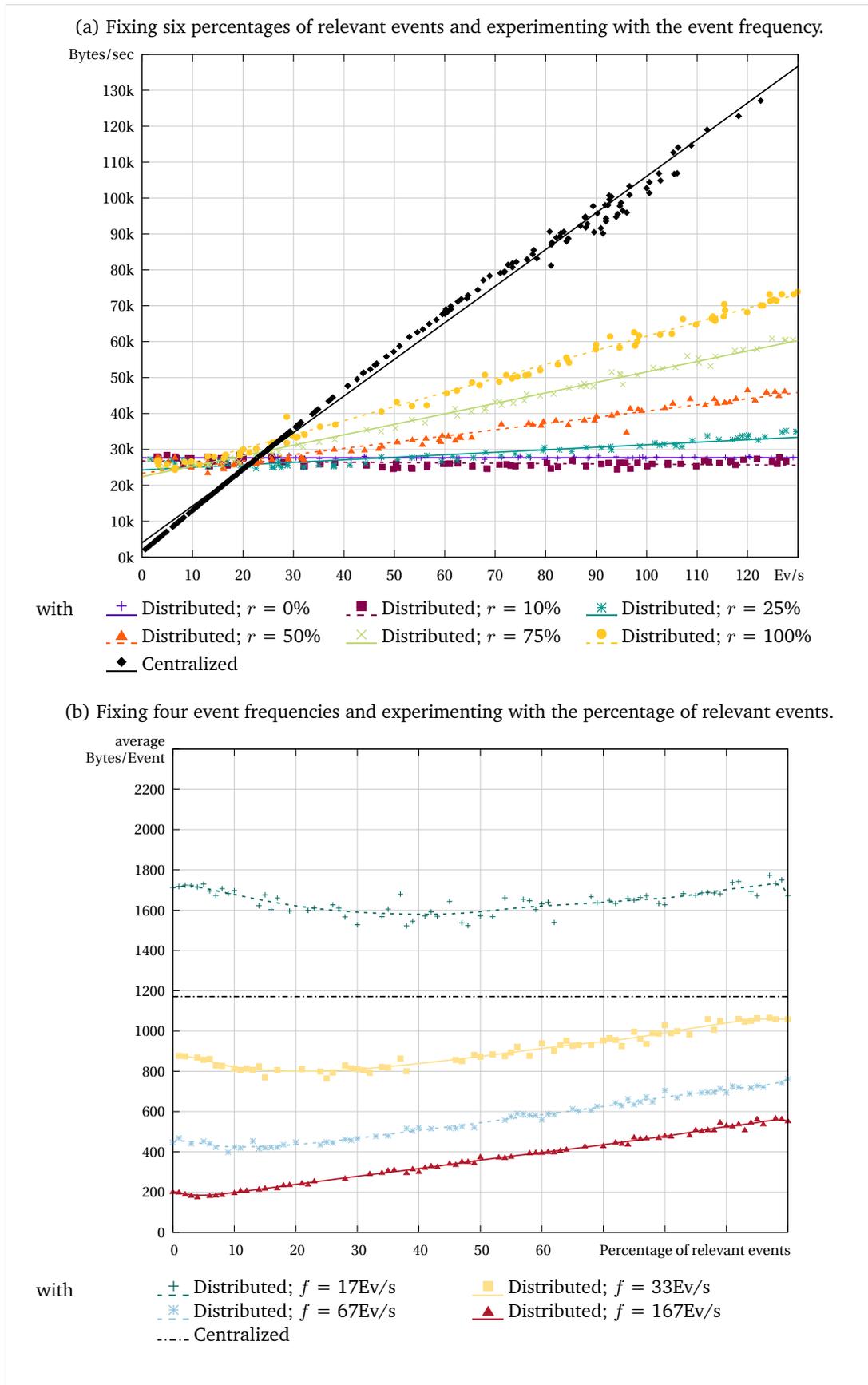


Table 5.11: Median KB per second and standard deviation for ECA rule 3 on eleven systems.

Relevant events	Event frequency			
	16Ev/s	32Ev/s	62Ev/s	113Ev/s
0%	72.8 ± 0.61	72.2 ± 0.59	72.2 ± 0.57	71.8 ± 1.56
10%	68.1 ± 2.07	64.6 ± 3.35	63.2 ± 2.10	74.9 ± 2.82
25%	63.2 ± 2.13	66.8 ± 3.10	88.4 ± 2.20	128.1 ± 2.27
50%	68.5 ± 2.76	90.9 ± 1.73	141.2 ± 3.06	224.6 ± 1.99
75%	78.6 ± 3.35	117.8 ± 3.67	190.6 ± 5.32	338.2 ± 5.33
100%	90.9 ± 2.64	145.3 ± 3.26	245.7 ± 7.80	448.4 ± 7.28

of six as above) when using consistency level *Quorum*. Naively combining these two parameters leads to a factor of $11/5 \cdot 6/3 = 22/5 = 4.4$, which is in fact within the order of magnitude of the above numbers.

Lastly, Figures 5.14a and 5.14b show that for the latter scenario of enforcing ECA rule 3 within five out of a total of eleven systems the decentral approach is again able to outperform the centralized approach as long as the event frequency is at least $\sim 25\text{Ev/s}$.

Again, the experiments were executed twenty times for certain parameter configurations. The corresponding results, i.e. the median amount of exchanged bytes per second as well as the standard deviation, are presented in Table 5.11. The numbers are in line with Figure 5.13a.

Before summarizing these analyses' results in Section 5.3.7, the subsequent sections discuss the imposed performance overheads within the above scenarios.

5.3.4 Performance Overhead: Policy 1

Figures 5.15a, 5.15b, 5.16a and 5.16b show the performance overhead when enforcing ECA rules 1a and 1b, i.e. the average processing time in milliseconds (y-axis) per event for different event frequencies (x-axis). This processing time includes signaling of the event from the PEP to the Controller, evolution of the PIP's data flow state, (distributed) policy evaluation by the PDP, and signaling of the PDP's decision to the PEP. For the distributed infrastructure this overhead also depends on the percentage of relevant events r , which is why several such percentages (i.e. 0% ($\color{purple}{+}$), 25% ($\color{teal}{*}$), 50% ($\color{red}{\triangle}$), 75% ($\color{green}{\times}$), 100% ($\color{orange}{\bullet}$)) were fixed. Note that Figures 5.15a and 5.16a show the corresponding performance overhead for actual events, while Figures 5.15b and 5.16b show the same for desired events. As motivated in the introduction of Section 5.3, the former was expected to yield worst case performance overheads, while the latter was expected to yield best case performance overheads.

Each data point depicted in Figures 5.15a, 5.15b, 5.16a and 5.16b is based on several hundred individual measurements. Recall that each experiment ran for 30 seconds and that event frequencies up to 167Ev/s were used. Then for an event frequency of e.g. 100Ev/s, 3000 events are issued. Because for each event both its

Figure 5.15: Performance overheads for ECA rule 1a on three systems.

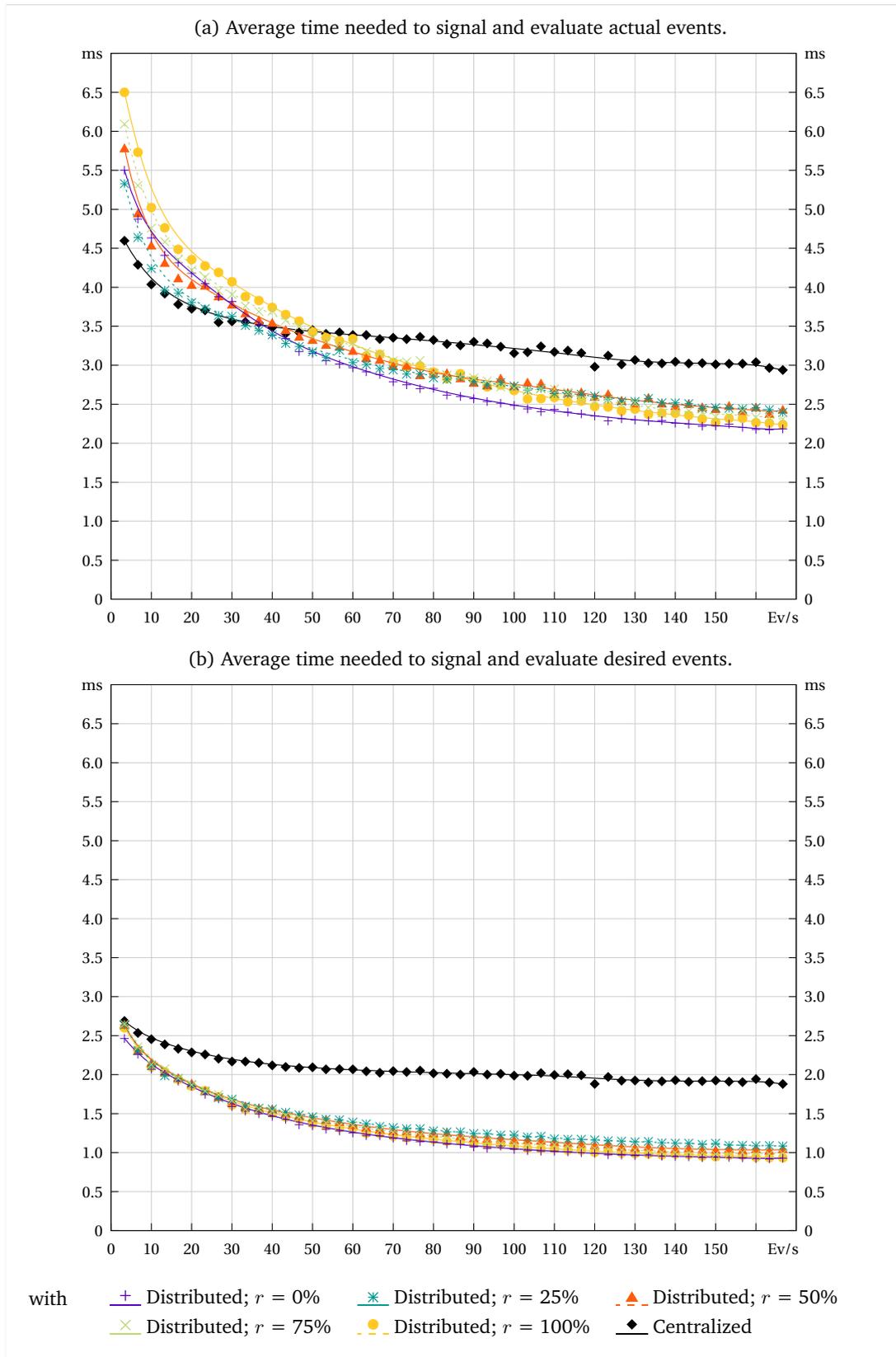
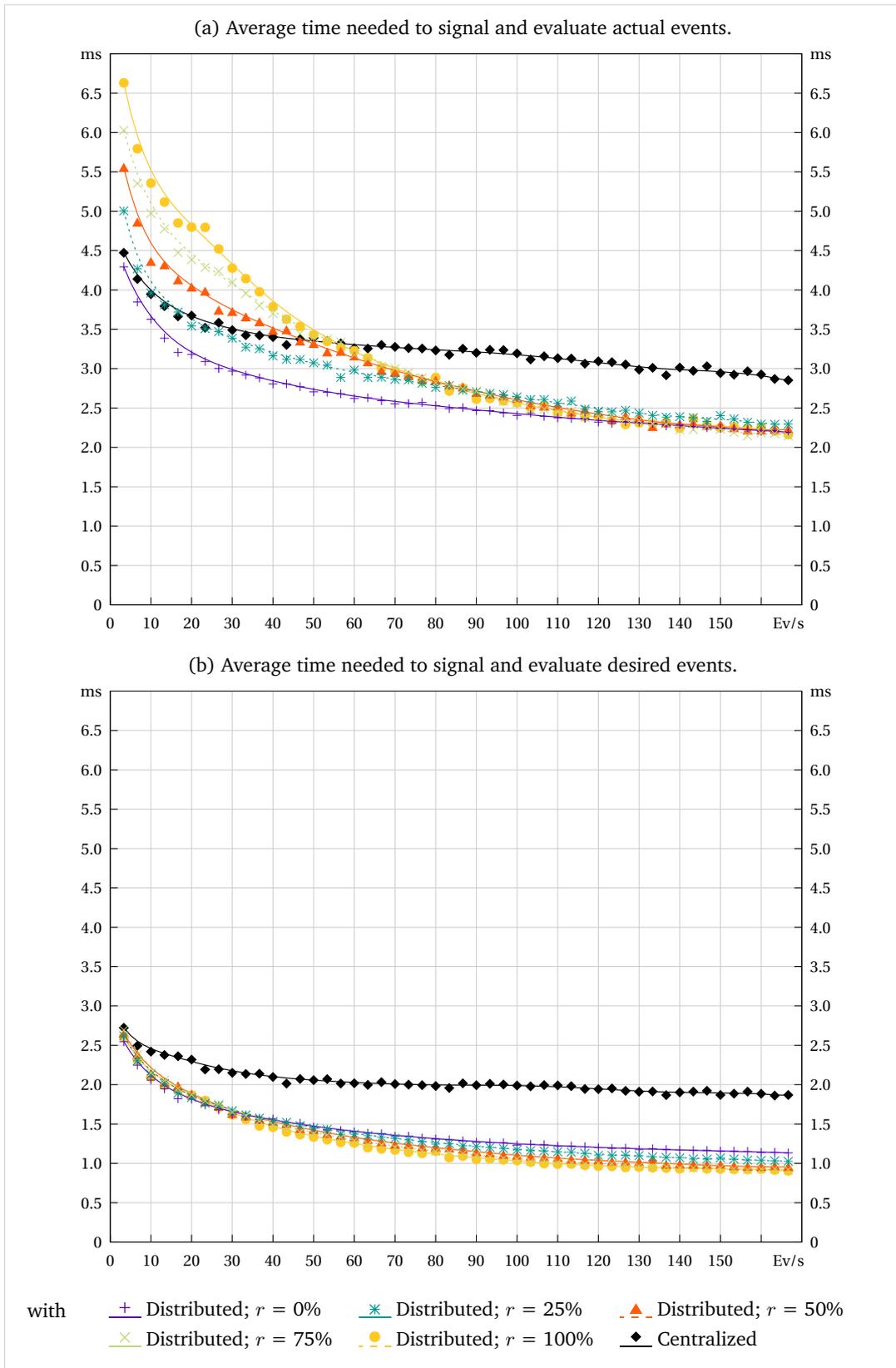


Figure 5.16: Performance overheads for ECA rule 1b on three systems.

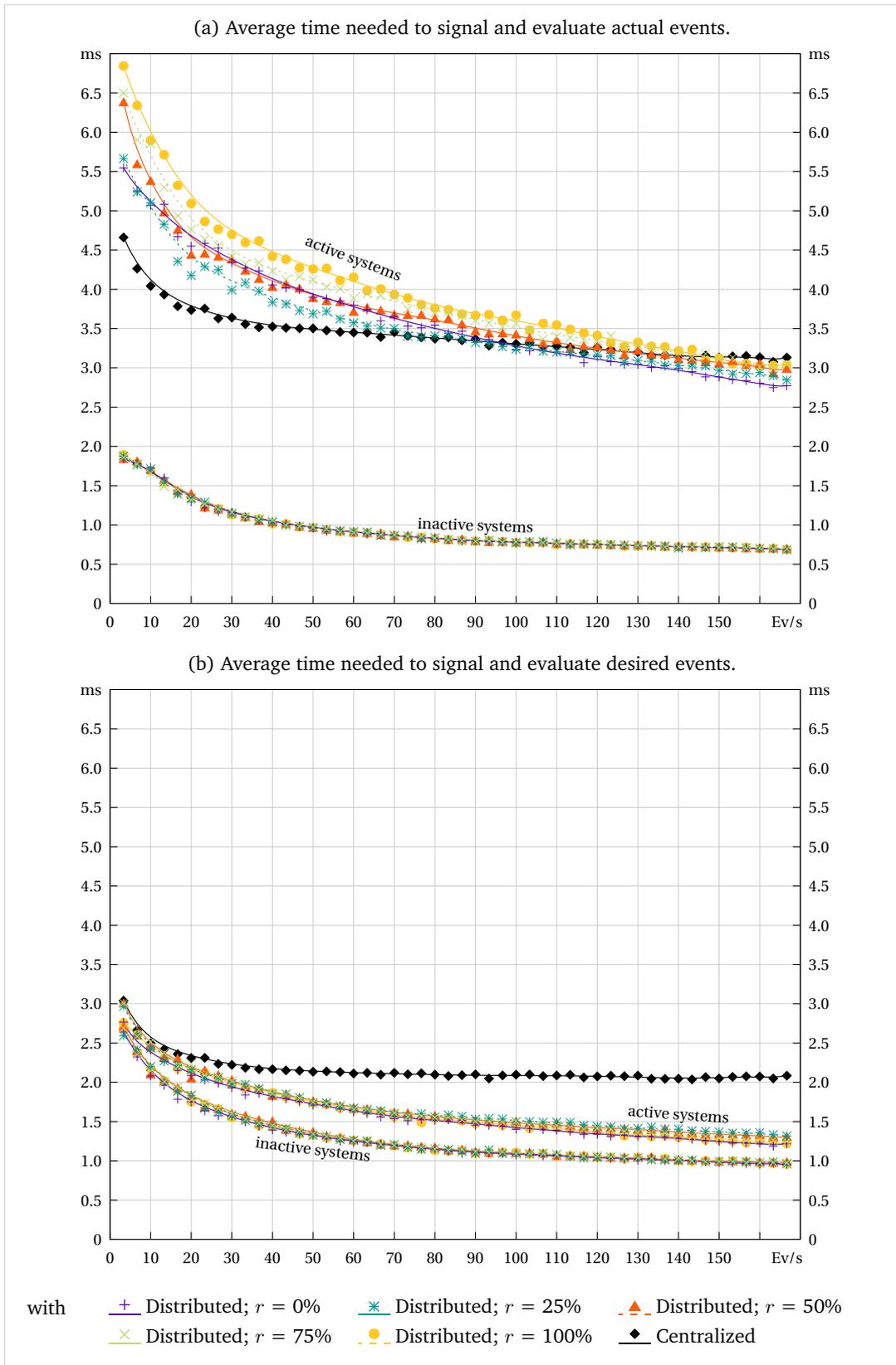


intended and its actual counterpart is intercepted by the PEP and signaled to the Controller (before and after the event's execution, respectively), 6000 measurements could be performed for one experiment run with an event frequency of 100Ev/s. Similarly, for $f = 3.3\text{Ev/s}$, 200 events were measured while for $f = 167\text{Ev/s}$, 10000 events were measured. In order to get rid of exceeding outliers, the depicted data points were obtained by calculating the 10% trimmed mean of all measurement results. In other words, the mean was calculated after discarding the 10% highest values as well as the 10% lowest values. Note that these explanations also apply to Figures 5.17a, 5.17b, 5.18a, 5.18b, 5.19a, 5.19b, 5.20a and 5.20b.

Comparing Figures 5.15a and 5.16a as well as Figures 5.15b and 5.16b, it turns out that the performance measurement results for enforcing ECA rules 1a and 1b within a total of three usage controlled systems ($e = 3 = t$) are very similar. Considering the signaling and evaluation of actual events in the presence of a low event frequency ($f < 50\text{Ev/s}$) as well as many relevant events (Figures 5.15a and 5.16a, 50% (\blacktriangle), 75% (\times), 100% (\bullet)), the distributed approach performs particularly bad. The reason is that in this case many state changes must be written to the distributed database, resulting in wait times for the corresponding acknowledgements of remote nodes. At the same time, however, Cassandra's benefits can not be fully utilized for low percentages of relevant events as also shown in the communication overhead evaluation in Sections 5.3.1 to 5.3.3. In addition, some of this additional overhead is attributed to Java's warm up phase for both the usage control infrastructure and the corresponding Cassandra nodes: For the distributed scenario both the code of the usage control infrastructure as well as that of Cassandra must be warmed up on three systems, whereas for the centralized approach only the usage control infrastructure needs to be warmed up on only one system. For higher event frequencies (Figures 5.15a and 5.16a, $f > 50\text{Ev/s}$) the reliance on Cassandra is more advantageous, which is why in this case the distributed infrastructure performs better than the centralized approach. Comparing the performance for low percentages of relevant events (Figures 5.15a and 5.16a, 0% ($+$), 25% ($*$)), it turns out that actual events can be evaluated more efficiently for ECA rule 1b. This observation can be explained by the different trigger events of ECA rules 1a and 1b: While ECA rule 1a is evaluated upon *every* event, ECA rule 1b is only evaluated upon event ($sendOffer, \{(obj, d)\}$) which occurs with a probability of at most 25% for $r = 25\%$, and with a probability of 0% for $r = 0\%$. In particular for $r = 0\%$ these observations are in correspondence with the measured communication overheads in Section 5.3.1.

Investigating the performance for desired events (Figures 5.15b and 5.16b), the first fact to note is that desired events never cause any state changes to be written to the distributed database, since desired events never change the system's actual state. It turns out that the decentralized approach outperforms the centralized approach for ECA rules 1a and 1b for all event frequencies and for all percentages of relevant events. Note that for desired events, unlike for actual events, event traces with *many* relevant

Figure 5.17: Performance overheads for ECA rule 1a on three out of seven systems.



events perform comparatively good. The reason is that the presence of many relevant events allows for conclusive local policy evaluations, while no state changes must ever be written to the distributed database for the reasons mentioned above.

Enforcing ECA rule 1a within three out of a total of seven usage controlled systems ($e = 3 < 7 = t$) necessitates the differentiation between those three systems that do in enforce ECA rule 1a and those four systems that do not enforce it. In the following these two sets of systems are also referred to as *active systems* and *inactive systems*. Intuitively, the performance of the latter is expected to be significantly better, since by definition no policy evaluation must be performed. The corresponding performance measurement results for actual events and desired events are depicted in Figures 5.17a and 5.17b, respectively.

Considering the centralized infrastructure, the performance is slightly worse than when enforcing ECA rule 1a within a total of three systems (cf. Figure 5.15), both for actual as well as for desired events. While this slight difference is actually insignificant, one reason for it might be the fact that in the current scenario of seven usage controlled systems more events end up waiting in the Controller's event processing queue since the probability that an event from another PEP is already being processed is slightly higher.

Looking at the performance of the distributed infrastructure, the first fact to note is that for the four inactive systems (cf. the lowest data point lines in Figures 5.17a and 5.17b) the percentage of relevant events r does not at all influence the performance measurement results. Since no policies are deployed at the corresponding PDPs and hence no matching with any such policies yields a positive result, this is not surprising. Accordingly, the distributed infrastructure performs particularly good when considering inactive systems: As expected, systems that are not part of the policy evaluation process outperform the centralized approach significantly, because for the centralized infrastructure still each event must be signaled to the central Controller.

The performance of the three active systems that do enforce ECA rule 1a is slightly worse than in the earlier scenario in which no additional inactive systems existed (Figure 5.15). The reason for this performance slowdown is that in the current implementation the remaining four inactive systems still participate as nodes in the distributed Cassandra cluster even though the corresponding PDPs do not enforce any policy that necessitates coordination with other PDPs (cf. Section 4.3.5). Hence it stands to reason that a revised implementation would yield performance results similar to the ones provided in Figure 5.15a.

In summary, there are multiple factors that influence whether the centralized or the decentralized approach performs better. When considering desired events, which do never cause any write operations on the distributed database and which never caused policy evaluation in the present scenarios, the decentralized approach always outperformed the centralized approach. When evaluating actual events, which potentially triggered policy evaluation and writes on the database, the centralized approach

performed better for low event frequencies and high percentages of relevant events. Lastly, the decentralized approach performed inexorably better when considering usage controlled systems that did not enforce the usage control policy (i.e. inactive systems): in the centralized case these systems still need to signal all observed events to the central PDP, while in the decentralized approach no such communication is ever needed.

5.3.5 Performance Overhead: Policy 2

When enforcing ECA rule 2 within a total of seven usage controlled systems, it turned out that neither the percentage of relevant events r nor the differentiation between actual and desired events did significantly influence the performance. The corresponding results are depicted in Figures 5.18a and 5.18b. In either case the decentralized infrastructure outperformed the centralized approach for event frequencies greater than 12Ev/s , $f \gtrsim 12\text{Ev/s}$. Again, for lower event frequencies also Java's warm up phase influences the measurement results. Recapping the enforced policy 2, which does not necessitate any further coordination between PDPs once the event (*declineOffer*, $\{(obj, d)\}$) has happened, it is not surprising that the decentralized approach performs better than the centralized approach. However, note that the decentralized approach's performance is better than the centralized even if the corresponding event never happens (Figures 5.18a and 5.18b, [+](#)).

Further note the better performance of the centralized approach in comparison with the enforcement of ECA rules 1a and 1b in Section 5.3.4, in particular for the evaluation of actual events. Clearly this performance improvement is due to the simpler ECA condition of ECA rule 2, thus revealing that the condition's complexity significantly influences the PDPs' evaluation performance.

5.3.6 Performance Overhead: Policy 3

Lastly, ECA rule 3 was enforced within eleven usage controlled systems ($e = 11 = t$) as well as in a setup in which five out of eleven systems were enforcing the policy, whereas the remaining six systems were not enforcing the policy and thus inactive ($e = 5 < 11 = t$). Figures 5.19a and 5.19b show the performance for eleven active systems, while Figures 5.20a and 5.20b show the performance for five active and six inactive systems.

For the centralized approach (Figures 5.19a, 5.19b, 5.20a and 5.20b, [♦](#)) the performance measurement results are very similar to the enforcement of ECA rules 1a and 1b within three active systems as described in Section 5.3.4. Since the conditions of ECA rules 1a, 1b and 3 are technically similar (cf. Section 5.3.3), this observation suggests that for the centralized approach the condition's complexity has a significantly larger influence on the performance than the amount of active systems enforcing the ECA rule.

Figure 5.18: Performance overheads for ECA rule 2 on seven systems.

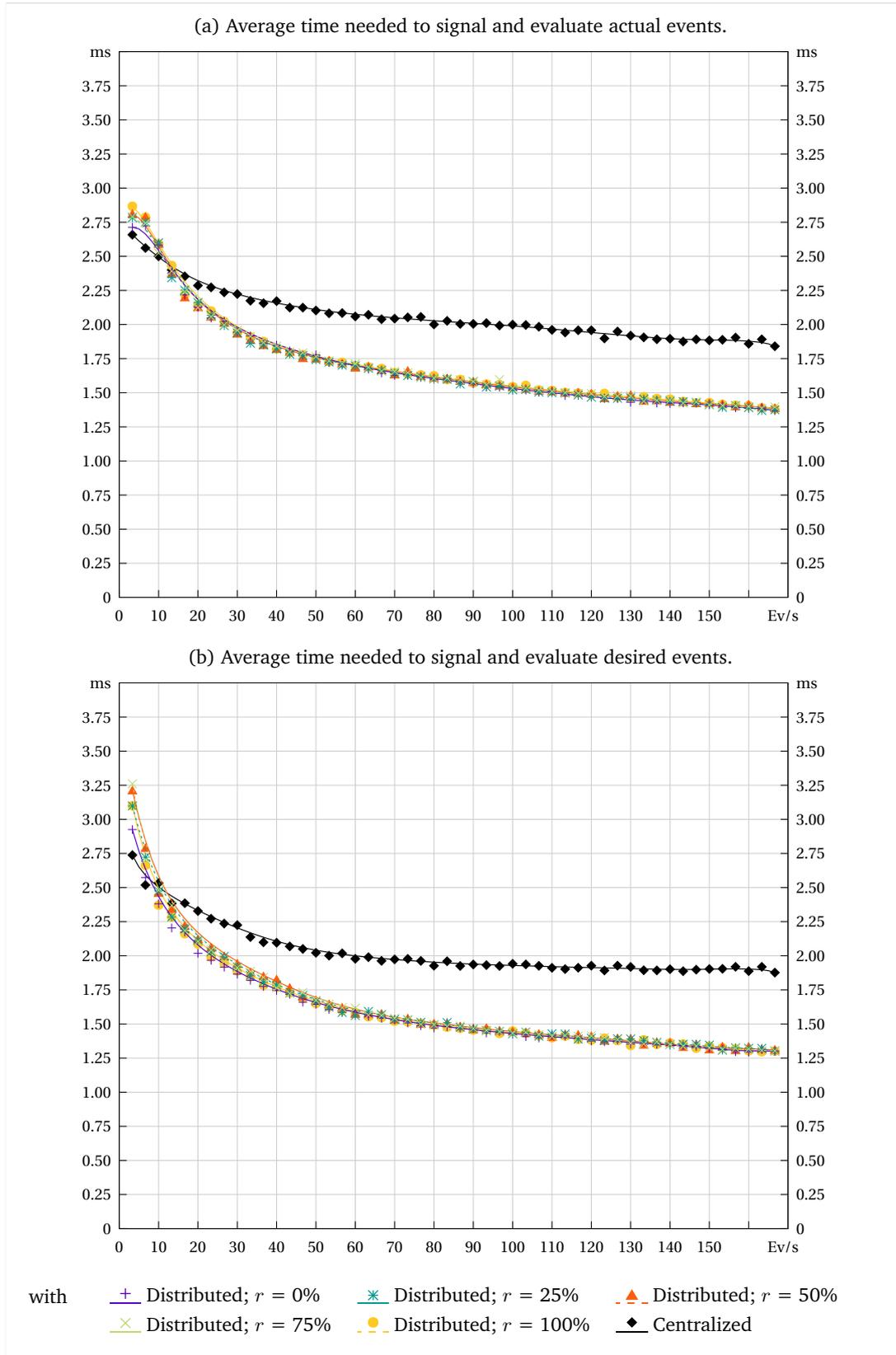


Figure 5.19: Performance overheads for ECA rule 3 on eleven systems.

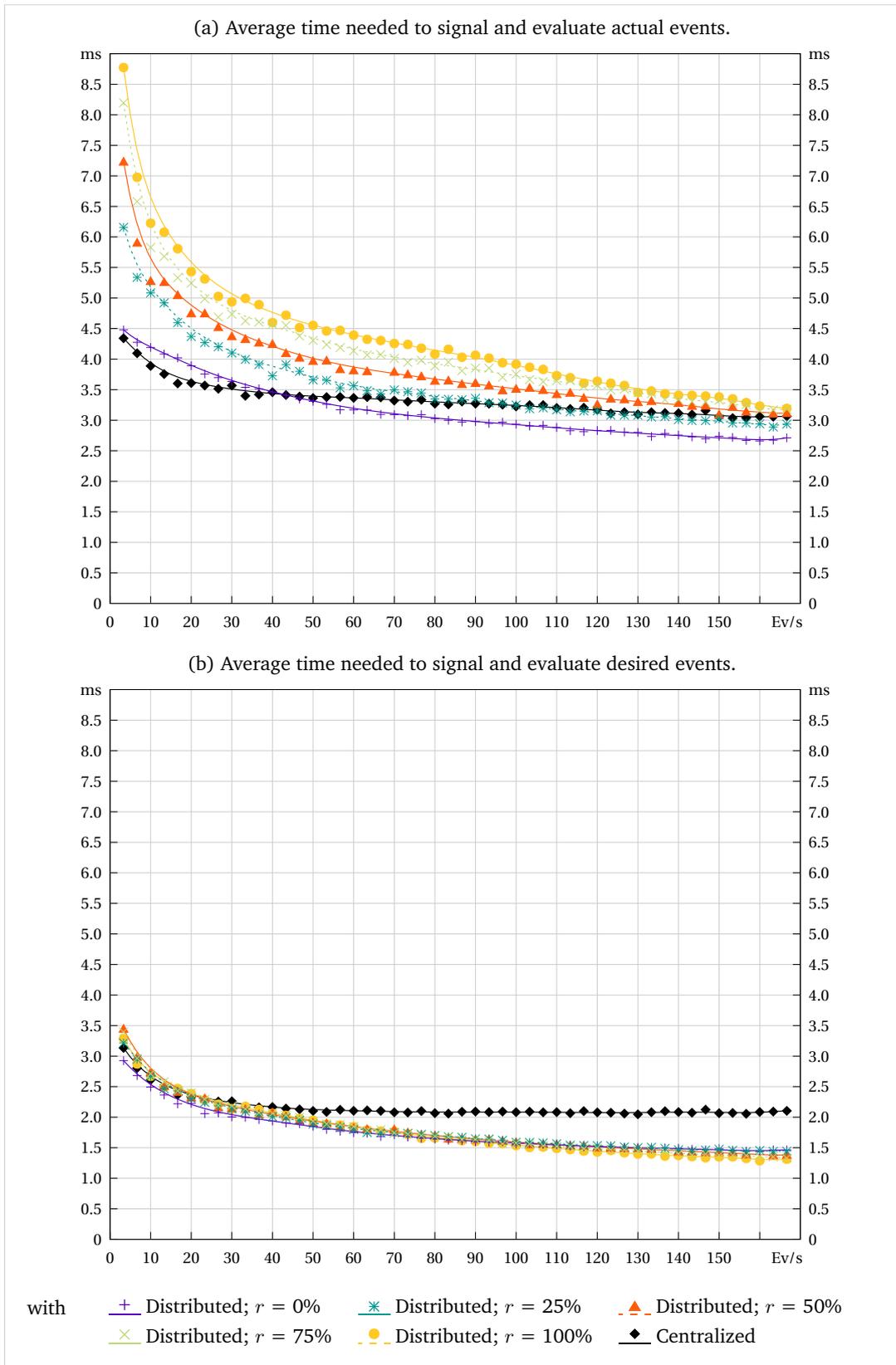
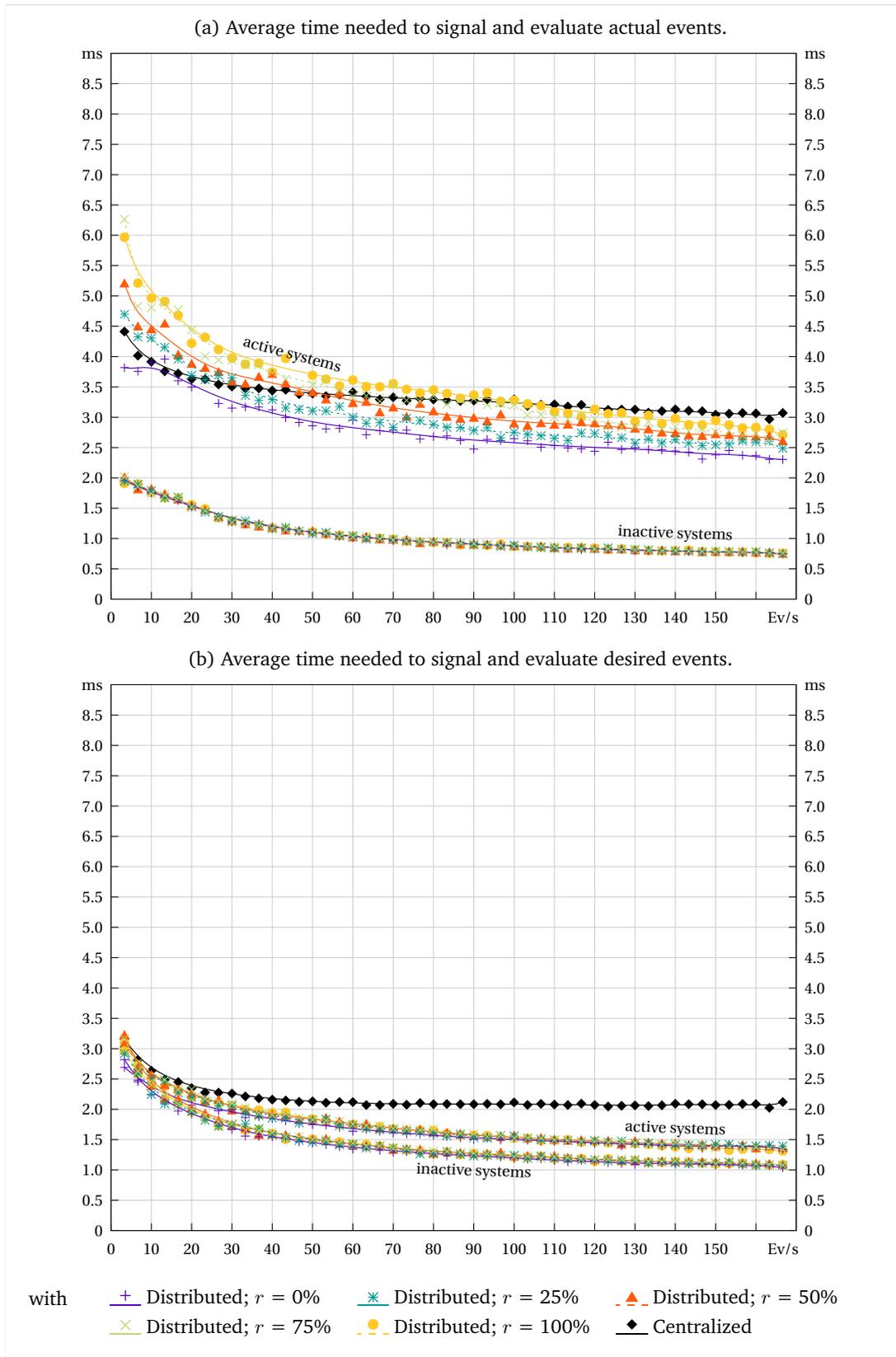


Figure 5.20: Performance overheads for ECA rule 3 on five out of eleven systems.



As for the decentralized infrastructure, some further distinctions are necessary. In the case of signaling and evaluating actual events (Figure 5.19a) the decentralized approach is only able to outperform the centralized approach for high event frequencies f and low percentages of relevant events r . E.g. for $r = 0\%$ and $f \geq 40\text{Ev/s}$ as well as for $r \leq 25\%$ and $f \geq 100\text{Ev/s}$ the decentralized approach performs better than the centralized approach. For higher percentages of relevant events and lower event frequencies, however, the overhead imposed by Cassandra is quite large. Again, a further influencing factor in this case is Java's warm up phase for eleven instances of both Cassandra and the data usage control infrastructure. Considering the evaluation of desired events (Figure 5.19b), the decentralized approach performs better than the centralized approach for larger event frequencies ($f \gtrsim 30\text{Ev/s}$); for very low event frequencies ($f \lesssim 20\text{Ev/s}$), the centralized approach performs slightly better.

Enforcing ECA rule 3 within five out of a total of eleven usage controlled systems ($e = 5 < 11 = t$), it is again useful to differentiate between the performance of active and inactive systems. The performance measurement results are depicted in Figures 5.20a and 5.20b. Comparing these results with Figure 5.17, in which three out of seven usage controlled systems enforced ECA rule 1a, it turns out that in particular the performance of the inactive systems (cf. lower data point lines in Figures 5.17a, 5.17b, 5.20a and 5.20b) is comparable. Since in those cases the corresponding PDPs do not enforce any policies and can thus immediately provide a default evaluation 'decision' upon any signaled event, this similarity in the performance measurements is not surprising. Further worth noting is the performance of active systems when evaluating actual events (Figure 5.20a). As expected, the performance is better than in the previous scenario in which a total of eleven systems were enforcing ECA rule 3. This performance improvement can be observed because only five instead of eleven PDPs synchronize via the distributed database. Since for the evaluation a Cassandra consistency level of *Quorum* was used, this implies that each operation on the database must only be acknowledged by three instead of six Cassandra nodes (cf. Section 5.3.3), thus resulting in the improved performance.

5.3.7 Summary

In summary, it depends on many different factors whether a centralized or decentralized approach performs better when enforcing global policies. For none of the evaluated use cases either of the two infrastructures performed inexorably better than the other. Hence this summary discusses which of the identified parameters influences the communication and performance overheads to which extent. Since both the communication and the performance overhead imposed by the centralized infrastructure can be rather easily quantified (the communication overhead is linear in the event frequency while the performance overhead mostly depends on the ECA rule's condition and trigger event), the following elaborations focus on a decentralized system setup.

Event Frequency. Summarizing the results of Figures 5.7b, 5.8b, 5.9b, 5.11b, 5.13b and 5.14b as well as those of Figures 5.15 to 5.20, it turns out that the decentralized infrastructure performs better the higher the event frequency: Both the communication overhead as well as the performance overhead decrease with higher event frequencies. There are two major reasons for this: (i) higher event frequencies allow to better utilize Cassandra's data distribution and synchronization capabilities since many state changes must be coordinated between the distributed PDPs; (ii) the presence of many events allows for many conclusive policy evaluations by the local PDPs and consequently necessitates less costly (both in terms of communication and performance) lookups within the distributed database.

Percentage of Relevant Events. As can be seen in Figures 5.7a, 5.8a, 5.9a, 5.11a, 5.13a and 5.14a, it is generally the case that lower percentages of relevant events cause a decrease in the communication overhead. There are two reasons for this: (i) less relevant events imply that less events match the ECA rule's trigger event which in turn causes less policy evaluations and therefore less coordination between PDPs; (ii) less relevant events cause less state changes to be written to the distributed database. Notably, however, there exist exceptions in which 0% of relevant events cause comparatively high communication overheads, cf. Figures 5.7a, 5.9a and 5.11a. Also in terms of performance overhead the decentralized approach performs better for less relevant events. Again, the reason is the reduced amount of policy evaluations due to the presence of less trigger events.

Policy. Considering the ECA rule being enforced, it is mostly the trigger event as well as the condition's complexity that influence both communication and performance overheads. As mentioned above, the policy's trigger event influences how often the policy must be evaluated, whereas the condition influences the complexity (and thus the runtime) of the policy evaluation process as well as how many state changes must be potentially exchanged between PDPs upon each evaluation. As such, it is beneficial for the decentralized infrastructure to deploy policies the evaluation of which is triggered rarely, as well as the condition of which is simple.

Total Number of Usage Controlled Systems and Number of Systems Enforcing the Policy. When compared with the centralized approach, the decentralized infrastructure performs particularly well (both in terms of communication and performance) if the ratio between those systems that do enforce the deployed policy (e , active systems) and the total amount of usage controlled systems (t) is small. The reason is that for the centralized approach each and every event of all t usage controlled systems must be signaled to the central PDP, whereas for the decentralized infrastructure at most e/t of all events (i.e. those events that are issued within the set of systems that do enforce the deployed policy) cause any policy evaluation and consequently potentially coordination between PDPs. In particular, it could be observed that the communication

and the performance overhead of the inactive systems is significantly smaller than for the centralized approach.

Using Cassandra as a Means to Coordinate Policy Decisions. Using Cassandra for the sake of synchronizing DMPs, and hence PDPs, eased the implementation of the presented infrastructure. However, being a general-purpose distributed database it may be hypothesized that synchronization mechanisms tailored to usage control requirements perform better in terms of communication and performance overheads. The main reason is that in this case domain specificities could be leveraged. However, implementing such a distributed synchronization solution is by no means trivial and is thus likely to impose significant development costs.

While the performed evaluation is by no means exhaustive, it provides first insights into how well a decentralized data usage control infrastructure as developed within this thesis might perform. In conclusion, it depends very much on the individual parameters of a given scenario whether the adoption of a centralized or a decentralized infrastructure is beneficial. Given one concrete application scenario for which the above parameters are known, experiments as the ones above could reveal which type of infrastructure should be applied in practice.

5.4 Threats to Validity

While the above experiments have been carefully designed and executed, there exist several threats to validity which are concisely discussed in the following.

Evaluation Environment and Generalizability to Real-world Application Scenarios. All experiments were executed within a lab environment and no real-world usage profiles were used within the evaluation. As for the evaluation within Section 5.2, real-world applications were used in order to transfer files between systems, supposedly resulting in event traces that approximate real-world usage profiles. The evaluation in Section 5.3, however, is based on artificial and randomly generated event traces. In addition, all executed event traces had a nominal runtime of 30 seconds. In the real world, traces would rather have a runtime of days, months and even years. For these reasons it remains open whether the results presented in Section 5.3 generalize to real-world environments in the presence of real-world usage profiles (i.e. event traces).

Randomness of Experiment Execution. In Section 5.3 the executed event traces were randomly generated. While for certain parameter values these event traces were generated and executed 20 times, such repeated experiment execution could not be performed for all possible parameter values due to the vast input space. However, for those parameters for which the experiments were executed several times, the measured standard deviations were rather small such that these results are hypothesized to generalize to other parameter values.

Choice of Parameters. The parameters considered for evaluation were manually selected, essentially relying upon common sense knowledge about which factors might influence the experiment execution. For Section 5.2 these parameters include file sizes, bit rates and applications used. The fact that in this case only trivial policies were used is likely to have influenced the results. However, the evaluation yielded consistent results, which is why it is hypothesized that the obtained results generalize to other parameter values (i.e. other file sizes, bit rates, and applications). For Section 5.3 the choice of parameters included policies, the amount of systems, event frequencies, and percentages of relevant events. For this set of experiments the choice of policies is a particular threat to validity, as there exist endless possibilities to formulate policies which might yield fundamentally different results.

System Setup and System Environment. For experiment execution a virtualized system environment (i.e. virtual machines and a virtual network) was used. Hence, the performed experiments do not reveal how the evaluated infrastructure would perform in case physical machines and networks were used. Along similar lines, other kinds of systems (e.g. tablets, mobile phones, integrated devices) and networks (e.g. mobile networks) might yield different results when equipped with the developed and evaluated usage control infrastructure.

Missing Peer Review. Another important threat to validity is that the design, the execution, as well as the evaluation of the experiments was performed by one single person, thus leading to inevitable biases. Ideally, all of these tasks should have been performed by individual persons and peer reviewed for correctness or at least plausibility.



Related Work

6.1 Data Usage Control

Usage control as a generalization of access control was first introduced by Park and Sandhu [164, 166]. Further research led to development of the $UCON_{ABC}$ usage control model [165, 185], which focused on single systems and addressed the shortcomings of traditional access control models which fall short when it comes to the protection of objects once they are accessed. Thus, besides authorizations, $UCON_{ABC}$ includes obligations and (environmental) conditions “as part of the decision process to provide a richer and finer decision capability” [165]. To this end, $UCON_{ABC}$ imposes to evaluate authorizations and conditions both before and during the usage of an object. Similarly, obligations might need to be performed both before and during the usage of an object. Being flexible and generic, the model has been shown to support Mandatory Access Control (MAC), Discretionary Access Control (DAC), Role-Based Access Control (RBAC), Trust Management, as well as Digital Rights Management (DRM) (cf. Section 6.6). However, differently than DRM, usage control also encompasses the protection of end users’ objects and data [165].

Hilty et al. further investigate obligations in the context of usage control [78]. Most importantly, their work differentiates between observable and non-observable events, which is necessary to distinguish between controllable and observable obligations [171]: While the adherence to controllable obligations can be technically ensured, the violation of observable obligations can only be detected in hindsight. These considerations also lay the grounds for preventive and detective enforcement of policies [175]. In further work the Obligation Specification Language (OSL) is introduced [79], a general-purpose language to specify data usage control requirements upon which this thesis builds. On this basis, first consumer-side policy enforcement mechanisms, which operate on traces of events, are proposed [172]. By integrating these models and mechanisms with a generic model for data flow tracking [74, 170], it becomes possible to enforce data usage control policies on all copies of some sensitive data.

A comprehensive survey on usage control is provided by Lazouski et al. [116].

6.1.1 Policy Specification

Before policies can be enforced by technical usage control infrastructures, they ought to be specified in a suitable language by eligible entities.

While this thesis builds upon the Obligation Specification Language, other models and languages such as UCON_{ABC} [165], XACML [160], Ponder [40, 211], Metric First-Order Temporal logics [15, 16, 35], and the PrimeLife Policy Language [7, 26, 205] have as well been used for the specification of data protection requirements. A comparison of some of those languages is provided in [73, 105].

Most relevant for this thesis is the work by Kumari et al. [106, 108, 109], which translate high-level specification policies into low-level implementation policies. The motivation is that regular end users aiming to protect their data are usually not capable of writing policies in low-level policy languages. Thus, the idea is to provide assistance to end users by developing templates which can then be easily instantiated by regular users. Such an instantiation then results in ECA rules as described in Section 2.1.3, making the instantiated policy template enforceable by the infrastructure provided in Chapter 4. Policy templates are obtained by (i) having expert users define the problem domain, including high-level user actions, systems, low-level system events, data, data representations and their interrelations, (ii) having expert users define high-level specification policies in terms of future temporal logics, (iii) automatically translating these policies into low-level policies in terms of past temporal logics.

The work in [177] investigates the problem of evolution of data usage control policies upon re-distribution of the corresponding protected data, i.e. how policies may be altered before shipping them to other subjects. The general idea is to only allow strengthening of policies by users that have a particular role. Thereby, strengthening of policies boils down to reducing rights granted by the usage control policy and/or imposing additional duties. To this end, the paper proposes to impose a partial order onto event names as well as parameter values. Upon re-distribution, policies may then only be altered in correspondence with this partial order.

Section 7.2 provides a further discussion on the question which users may specify and deploy policies for which data.

6.1.2 Policy Enforcement

Policy Enforcement Points that are compatible with this thesis' infrastructure (cf. Sections 2.2 and 4.1) have been developed for many different system layers as described in Section 2.2.2. Since these solutions' general concepts are very similar to what has been described in this thesis, they are not further investigated at this point.

Being a prominent usage control model, $UCON_{ABC}$ has been applied in many different contexts, some of which are concisely described in the following. Note that none of these solutions integrates with data flow tracking technology. Katt et al. [91] implement usage control for a health care information system, whereby the PEP is implemented for a simple text editor. To represent $UCON_{ABC}$ policies, the solution develops a XACML policy specification. Xu et al. [217] leverage $UCON_{ABC}$ to protect the integrity of operating system kernels. This is achieved by implementing a usage control monitor at the virtual machine level. Zhang et al. [225] implement $UCON_{ABC}$ for collaborative systems in which sensitive shared resources ought to be protected, e.g. by only allowing access from certain locations or by only allowing one user to access an object at each point in time. The prototype integrates with Subversion and WebDAV; $UCON_{ABC}$ policies are specified using XACML. Karopoulos et al. [90] integrate $UCON_{ABC}$ with SIP (Session Initiation Protocol) applications, thus supporting authorizations, obligations and conditions for multimedia delivery. E.g., this approach could be used to automatically issue a payment once multimedia content is streamed.

Martinelli et al. [37, 134] realize usage control for grid computational services by making the grid user deploy her application together with the policy. The application is monitored at the level of the Java Virtual Machine, whereby system calls are considered as security-relevant actions. Their approach differs in that the policy is defined for the application instead of data. Since this approach does not consider data flows, cooperating applications could circumvent the usage control enforcement.

Mont et al. [143, 144] propose a technical framework to model, deploy and enforce privacy-aware access control policies and privacy obligations which, among other things, cater to the purpose of collection and usage of data. The framework is integrated with an access control system as well as an identity management component. After deploying privacy policy constraints at the Policy Decision Point (PDP), a Data Enforcer (i.e. PEP) at the SQL database layer, intercepts accesses to personal data (i.e. SQL queries) and enforces the PDP's decision. In order to retrieve such a decision from the PDP, the Data Enforcer must not only signal the attempted access to the PDP, but also information such as the access' purpose and its subject. Besides allowance and inhibition, the PDP might also decide to allow conditional access, e.g. only if the accessed data is filtered before access.

A survey on usage control enforcement mechanisms is provided by Nyre [158].

6.1.3 Intra-System Data Flow Tracking

Many solutions for tracking data flows within one system have been proposed. While this thesis mainly leverages data flow tracking technology at the operating system layer (Section 3.2), integrating data flow tracking technologies at multiple layers would allow for more fine-granular cross-system data flow tracking. A discussion of intra-system data flow tracking technologies follows.

In the area of data usage control, tailored data flow tracking solutions have been proposed and implemented for Mozilla Thunderbird [125], X11 [170], Java [61], MS Office [188], MySQL [119], JavaScript [168], MS Windows [216], and Chromium OS [214]. All of these solutions leverage the generic data flow model introduced in Section 2.1.2 and provide a corresponding instantiation. Generally, data flow tracking technology at the application-layer [125, 188] has the advantage that application-specific abstractions and knowledge can be used to perform fine-grained data flow tracking. On the other hand, such application-specific solutions must be developed for many different applications since they are generally not reusable within different contexts. In contrast, data flow tracking technology at the operating system layer [214, 216] or even at the CPU instruction layer is more general and can be used to track data flows both within and across applications. However, such solutions can not leverage application-specific knowledge. As a third option, data flow tracking at an intermediate platform layer [61] might be able to compromise between the above (dis-)advantages.

Recent research combines data flow tracking technologies at different system layers [126, 174]. The idea is to leverage knowledge about system events and their interrelation at *multiple* system layers and to combine this knowledge to create data flow tracking technology that spans multiple system layers. Using such an integrated solution, it is possible to track data flows both within and across system layers and to compensate the disadvantages of the individual solutions mentioned above.

While not specifically developed with data usage control in mind, solutions for data flow tracking have been developed in different contexts as presented in the following. Integration of these solutions with data usage control technology would improve the granularity of data flow tracking and consequently policy enforcement.

A framework that dynamically and transparently gathers data flow information for any kind of application without code modifications is presented in [67]. Building upon DTrace [27], data flows within applications can be recognized even if their source code is not available. The proposed solution does not intend to perform any policy enforcement, but rather to log code points, system calls, or function calls whenever some data passes them. The framework has been instantiated at the system call layer for file manipulation, for the SQLite database [81], as well as for the Safari browser.

libdft [97] achieves intra-system data flow tracking without the need to modify the applications being monitored or the underlying operating system. For this, libdft leverages Intel's Pin dynamic binary instrumentation framework [128], which allows to instrument every program instruction that moves or combines data. Data flows are monitored at the granularity of single bytes and each byte may be tagged with up to eight distinct values. These tags are updated in correspondence with the observed program instructions. Since libdft also allows to hook system calls, it could be used as an alternative to `strace` which is used by the PEP presented in Section 2.3. By operating at the granularity of single bytes, libdft could be leveraged to improve on the precision of data flow tracking in the context of data usage control.

DBTaint [44] provides data flow tracking for the PostgreSQL database at the granularity of single cells. By leveraging SQL rewriting techniques, this approach is transparent to applications. In comparison with system-wide taint tracking solutions such as libdft, this approach has both the advantage and disadvantage of being tailored to one particular kind of application, i.e. databases. While making the solution less generic, it also allows to leverage application-specific knowledge and semantics.

LabelFlow [34] is similar to DBTaint by tracking data flows within the MySQL database at the granularity of rows. However, LabelFlow integrates data flow tracking for databases with data flow tracking for web applications written in PHP. The goal is to provide an API that allows to improve the security and privacy in legacy applications without major code changes. Besides tracking data flows, LabelFlow allows to place hooks in the application code and to enforce security policies upon reaching those hooks. However, this specification of policies is rather ad-hoc and application-dependent.

Asbestos [50] is a prototype operating system providing isolation mechanisms in order to “contain the effects of exploitable software flaws”. To this end, Asbestos makes use of labels which are, among other things, used to “track and limit the flow of information within system- and application-defined compartments”. By allowing processes to keep private and isolated states for multiple users, contamination of a process is limited to the compartment of single users—effectively tackling the problem of data flow tracking overapproximations.

Similarly, HiStar [221] is a new operating system providing “strict information flow control” to the end of minimizing the trusted code base. For this, HiStar allows to attach labels to sensitive files and other objects, such as threads, address spaces, and devices, which are then propagated in correspondence with observed system calls. Security properties can be enforced by controlling where the labeled information might flow.

Similar to Asbestos and HiStar, Flume [103] provides process-level information flow control at the granularity of processes and communication abstractions such as sockets, pipes, and file descriptors. Corresponding tags are propagated in correspondence with system calls. However, unlike Asbestos and HiStar, Flume is implemented in user-space within a traditional Linux environment.

Lastly, many recent works address the problem of collecting, storing and maintaining data provenance at a multitude of software layers [24, 191]. While these technologies have not been developed for data usage control purposes, most of them could be leveraged in order to improve the precision of data flow tracking. More concretely, provenance solutions have been developed for file systems [153, 187], databases [2, 23, 25], service-oriented architectures [57, 58] and many different kinds of application-layer software, e.g. [132]. Finally, and similar to Lovat in the usage control context [126], Muniswamy-Reddy et al. [152] design and implement a framework that integrates provenance collection and maintenance across multiple software layers, i.e. Python, a workflow engine, a text-based web browser, and a file system.

6.2 Cross-System Data Flow Tracking and Policy Propagation

By discussing works related to cross-system data flow tracking and policy propagation, this section constitutes the basis for **RQ1** (Section 1.1.1) and complements the corresponding solutions, contributions and evaluation presented in Sections 3.2, 4.2 and 5.2.

Fundamental conceptual work on sticky policies is provided by Chadwick et al. [29], which propose and compare three different models for propagating policies across systems along with the corresponding protected data. By not requiring any changes to the monitored applications, the proposed ‘back channel model’ is most close to this thesis’ approach. In this model, the communication between the PEP and the PDP is mediated through an application independent PEP (AIPEP). Once protected data is sent to a remote system, the AIPEP is responsible for sending the policy to the AIPEP of the receiving system. This thesis splits the AIPEP’s responsibilities in a manner that is more suitable for the data usage control context: while the PIP detects cross-system data flows, the DMP is responsible for propagating the corresponding policies. Notably, [29] does not provide any implementation of the proposed models.

Building upon libdft (cf. Section 6.1.3), Taint-Exchange [220] is a framework that generically and transparently tracks data taints across processes both within and across systems. Similar to the cross-system data flow tracking approach presented in this thesis, Taint-Exchange intercepts relevant system calls. A global array keeps track of all important channels, i.e. pipes and sockets, through which tainted data might be exchanged and is updated in accordance with the observed system calls. Once the framework detects that tainted data is written to one of these channels, a taint-header is attached to the payload. Both the payload data and the taint-header are then sent to the destination. Upon receiving the combined data, the framework detaches the taint-header. While the payload data is handed to the receiving process, the taint information is handed to the libdft framework for further intra-process taint tracking. A limitation of Taint-Exchange is that it only makes use of one bit taints, effectively not allowing to differentiate between different data. At the same time, however, the underlying libdft framework allows for taint tracking within processes at a granularity of bytes, which is currently not possible in this thesis’ approach. On the other hand, such fine-grained tracking also comes with the drawback that grasping higher-level application semantics becomes much more difficult. In any case, libdft or similar frameworks could be integrated into the work presented in this thesis. It remains unclear how Taint-Exchange handles forking of processes, i.e. whether forked child processes are monitored as well and whether data taints are propagated to the forked child process. Most importantly, this thesis goes beyond Taint-Exchange’s taint tracking capabilities by propagating complex data usage policies rather than simple labels. Different to this thesis’ approach, Taint-Exchange requires the presence of additional hardware (i.e. Intel’s Pin) as it builds upon libdft.

CloudFilter [161] aims at controlling data propagation between enterprises and cloud storage services. More precisely, the solution allows employees of an enterprise to upload confidential files to cloud storage services while ensuring that “(i) the operation is logged, (ii) the action can be attributed to a user and (iii) other users can only access [the file] from a designated set of networks ” [161]. To this end, a web browser extension detects file uploads at the HTTP layer and attaches corresponding metadata about the uploaded file. Once this HTTP request is intercepted by the enterprise’s proxy, the metadata is evaluated and according Event-Condition-Action rules are triggered. Corresponding actions may, among other things, log the upload, ask the user for a security label for the file, or deny the upload altogether. A server-side proxy will then receive the uploaded file and attached policies. While at the server side no policy enforcement is taking place, the attached policies will be evaluated and propagated once a user downloads the previously uploaded file. Different to the approach in this thesis, CloudFilter is limited to one single application-layer protocol, namely HTTP. Also, the presented implementation does not yet support file transfers over HTTPS. Moreover, CloudFilter relies on the fact that the cloud storage service stores the uploaded content in terms of files. Some of these limitations stem from CloudFilter’s different trust model, in which end users are *not* expected to *intentionally* break the security framework. This thesis goes beyond CloudFilter by also constraining the data usage both within the client side and the server side system. In addition, the approach taken in this thesis does not necessitate the adaptation of existing applications as is the case for CloudFilter. However, the idea of tracking data flows across systems at the HTTP(S) layer using proxies could be integrated with this thesis in order to perform more-fine grained data flow tracking.

The goal of CloudFence [162] is to support benign cloud service providers in enhancing the security of their provided services. The work strictly distinguishes between infrastructure providers and providers of higher-level services building upon those infrastructures. In a nutshell, CloudFence is a data flow tracking framework which is supposed to be deployed by cloud infrastructure providers and to be used by higher-level service providers and end users. Essentially, CloudFence provides an API for service developers which allows to tag sensitive data of end users using taint marks. These taint marks are then transparently propagated in correspondence with the observed CPU instructions by the underlying infrastructure. For this, Intel’s Pin dynamic binary instrumentation framework [128] is leveraged. While observing the data flows throughout the cloud infrastructure, CloudFence stores corresponding data trails. As such, CloudFence’s data flow tracking capabilities are comparable to Taint-Exchange and to what has been developed within this thesis. Additionally, however, CloudFence integrates with a web-based dashboard [219] which allows end users to visualize the audit trails of their data. By using 32-bit taint marks, CloudFence supports up to 4 billion distinct taint marks. However, this value decreases as taint marks are combined—in the worst case allowing for only 32 distinct taint marks. Along the same

lines, memory consumption of CloudFence is enormous: Due to the 32 bit taint marks for each tracked byte of program data, four additional bytes are needed for its taint mark. Different to the work presented in this thesis, CloudFence only allows to track a limited set of distinct data. Further CloudFence relies on additional hardware and is not capable of propagating complex data usage policies.

Trabelsi et al. [19, 206] tackle the problem that cloud applications do not provide technical guarantees that a user's data is used in accordance with privacy regulations and user preferences. Thus, they provide a software layer called SPACE which mediates all accesses and usages of the protected data. In their scenario, users access cloud services using their mobile phones. Whenever a user sends sensitive data to a cloud provider, a client-side service attaches corresponding access and usage control policies to the data. Upon data access and usage by the cloud provider, SPACE enforces compliance with those policies. As SPACE is implemented using the concept of sticky policies, the policies travel with the data upon further data dissemination. Furthermore, SPACE allows the user to investigate accesses to her data, as well as at which IP addresses his data has been stored. While very promising from an abstract point of view, no details about the implementation are provided. Thus it remains unclear how the proposed sticky policy paradigm is actually implemented. Hence, advantages as well as limitations of the proposed approach are hard to identify. For example, it remains unclear whether sticky policies are bound to certain file formats as in [115].

Neon [224] is a monitoring system at the hypervisor level which allows for transparent information flow tracking both within and across systems. While intra-system tracking is achieved at a granularity of single bytes, cross-system tracking is performed at a granularity of network packets. Neon achieves its information flow tracking capabilities by associating each byte of memory with a 32-bit label, where each bit is used to represent a distinct policy. This allows for a combination of 32 policies. Neon has been implemented within Domain 0 of the QEMU emulator [179], leaving the operating system as well as applications unmodified. Locally, taints are propagated by an appropriate QEMU process whenever the virtualized CPU handles tainted memory addresses. For remote taint propagation, the QEMU process taints each outgoing IP packet with the corresponding taint values by reusing the 8-bit Type-of-Service field, effectively performing some overtainting with respect to the intra-system byte-level tainting. Besides taint propagation, Neon is able to enforce information flow control policies at the network/firewall layer. Corresponding firewall rules might mandate encryption, logging, rerouting, or dropping of tainted network packets. While Neon's intra-system byte-level taint propagation is orthogonal to what has been presented in this thesis, the cross-system information flow tracking capabilities of Neon and this thesis are similar. The main difference is in the implementation: Neon performs tracking at the hypervisor/firewall layer, this thesis features an implementation at the operating system layer. One major limitation present in Neon but not in this thesis

is the limitation to a fixed number of policies (i.e. 32) as well as the reliance on a modified hypervisor. Besides, this thesis integrates cross-system data flow tracking with the enforcement of expressive data usage policies.

While their main goal is to control which subject is allowed to receive which information in a distributed setup, the solution proposed by Janicke et al. [89] also performs cross-system data flow tracking and policy propagation. The key idea is to build a middleware that intercepts application-layer events in order to track the flow of data within that application. Data dissemination policies stipulate which data may be sent to which recipients. Once the PEP observes an event of sending protected data to a remote location, the PDP decides based on the deployed policies whether the data flow is allowed. If so, then the corresponding policies are attached and the data is sent. The receiving PEP will then detach the policy from the payload and deploy it at its local PDP. Being implemented for the Java-Bytecode, the prototype results “in overheads of factor 2-3 in execution time and memory usage” [89]. Unfortunately, no further details about the implementation are provided. At this abstract level, the approach taken in this thesis is different in that it builds on a more expressive data flow model and policy language. Further, the approach of this thesis is built into a lower system layer, i.e. the operating system, thus applying not only to Java applications. In sum, the work by Janicke et al. [89] can thus be considered a subset of what is presented in this thesis.

The goal of DStar [222] is to make distributed applications more secure despite the presence of potentially malicious application code. For this, information flow control solutions for single operating systems [50, 103, 221] (cf. Section 6.1.3) are extended to distributed systems. The proposed solution tracks information flows both within and across systems using labels which may be associated with processes as well as network messages. These labels are supposed to be in a partial order, which determines whether data might flow from one process/message to another. When some data is sent from one system to another, DStar determines whether the attempted cross-system data flow is permissible. By also operating at the operating system layer, DStar’s granularity of information flow control is similar to the cross-system data flow tracking presented in Section 3.2.2. Different from this thesis, DStar has been designed for the sole application in the information flow control context. Consequently, the propagation and enforcement of more complex data usage policies was neither intended nor can it be easily integrated. Besides, DStar is not transparent to applications, which must, among other things, explicitly define trust between different systems. Again, this is different to the approach in this thesis where all data flow tracking is transparent to the applications.

Similar to DStar, Evans et al. [53] leverage information flow control to ensure that inter- and intra-organizational information flow adheres to high-level data sharing agreements. Different from DStar, their focus is on event-driven systems, in which senders publish events for which receivers might subscribe. To perform information flow

control, labels are assigned to data and events, as well as to data processing units. By comparing these labels, it can be determined which processing unit is allowed to process which data or event. If labeled events are sent to another organizational domain, trusted gateways in both the sender's and the recipient's organization ensure that the events are only forwarded to data processing units that are labeled appropriately. Different from this thesis, the proposed approach is concerned with sole information flow control and does not support to enforce or propagate more complex policies.

Lastly, recent research shows how data provenance technology (cf. Section 6.1.3) can be extended to distributed systems. Bădoiu et al. [10] design and implement a provenance aware distributed file system. The proposed solution intercepts operations, i.e. system calls, at the virtual file system layer and passes them to a custom file system layer. This custom file system layer is responsible of collecting and maintaining the provenance based on the intercepted system calls. It is shown that the proposed approach works for the Network File System [190]. Further research extends the collection of provenance metadata to distributed systems [66, 130], distributed enterprise service buses [4] and cloud services [154, 155, 223]. In particular, [154, 155] integrate provenance into the Amazon Simple Storage Service (S3). The provenance information itself is collected in correspondence with system calls issued by the client application that accesses files from S3. Once the application closes a file which is stored within S3, the provenance framework takes care of storing the collected provenance information. Depending on the chosen architecture, the provenance data is stored along with the payload data in S3 or within Amazon's SimpleDB database. In general, however, data provenance is only concerned with the collection of information about data flows and does not consider the propagation or enforcement of any policies.

Summary. While there exist many approaches that track data flows across systems and propagate corresponding policies, they also come with several shortcomings that are not present in thesis' approach. (1) Many of the discussed approaches present obstacles that make them difficult to deploy within commodity systems: [162, 220] necessitate additional hardware by building upon Intel's Pin dynamic instrumentation library; [161, 162, 223] necessitate the adaptation of existing applications; [224] necessitates the presence of a particular hypervisor. (2) Some approaches are not generic as they are tailored to specific applications and/or application protocols [89, 161, 223]. (3) Other approaches are limited in the number of distinct labels or policies that can be tracked [162, 220, 224]. (4) Lastly, many approaches only support the propagation of simple labels rather than complex policies [53, 162, 220, 222, 223].

6.3 Distributed Policy Decisions

This section discusses related work that covers decision making in case different aspects of the decision process are distributed. Such aspects might include the events being observed/controlled, resources being used, as well as additional information such as

attributes of subjects or resources. As such, this thesis constitutes the basis for **RQ2** (Section 1.1.2) and complements the solutions, contributions and evaluation presented in Sections 3.3, 4.3 and 5.3.

Service Automata [64] aim at enforcing policies that cannot be decided locally. For this, so-called local ‘service automatons’ monitor the execution of programs within a distributed system. Formally, Service Automata are modeled as Communicating Sequential Processes (CSP), where the internal components of an automaton, roughly equivalent to PEP, PDP and PMP within this thesis, run in parallel and communicate with each other. In case an automaton’s knowledge is insufficient to take a policy decision, an automaton-internal component called ‘coordinator’ delegates the decision process to some other automaton via delegation requests/responses. Different to the approach taken in this thesis, each security-relevant event is statically mapped to one single, possibly remote, responsible automaton. Thus, whenever Service Automata are instantiated for a new application scenario, possibly conflicting events must be mapped to the one service automaton responsible for the corresponding decisions. In contrast, the approach taken in this thesis does not rely on such a static mapping, but allows each local PDP to take the corresponding decision. Mapping each event to one single responsible automaton also comes with the drawbacks of being one single point of failure and that communication with this responsible automaton must always be possible. Lastly, Service Automata do not cater to dynamic data flows and the fact that the data to be protected might be copied both within and across systems.

CliSeAu [63] builds upon Service Automata [64] and allows to secure distributed Java programs. Similar to the work presented in Section 3.3, CliSeAu enables the enforcement of policies for which the knowledge of one single local entity (called ‘enforcement capsule’ (EC) in CliSeAu) is insufficient. To this end, CliSeAu provides a framework to intercept security-relevant actions, i.e. Java method calls, and to specify and enforce security policies. If an EC intercepts an action for which its local knowledge is insufficient to take a policy decision, a decision request is delegated to another EC. This process might be repeated several times until enough information for taking a decision has been gathered by multiple ECs. CliSeAu differs from the approach taken in this thesis as it does not consider data flows. It remains unclear how CliSeAu copes with copies of the data for which the original security policy was specified. Moreover, in CliSeAu all related ECs are configured with the corresponding security policy from the beginning. While this renders remote policy propagation as presented in Section 4.2 unnecessary, it also makes the framework less flexible. In addition, the formal approach for taking distributed policy decisions presented in this thesis (Section 3.3) addresses the problem in a more general manner, as it does not fix one particular implementation scheme, such as delegation used in CliSeAu.

Lazouski et al. [115] provide a framework and a prototype implementation allowing for the enforcement of usage control policies if data copies are distributed. The authors

embed data usage policies into the data to be protected, effectively allowing the policy to transparently migrate whenever the protected data is migrated. Besides access and usage control rules, so-called PDP/PIP allocation policies are embedded into the protected data. These allocation policies specify which PDPs and PIPs are involved in the decision process and how they can be reached. Subject and object attributes required for policy evaluation might be stored at different PIP locations, which will be queried by the PDP as required. The work by Lazouski et al. is different from the work presented in this thesis in several ways. Embedding policies into the protected data comes with the drawback that implementations of the framework must be able to cater with different file formats. Usage control enforcement is likely to break if the protected data is converted into non-supported formats. This is inherently different in the approach taken in this thesis, as policies are not embedded within the protected data, but associated with it at the infrastructure level. Moreover, PDP/PIP allocation policies as proposed by Lazouski et al. fix the PDPs and PIPs for the data's entire lifetime. Consequently, the same PDP will take all decisions w.r.t. a data's usage, and for each attribute the same single PIP will always maintain its state. From this perspective, the approach still depends on central components, effectively introducing several single points of failure: if only one of the PDPs and/or PIPs mentioned in a PDP/PIP allocation policy breaks, no more policy evaluation is possible. In contrast, the approach presented in this thesis does not employ any central components.

Basin et al. [12, 13, 14, 17] perform offline monitoring of usage control policies in distributed systems. In their work policies are formalized in Metric First Order Temporal Logic (MFOTL), a logic that is very similar to the Obligation Specification Language (OSL) which was used in this thesis. In their approach system logs, i.e. sequences of timestamped events, are decentrally collected by local monitors and later pre-processed and merged for further evaluation against MFOTL policies. Their work particularly considers the challenge that multiple concurrent monitors may associate events with the same timestamp, effectively only imposing a partial rather than a total order on the set of decentrally observed events. Differently, the approach taken in this thesis does not specifically consider the problem of partially ordered event traces: the distributed Cassandra database implicitly imposes an order on any pair of events that (i) happen within the same timestep, (ii) happen at different systems, (iii) are relevant for more than one systems for policy evaluation. Similar to this thesis' approach in Section 3.3.2, Basin et al. also identify situations in which the evaluation of 'partial logs' is sufficient because satisfaction (violation) of of this partial log implies satisfaction (violation) of the overall merged log. Most importantly, the results of this thesis do not only support a posteriori detection of policy violations, but they also allow for the preventive enforcement of policies, e.g. by denying events which would otherwise result in a policy violation.

While not concerned with actual distribution of policies and/or PDPs, PIPs, and PEPs, the work presented by Janicke et al. [88] addresses the concurrent enforcement of dependent usage control policies. It is identified that such dependencies occur if policies are stateful, e.g. by referring to attributes and/or history. By deploying one single PEP/PDP/PIP (the combination thereof is called ‘controller’ in [88]), the problem could intuitively be solved by interleaving all access and usage decisions. However, the authors argue that such an approach is prohibitively expensive, in particular if some of the deployed policies could actually be enforced independently of each other. The presented solution is a static analysis of the policies, breaking them down into constraints that can be independently enforced by different controllers. Two constraints are said to be independent if they do not share any mutable attributes. The output of a corresponding algorithm is visualized as a dependency graph: independent subgraphs can be enforced without any interaction between controllers, while nodes that are connected but not neighbours can be concurrently enforced by the *same* controller. The presented approach is complementary to what has been presented in this thesis. While the static policy analysis is performed for policies written in Interval Temporal Logics [146], the same approach could be performed for the policy language presented in Section 2.1.3, effectively integrating the presented solution with this thesis’ results.

The work by Cormode [39] is more fundamental by providing a model in which distributed observers each see a stream of observations and which would like to compute a global function of these distributed observations. At this abstract level, the proposed model is capable of expressing the distributed data flow state (Section 3.2) and the distributed policy decisions (Section 3.3). While the proposed model is very generic, the discussed instantiations are, however, very narrow and leverage particularities of the function to be computed in order to reduce communication overhead. Most importantly, the problem of *threshold counting* [100] is discussed, in which the decentral observers would like to know whether the total number of occurrences of a certain event is below a certain threshold. While the general model still features a central component, the obtained theoretical results on how to improve communication cost in the context of threshold counting can be leveraged to improve on the performance of cardinality operators such as *repm* in this thesis. Besides, Cormode proposes to develop systems dedicated to distributed monitoring “which are as accessible and as general purpose as traditional centralized [databases]”. While Cormode is unsure “what classes of functions such tools should support”, in our case these would be functions related to distributed data flow tracking, policy propagation, and distributed policy decisions. Thus, this would boil down to implementing a distributed database tailored to usage control requirements.

Alzahrani et al. [5, 6] propose and implement an overlay network that allows for the dynamic and distributed deployment of multiple PDPs which are expected to collaboratively enforce history-based XACML policies such as dynamic separation of

duties [54]. Similar to the approach taken in Section 3.3.2, policies are analyzed and divided into sub-policies which can be independently enforced by different distributed PDPs. Policies which still pose dependencies among another, are “distributed among various synchronized PDPs” [6]. Different from this thesis, Alzahrani et al. assume that the objects being protected by policies are static, meaning that they are neither moved across nor within systems and/or domains.

Chadwick et al. [28, 30, 31, 195] aim at controlling the cumulative use of distributed but static grid resources. To this end, high-level grid-wide access control policies for distributed resources are decomposed into low-level policies that can be enforced at each site individually. Hereby, policies are formalized as ‘arithmetic and logical expression trees’ (ALET) whose non-leaf nodes may be composed of arithmetic operators, relational operators, logical operators, and complex functions; leaf nodes could be constants or variables, possibly referring to attributes of subjects or resources. In a nutshell, the policy decomposition algorithm leverages the fact that resources are organized hierarchically and that a high-level resource (e.g. a computing facility) represents multiple low-level resources (e.g. PCs 1 to 3, printer A, scanner B). It is then determined which parts of the ALET can be evaluated by which sites and accordingly ALETs are created that can be evaluated independently and locally. Attributes or values that are shared between multiple PDPs are coordinated via a, possibly distributed, database. While the general approach taken in Section 3.3 is similar to the above works, the considered policy languages differ significantly. Further, this thesis allows to protect data that keeps being propagated throughout the entire distributed system. In addition, this thesis showed the correctness of the performed policy decomposition.

Bauer et al. [18] propose an approach for decentralized LTL monitoring, i.e. observing satisfaction of LTL formulas in a distributed system without a central data collection point. For this, the work assumes a synchronous system bus which acts as a global clock, whereas the decentralized components emit events that can be observed by local monitors. The central idea to monitor LTL formulas in a decentralized manner is based on formula rewriting, which essentially splits a formula into a part that must currently be satisfied and another part that must be satisfied in the future. After a given LTL formula was distributed to all local monitors, the latter monitor the satisfaction of those parts of the formula for which their local observations are sufficient. In case a local monitor is unable to decide upon a subformula, the corresponding part is rewritten and sent to other local monitors for further monitoring. While the proposed approach builds upon assumptions that make it inapplicable in this thesis’ context (e.g. synchronous system bus, monitoring of future time LTL formulas), the work presented in Section 3.3 was indeed inspired by the work of Bauer et al.

Summary. While there exist several works that aim for the decentralized enforcement of policies, all of them impose one or more of the following shortcomings that are not present in this thesis’ approach: (1) For many approaches the decision process

is not entirely distributed, as they effectively rely on certain central components [39, 63, 64, 115]. (2) Others do not consider that protected resources (i.e. data) keep being propagated throughout the entire distributed system and are thus only able to protect static resources [6, 28, 31, 63, 64]. (3) Some works only consider a posteriori detective enforcement of policy violations rather than preventive enforcement [12, 13, 18]. (4) Lastly, one of the discussed approaches can not be deployed within commodity systems/networks as it relies on a particular system bus [18].

6.4 Orthogonal Approaches to Distributed Usage Control

This section discusses related works on distributed data usage control that are orthogonal to this thesis' results. Hence, many of the discussed works could be integrated with the models and infrastructure proposed in this thesis. Alternatively, some of the cited works propose alternative approaches for problems also tackled in this thesis.

xDUCON [181, 182] is a framework that aims at the enforcement of Data Sharing Agreements (DSAs) across collaborating organizations. The framework builds upon the Shared Data Space (SDS) abstraction—some storage that is shared by the collaborating entities. Before exchanging any sensitive resources, the organizations' natural-language DSAs are translated into technical representations, called xDPolicies (Cross Domain Policies), which are stored within the SDS. Besides xDPolicies, also Subject tuples and Target tuples are shared using the SDS. Hereby, Subject tuples refer to the organizations' entities that might perform actions on resources, while Target tuples represent those resources. xDPolicies are evaluated whenever a subject is about to disseminate or access some resource: Once the PEP observes such an action, the PDP evaluates the PEP's decision request in correspondence with all applicable xDPolicies obtained from the SDS. Since xDUCON also supports the evaluation of ongoing conditions, it is well suited to revoke access rights while long-lived accesses are in progress. As xDUCON has been instantiated to loosely-connected networks in which mobile devices build up SDSs in an ad-hoc manner, the authors conclude that the framework "is well suited to the needs of a decentralized dissemination and enforcement of policies for controlling the accesses to data" [182]. However, it remains unclear to which extent data usage can be controlled once a subject got access to some sensitive resource, i.e. whether an accessing subject might create further copies of the resource and to which extent those copies are protected. In addition, xDUCON does not seem to enforce policies requiring coordination between multiple entities as described in Section 3.3.

Chen et al. [32, 33] present DataSafe, a solution to prevent "illegitimate secondary dissemination of protected plaintext data by authorized recipients". Similar to this thesis' approach, DataSafe allows unmodified applications to transparently use protected data while ensuring that according usage policies will not be violated. For this, DataSafe builds upon additional hardware as well as a trusted hypervisor, both of which must be deployed on all devices that are expected to access and use protected data. In terms of

data flow tracking, DataSafe makes use of hardware tags that are propagated whenever data flow related hardware instructions are executed. Yet, it remains unclear how data flow tracking can be maintained in case additional processors, such as graphics processing units, are used for data processing. Instead, performing data flow tracking at higher abstraction layers, such as the operating system as proposed in this thesis, does not come with such problems. In addition, DataSafe is not able to enforce obligations, e.g. in order to enforce the deletion of some sensitive data.

Stihler et al. [194] present a usage control architecture for business coalitions. In their scenario, two businesses collaborate via web services. Hence, one of the two collaborating businesses is called service provider, while the other is called service consumer. The goal of this work is to constrain the usage of the provided web service for users from within the service consumer business. For this, an independent broker entity is introduced which provides a web service lookup functionality, a notification service, a policy repository, and methods for policy provisioning. Before users of the service consumer might use a provided web service, an administrator from within the service consumer's business must write usage policies. Those policies are sent to the broker and then forwarded to the service provider. Whenever a user accesses the web service, the request is mediated through a PEP which will contact a local PDP for policy evaluation. In case of a policy violation, the broker's notification service is leveraged to inform the corresponding user about the policy violation. If the user does not take any compensating actions within a predefined amount of time, the session will idle and an administrator is notified. Notably, the proposed solution also allows for the ongoing evaluation of policies: If relevant attributes of users, and/or the environment change while a web service usage is in progress, a policy re-evaluation is triggered. This might lead to a policy violation and consequently to a corresponding notification being sent to the user (cf. above). This work is orthogonal to what has been presented in this thesis, as the solution is specifically tailored to web services. As such, it does neither cater to the data dimension (Section 3.2), nor to the fact that there might be multiple PEPs and/or PDPs that might necessitate coordination (Section 3.3).

SafeShare [203, 204] introduces so-called self-controlling objects (SafeShare objects)—encrypted containers which contain the sensitive data to be protected as well as metadata about that data and allowed operations on it, i.e. policies. The approach is embedded into a cloud environment in which data owners provide their SafeShare objects via a cloud-based data sharing service. Once a data consumer downloads a SafeShare object and tries to access it, at first it is evaluated whether the requestor was granted access rights by the data owner. Such access rights for individual data consumers are embedded within the SafeShare object a priori. Once a rightful data consumer opens the SafeShare object, an additional background process is started which continuously monitors the data consumer's compliance with the granted permissions.

Unfortunately, no further details are provided on how this continuous monitoring as well as policy enforcement is technically achieved.

6.5 Securing Data Usage Control Infrastructures

By discussing different approaches that have been tailored to secure data usage control infrastructures, this section complements the security evaluation in Section 5.1. The solutions presented in the following could be integrated with this thesis' infrastructure in order to protect its security.

Zhang et al. [227] build a usage control reference monitor on the basis of the Security-Enhanced Linux kernel security module (SELinux). The main issue being tackled is that such reference monitors usually reside outside of the domain of the policy stakeholder, which, however, wants some assurance on the enforcement of the policy. The paper thus defines and implements a trusted usage control subsystem with a minimal trusted code base, the integrity of which is protected using a TPM (Trusted Platform Module) and a Core Root of Trust Measurement (CRTM). Upon boot time, the platform's components (i.e. BIOS, boot loader, OS kernel) are measured and the corresponding values are stored to the TPM's Platform Configuration Registers (PCR). Once the kernel is booted, an integrity measurement service takes over and measures runtime components such as the usage control infrastructure as well as configuration files, data usage policies, client applications, etc. The measured values are then checked against known good values. If the measured values are not known to be good, then the reference monitor could be configured to shut down the corresponding applications or to disallow access to protected data.

A solution to the secure enforcement of usage control policies within service-oriented architectures is presented in [3]. Whenever an authenticated entity requests a protected resource via a web service, a security gateway attaches data usage control policies to the data which ought to be enforced by the resource requestor. However, before releasing the data and its policy, the resource provider (i.e. the web service) performs a remote attestation of the resource requestor's platform with the help of the TPM module. As for the solution above, the corresponding software stack is measured. The measured values are then sent to the resource provider which will compare them against a set of known good values. Only if this check yields a positive result, then the requested resource and the attached policy are released.

UCLinux [111] is a Linux security module [215] that specializes in usage control enforcement and supports attestation and sealing using the TPM. In a nutshell, UCLinux stores usage controlled files within an encrypted file system. The corresponding data usage policies are stored as file metadata, while the key for decryption of the encrypted file system is sealed to a trusted configuration using the TPM. This ensures that the file system decryption key is only accessible if the system's current software stack, which is

measured similarly to the approaches above, is in a valid state. Upon distribution of usage controlled files, UCLinux uses a modified version of the TLS protocol in order to perform attestation of the remote platform that is about to receive the files. For this, the remote platform essentially sends its signed list of PCR values which will then be checked against known trustworthy configurations. Building upon UCLinux and leveraging its capabilities for encrypted storage and remote attestation, further work [47] enables the secure posting and retrieval of protected documents via web servers.

6.6 Digital Rights Management (DRM)

Digital Rights Management (DRM) technology [45, 82, 85, 124, 193, 196, 218] primarily aims at the persistent protection of copyrighted digital content such as images, video, audio, book, games, text, or any combination thereof. In a nutshell, the digital media created by creators is packaged by producers into formats that are suitable for easy distribution as well as consumption by end users. Depending on the concrete DRM system, the packaging of the digital content may also involve its (partial) encryption and adding of digital watermarks. Further parties such as distributors (e.g. online shops, web retailers) may be involved within the DRM process. Usually, the packaged content is enriched with mechanisms for access control, copy protection as well as mechanisms for the management of usage and payment (i.e. usage policies or digital licenses). Thereby usage rules may be defined by a multitude of criteria such as access frequency, access expiration, pay-per-use, product quality, etc. Among others, central requirements for DRM systems are ease of use for all involved parties and robustness to circumvention.

Notably, distribution of the content and its corresponding policies/licenses are not necessarily intertwined. However, in any case the central idea is that the digital content can only be used (in an unrestricted manner) after a license has been obtained, e.g. from a central license server. In case the packaged content is encrypted, providing such a license will usually trigger decryption of the content and consequently allow for its usage in accordance with the obtained license. In fact, separation of the digital content from licenses makes the system more flexible by allowing for the free distribution of the (often encrypted) digital content across consumers. Once a license has been obtained, client-side applications are responsible for monitoring the compliance with granted usage rights (e.g. viewing, listening, copying, saving, modifying, distribution of the content) as well as with payment obligations. Such applications may come in the form of dedicated applications, such as multimedia players or PDF document viewers, or in the form of plugins for existing applications.

One major drawback of DRM is that it lacks any standards [65]. Different DRM solutions by different providers use different proprietary approaches and file types [1, 138] which hinder interoperability between the solutions of different producers. In other words, the content of one provider can usually only be accessed using this

provider's applications. Often such a lack of interoperability also leads to lock-in effects and dependency on these providers. Moreover, the corresponding client applications are often enough only available for particular platforms. Consequently, DRM solutions present usability obstacles that make users reluctant to adopt these technologies.

While some research addressed the lack of interoperability between DRM solutions [75, 86, 87, 101, 136, 197], in recent years the HTML5 standard [213] aimed at making web-based DRM more interoperable by replacing de facto DRM standards such as Adobe Flash and Microsoft Silverlight. For this HTML5 introduces Encrypted Media Extensions, primarily developed with the goal to provide interoperable streaming of digitally protected videos within web browsers. Despite many disputes around the technology [22, 148, 159] major web browser vendors (Apple [163], Google [69], Microsoft [141], Mozilla [148, 149]) have already adopted the technology and only the future will show whether it will also be adopted by end users.

In comparison with data usage control, DRM may be considered as one of its predecessors: In 2000 Iannella [83, 84] pointed out that traditional DRM (as discussed above) primarily focuses on the *protection* of content rather than on the corresponding *rights management* issue. In other words, he emphasizes traditional DRM methods are no longer sufficient in an environment in which digital media is not only consumed but also re-used, combined and extended. Hence Iannella proposed the *Open Digital Rights Language* [83] and moving towards more interoperable *Open Digital Rights Management* [84], therewith also influencing original usage control work [165].



Conclusions, Discussion and Future Work

After providing conclusions and summarizing this thesis' results in Section 7.1, Section 7.2 critically discusses this thesis' results, both from a technical and an ethical perspective. Finally, Section 7.3 discusses limitations and points to future work.

7.1 Conclusions

In recent years, many works addressed the enforcement of data usage control policies within individual systems. What has not been investigated in depth, however, is how usage control requirements can be enforced within complex distributed systems. In fact, many data usage policies do not only address single individual systems but are rather of a global scale, as they refer to data and data usage events that are distributed across multiple systems. While such policies can trivially be enforced by a centralized infrastructure, such a solution also introduces several drawbacks such as posing a single point of failure, expected heavy communication and performance overheads, privacy concerns, as well as the necessity for the central component to be always available. Hence, this thesis investigated three related research questions, namely “How can the flow of data across different systems be tracked in a generic and transparent manner and how can data usage policies be propagated to the corresponding decision points?” (RQ1), “How can usage control policies be enforced in an effective, preventive, and decentralized manner if data, system events, and policies are distributed across different independent systems?” (RQ2), and “Which guarantees for distributed usage control enforcement are provided by the presented infrastructure and what are critical attack vectors that necessitate further investigation?” (RQ3).

In order to address these research questions, this thesis built upon a policy specification language from the literature as well as corresponding models and enforcement infrastructures (Chapter 2). These models represent system runs as event traces and

integrate with data flow tracking technology in order to capture which usage controlled data takes which concrete representations at runtime. Policies then allow to constrain the legitimacy of certain event traces and system states.

On the basis of this model, this thesis contributed by providing a comprehensive and generic model for distributed data usage control (Section 3.1). This allows to model the execution of multiple concurrent event traces of different systems. In addition, it was shown how these individual observations can be reassembled in order to mimic the behavior of classical monolithic systems in which only one event trace can be observed.

In terms of **RQ1**, this thesis provided the first model (Section 3.2) and implementation (Section 4.2) to track data flows and data usage control policies across systems in a manner that is independent of particular file types, applications, application-protocols, hypervisors, or hardware. Technically, this tracking is performed in a decentralized manner on the basis of intercepting networking related system calls. As such, no modifications to applications or the operating system are required. The evaluation (Section 5.2) showed that the performance overhead imposed by such cross-system data flow tracking is hardly measurable and usually negligible.

In terms of **RQ2**, this thesis provided the first model (Section 3.3) and implementation (4.3) that allow for the fully decentralized, efficient, and preventive enforcement of global data usage control policies, i.e. policies the data and events of which are distributed across different systems. This approach includes the identification of systems that are potentially relevant to enforce a given policy, as well as the identification of situations in which coordination of policy decisions can be safely omitted. The thesis proves that the performed optimizations are correct (Appendices A and B). The evaluation compared the implemented distributed infrastructure with a traditional centralized approach (Section 5.3). The results revealed that the decentralized approach is capable of outperforming the centralized approach in most cases.

In terms of **RQ3**, this thesis evaluated the proposed solution in terms of security (Section 5.1). The security analysis revealed assumptions and limitations of the proposed models and the implemented infrastructure. Whenever appropriate, possible solutions to the identified weaknesses were proposed.

7.2 Critical Reflection

While performing this thesis work, several technical, practical and ethical questions and discussions arose. The most important ones are described in the following.

Specification and Deployment of Data Usage Policies. A central question when designing usage control solutions is which entity should be allowed to specify and deploy data usage policies for which data. This question is even more daunting when it comes to distributed data usage scenarios, since the sensitive data being handled and protected in a distributed manner is likely to be the property of different entities or organizations. For example, when considering the insurance company's contracts,

both the client and the insurance company are likely to demand protection of parts of the contract. Possibly, the corresponding data protection requirements concern the same parts of the contract and may be in conflict. In addition, other data protection requirements, such as laws, might need to be considered. While such problems are out of the scope of this work and rather related to works such as [106], they are the basis for further critical considerations.

For example, if governments or other powerful organizations would have the possibility to specify and deploy policies for any kind of usage controlled data, then the usage control infrastructure could be (mis-)used as a censorship framework. For example, those powerful organizations could then demand deletion of any kind of data or impose arbitrary obligations upon data usage. These and similar attacks could also be considered as Denial of Service (DoS) attacks, since they are capable of making usage controlled data unavailable by specifying and deploying corresponding policies.

Control over the Infrastructure. A related question is who is in control over the distributed data usage control enforcement infrastructure. While this thesis suggests that each entity, such as end users, businesses or organizations, sets up its own usage control infrastructure, the posed question is more complex when it comes to real-world distributed deployment scenarios. The underlying problem is that at some point there must exist a trust anchor, be it a trusted application, a trusted operating system, or trusted hardware upon which the usage control infrastructure is deployed. However, this trust anchor must not only be trusted by the entity which deploys the infrastructure, but also by entities which deploy a remote infrastructure which will later communicate with the former. Otherwise, i.e. if the trust anchor is not trusted by remote entities, the corresponding infrastructures ought not exchange any sensitive information such as usage controlled data, usage control policies, data flow states, or information about policy decisions. However, whether such a universal trust anchor exists is questionable. If, however, such a trust anchor would exist, then the entity providing this anchor would essentially be in control over the entire usage control infrastructure.

Even assuming the existence of a universally trusted trust anchor, the question remains how the availability of the usage controlled data can be assured. Since the data would only be persistently stored in an encrypted manner in order to maintain its confidentiality, it would need to be assured that the corresponding decryption key is never lost. This in turn, would necessitate redundant and secure storage of the corresponding decryption keys, effectively leading to a complex bootstrapping problem.

In the light of this discussion, it is noteworthy that end users do in many cases no longer possess administrative privileges for many of today's and tomorrow's omnipresent computing systems such as mobile phones, tablet computers and embedded (cyber-physical) systems ((autonomous) cars, airplanes, robots, wearable devices, smart houses/cities, etc.). At a first glance this fact might seem to solve the 'trust anchor problem', since it could keep users from disabling or circumventing the data usage control infrastructure. However, it stands to reason that many end users would

consider such devices and a corresponding data usage control infrastructure as paternalism which they are unwilling to accept in practice. In the end, only real-world deployment scenarios will show how such technology is adopted by end users. Very likely, any such adoption process is to be accompanied by many more ethical, societal and political discussions.

Resume. Considering the above discussions, it is questionable whether a data usage control infrastructure as developed within this thesis and related works is indeed desirable in completely open environments such as the world wide web. E.g., Digital Rights Management (DRM), a technology aiming at the protection of payment-based information such as audio, video, and software, faced much criticism and was finally dropped by all major audio labels [135]. While the scope and application areas of data usage control technology are broader than those of DRM technology, the above example shows that deployment of such technology is not straight forward and that user acceptance plays a critical role. Nevertheless, data usage control technology could be deployed and adopted in business scenarios in which sensitive data is critical and in which certain trust anchors are present, e.g. because employees are provided with ready-to-use computing systems without administrative privileges.

7.3 Limitations and Future Work

Policy Specification and Deployment. This thesis does not cover aspects related to the specification, translation, derivation, and evolution of policies. These and related topics are partially covered in [79, 106, 108, 109]. However, none of these works does explicitly cater to the particular requirements of policy specification and deployment in complex distributed environments, e.g. addressing the problem which entities might specify and deploy policies for which data. Another related, and yet unexplored, problem is which entities, if any, should be capable of modifying policies for data that has already been disseminated. Very likely, such research would not only involve technical solutions, but also ethical, societal and political discussions.

Integration with Usage Control Solutions for Single Systems. This thesis focused on the distributed aspects of data usage control. As such, problems related to data usage control on single independent systems are not covered as they are addressed elsewhere [55, 74, 110, 119, 125, 126, 168, 170, 174, 188, 214, 216]. It remains to mention that all of the cited works are compatible with this thesis' work both from a conceptual and a technical point of view. Hence, it remains to build an encompassing infrastructure that integrates all of the above works with this thesis' approach, thus allowing for comprehensive system-wide and cross-system data usage control enforcement.

Improving the Granularity of Cross-System Data Flow Tracking. This work tracks cross-system data flows on the basis of system call interpositioning. While this approach is generic by supporting any kind of user space application, it comes at the cost of data

flow tracking overapproximations since processes, files, memory regions, sockets, pipes, etc. are treated as black boxes the contents of which can not be further distinguished. The cross-system data flow tracking approach presented in this thesis would thus benefit from an integration with application-layer usage control monitors (e.g. for Firefox [110], Thunderbird [125], Microsoft Office [188]). Since these monitors would be able to provide additional information about data flows within the corresponding applications, the granularity of cross-system data flow tracking could be improved.

Cross-System Data Flow Tracking Beyond TCP. Although this thesis' concepts and implementation support most applications by realizing cross-system data flow tracking at the level of TCP, other important applications such as multimedia streaming and Voice over IP are not covered. This is because such applications use UDP as transport layer, as low-latency delivery is preferred to the guarantees provided by TCP. Future work is thus required in order to extend cross-system data flow tracking to UDP or even other transport layers. Further, extending cross-system data flow tracking to raw sockets would completely decouple cross-system data flow tracking from the transport layer protocol and thus support any kind of IP-based communication.

Specifying Efficiently Enforceable ECA Rules. As researched in [106], there exist multiple ways in which high-level usage control specification policies can be translated into technical low-level ECA rules as enforced by this thesis' infrastructure. As shown in this thesis' evaluation, however, both the ECA rules' trigger events and conditions have a significant influence on the efficiency of the policy evaluation. Hence, future work is needed that aims at the specification of efficiently enforceable low-level ECA rules, possibly on the basis of given use case scenarios, i.e. under consideration of expected event traces.

Dedicated Technical Solutions for Distributed Policy Decisions. In order to ease implementation, this thesis made use of the off-the-shelf distributed database Cassandra in order to coordinate policy decisions across multiple PDPs. However, as the evaluation showed, Cassandra introduces additional communication overheads even in case no coordination between PDPs is required. Hence it stands to reason that a tailored implementation, specifically developed with usage control requirements in mind, would improve upon these overheads and operate with less communication and performance overhead.

Evaluation. This thesis' evaluation shows that the proposed distributed concepts are able to compete with traditional centralized infrastructures and that the imposed performance and communication overheads are acceptable in many cases. Often, the decentralized approach is even able to outperform a centralized approach. However, the present evaluation only gives a first insight into the performance of such distributed usage control infrastructures. In order to understand their full impact, rigorous evaluations using different and more complex policies as well as larger distributed system setups would need to be performed. Ideally, such evaluations should be

performed within real-world production environments and consider real-world use cases and event traces. This way, it would also be possible to perform usability studies with the goal to understand whether end users accept the imposed performance overheads as well as the way they are constrained when working with usage controlled data.

Security and Trustworthiness of the Usage Control Infrastructure. One limitation of this thesis' implementation is that communication between remote usage control infrastructures is only rudimentary secured using TLS. While TLS provides confidentiality and integrity of messages as well as both client and server authentication mechanisms, it does not solve the problem that the underlying infrastructure, such as the hardware and the operating systems, as well as the certificate authorities must be trustworthy. While the assurance of the hardware's and operating system's trustworthiness is an issue on its own (cf. below), valid certificates signed by trusted certificate authorities are at least able to provide a minimal amount of accountability by being able to know with which other entities some sensitive information was shared.

In order to further secure the distributed usage control infrastructure and to defend against stronger attackers than non-privileged end users, additional security mechanisms would need to be deployed. For this, the trust anchor would need to be embedded at a lower layer than the operating system, e.g. by using technologies such as TPMs [208] or SmartCards [76]. Corresponding solutions are described in Section 6.5.

Side Channels and Media Breaks. The solutions provided in this thesis do not cover side channel attacks such as timing attacks or power monitoring, which could be used to infer usage controlled data and consequently to use it in an unrestricted manner.

Further, media breaks are out of the scope of this work. Once data is legitimately displayed on the screen or printed as a hard copy, no further protection of the corresponding data can be guaranteed.

Bibliography

- [1] Adobe Systems Incorporated. *Adobe Content Server*. 2015. URL: <http://www.adobe.com/solutions/ebook/content-server.html> (visited on 09/08/2015) (cited on pages 17, 19, 172).
- [2] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha Nabar, Tomoe Sugihara, and Jennifer Widom. “Trio: A System for Data, Uncertainty, and Lineage”. In: *Proceedings of the 32nd International Conference on Very Large Data Bases. VLDB '06*. Seoul, Korea: VLDB Endowment, 2006, pages 1151–1154. URL: <http://dl.acm.org/citation.cfm?id=1182635.1164231> (cited on page 159).
- [3] Berthold Agraier, Muhammad Alam, Ruth Breu, Michael Hafner, Alexander Pretschner, Jean-Pierre Seifert, and Xinwen Zhang. “A Technical Architecture for Enforcing Usage Control Requirements in Service-oriented Architectures”. In: *Proceedings of the 2007 ACM Workshop on Secure Web Services. SWS '07*. Fairfax, Virginia, USA: ACM, 2007, pages 18–25. ISBN: 9781595938923. DOI: 10.1145/1314418.1314422 (cited on page 171).
- [4] M. David Allen, Adriane Chapman, Barbara Blaustein, and Len Seligman. “Capturing Provenance in the Wild”. In: *Provenance and Annotation of Data and Processes*. Edited by Deborah L. McGuinness, James R. Michaelis, and Luc Moreau. Volume 6378. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pages 98–101. ISBN: 9783642178184. DOI: 10.1007/978-3-642-17819-1_12 (cited on page 164).
- [5] Ali Mousa G. Alzahrani. “Efficient Enforcement of Security Policies in Distributed Systems”. PhD thesis. De Montfort University, Leicester, UK, Apr. 2013. URL: <http://hdl.handle.net/2086/9029> (cited on pages 18, 167).
- [6] Ali Alzahrani, Helge Janicke, and Sarshad Abubaker. “Decentralized XACML Overlay Network”. In: *IEEE 10th International Conference on Computer and Information Technology (CIT)*. June 2010, pages 1032–1037. DOI: 10.1109/CIT.2010.189 (cited on pages 18, 19, 167–169).
- [7] Claudio A. Ardagna, Laurent Bussard, Sabrina De Capitani di Vimercati, Gregory Neven, Stefano Paraboschi, Eros Pedrini, Franz-Stefan Preiss, Dave Raggett, Pierangela Samarati, Slim Trabelsi, and Mario Verdicchio. “PrimeLife Policy Language”. In: *W3C Workshop on Access Control Application Scenarios*. Nov. 2009, pages 1–6. ISBN: 9788897253006 (cited on page 156).

- [8] Francois Audet and Cullen Jennings. *RFC 4787: Network Address Translation (NAT) Behavioral Requirements for Unicast UDP*. 2007. URL: <https://tools.ietf.org/html/rfc4787> (cited on page 54).
- [9] Autodesk, Inc. *Autodesk Licensing*. 2010. URL: http://docs.autodesk.com/ACD/2011/ENU/pdfs/adsk_lic.pdf (visited on 09/02/2015) (cited on page 17).
- [10] Mihai Bădoiu, Kiran-Kumar Muniswam-Reddy, Anastasios Sidiropoulos, and Mythili Vutukuru. *A Distributed Provenance Aware Storage System*. Technical report. MIT, 2005 (cited on page 164).
- [11] Peter Bailis and Ali Ghodsi. “Eventual Consistency Today: Limitations, Extensions, and Beyond”. In: *Communications of the ACM* 56.5 (May 2013), pages 55–63. ISSN: 0001-0782. DOI: 10.1145/2447976.2447992 (cited on page 88).
- [12] David Basin, Germano Caronni, Sarah Ereth, Felix Harvan Matúšand Klaedtke, and Heiko Mantel. “Scalable Offline Monitoring”. In: *Runtime Verification*. Edited by Borzoo Bonakdarpour and ScottA. Smolka. Volume 8734. Lecture Notes in Computer Science. Springer International Publishing, 2014, pages 31–47. ISBN: 9783319111636. DOI: 10.1007/978-3-319-11164-3_4 (cited on pages 18, 19, 166, 169).
- [13] David Basin, Felix Harvan Matúšand Klaedtke, and Eugen Zălinescu. “Monitoring Data Usage in Distributed Systems”. In: *Software Engineering, IEEE Transactions on* 39.10 (Oct. 2013), pages 1403–1426. ISSN: 0098-5589. DOI: 10.1109/TSE.2013.18 (cited on pages 16, 18, 19, 166, 169).
- [14] David Basin, Felix Harvan Matúšand Klaedtke, and Eugen Zălinescu. “Monitoring Usage-Control Policies in Distributed Systems”. In: *18th International Symposium on Temporal Representation and Reasoning (TIME)*. Sept. 2011, pages 88–95. DOI: 10.1109/TIME.2011.14 (cited on page 166).
- [15] David Basin, Felix Klaedtke, and Samuel Müller. “Monitoring Security Policies with Metric First-order Temporal Logic”. In: *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies. SACMAT '10*. Pittsburgh, Pennsylvania, USA: ACM, 2010, pages 23–34. ISBN: 9781450300490. DOI: 10.1145/1809842.1809849 (cited on page 156).
- [16] David Basin, Felix Klaedtke, and Samuel Müller. “Policy Monitoring in First-Order Temporal Logic”. In: *Computer Aided Verification*. Edited by Tayssir Touili, Byron Cook, and Paul Jackson. Volume 6174. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pages 1–18. ISBN: 9783642142949. DOI: 10.1007/978-3-642-14295-6_1 (cited on page 156).

- [17] David Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. “Monitoring Metric First-Order Temporal Properties”. In: *Journal of the ACM* 62.2 (May 2015), 15:1–15:45. ISSN: 0004-5411. DOI: 10.1145/2699444 (cited on pages 18, 50, 166).
- [18] Andreas Bauer and Yliès Falcone. “Decentralised LTL Monitoring”. In: *FM 2012: Formal Methods*. Edited by Dimitra Giannakopoulou and Dominique Méry. Volume 7436. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pages 85–100. ISBN: 9783642327582. DOI: 10.1007/978-3-642-32759-9_10 (cited on pages 18, 19, 168, 169).
- [19] Michele Bezzi and Slim Trabelsi. “Data Usage Control in the Future Internet Cloud”. In: *The Future Internet*. Volume 6656. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pages 223–231. ISBN: 9783642208973. DOI: 10.1007/978-3-642-20898-0_16 (cited on page 162).
- [20] Andrea Bittau, Michael Hamburg, Mark Handley, David Mazières, and Dan Boneh. “The Case for Ubiquitous Transport-level Encryption”. In: *Proceedings of the 19th USENIX Conference on Security*. USENIX Security’10. Washington, DC: USENIX Association, 2010. URL: <http://dl.acm.org/citation.cfm?id=1929820.1929855> (cited on page 99).
- [21] Eric A. Brewer. “Towards Robust Distributed Systems”. In: *Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing*. Keynote. 2000. ISBN: 1581131836. DOI: 10.1145/343477.343502 (cited on page 88).
- [22] Peter Bright. *DRM in HTML5 is a victory for the open Web, not a defeat*. 2013. URL: <http://arstechnica.com/business/2013/05/drm-in-html5-is-a-victory-for-the-open-web-not-a-defeat/> (visited on 09/04/2015) (cited on page 173).
- [23] Peter Buneman, Adriane Chapman, and James Cheney. “Provenance Management in Curated Databases”. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’06. Chicago, IL, USA: ACM, 2006, pages 539–550. ISBN: 1595934340. DOI: 10.1145/1142473.1142534 (cited on page 159).
- [24] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. “Why and Where: A Characterization of Data Provenance”. In: *Database Theory – ICDT 2001*. Edited by Jan Van den Bussche and Victor Vianu. Volume 1973. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pages 316–330. ISBN: 9783540414568. DOI: 10.1007/3-540-44503-X_20 (cited on page 159).
- [25] Peter Buneman and Wang-Chiew Tan. “Provenance in Databases”. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’07. Beijing, China: ACM, 2007, pages 1171–1173. ISBN: 9781595936868. DOI: 10.1145/1247480.1247646 (cited on page 159).

- [26] Laurent Bussard, Gregory Neven, and Franz-Stefan Preiss. “Downstream Usage Control”. In: *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*. July 2010, pages 22–29. DOI: 10.1109/POLICY.2010.17 (cited on page 156).
- [27] Bryan Cantrill, Adam Leventhal, Mike Shapiro, and Brendan Gregg. *dtrace.org: About DTrace*. 2015. URL: <http://dtrace.org/blogs/about/> (visited on 09/02/2015) (cited on page 158).
- [28] David Chadwick. “Coordinated Decision Making in Distributed Applications”. In: *Information Security Tech. Report 12.3* (June 2007), pages 147–154. ISSN: 1363-4127. DOI: 10.1016/j.istr.2007.05.003 (cited on pages 18, 19, 168, 169).
- [29] David W. Chadwick and Stijn F. Lievens. “Enforcing “Sticky” Security Policies Throughout a Distributed Application”. In: *Proceedings of the 2008 Workshop on Middleware Security*. MidSec ’08. Leuven, Belgium: ACM, 2008, pages 1–6. ISBN: 9781605583631. DOI: 10.1145/1463342.1463343 (cited on page 160).
- [30] David W. Chadwick, Linying Su, and Romain Laborde. “Coordinating Access Control in Grid Services”. In: *Concurrency and Computation: Practice & Experience - Middleware for Grid Computing: Future Trends 20.9* (June 2008), pages 1071–1094. ISSN: 1532-0626. DOI: 10.1002/cpe.v20:9 (cited on pages 18, 168).
- [31] David W. Chadwick, Linying Su, Oleksandr Otenko, and Romain Laborde. “Coordination between Distributed PDPs”. In: *7th IEEE International Workshop on Policies for Distributed Systems and Networks*. 2006. DOI: 10.1109/POLICY.2006.14 (cited on pages 18, 19, 35, 168, 169).
- [32] Yu-Yuan Chen, Pramod A. Jamkhedkar, and Ruby B. Lee. “A Software-Hardware Architecture for Self-Protecting Data”. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS ’12. Raleigh, North Carolina, USA: ACM, Oct. 2012, pages 14–27. ISBN: 9781450316514. DOI: 10.1145/2382196.2382201 (cited on page 169).
- [33] Yu-Yuan Chen and Ruby B. Lee. “Hardware-Assisted Application-Level Access Control”. In: *Proceedings of the 12th International Conference on Information Security*. ISC ’09. Pisa, Italy: Springer-Verlag, 2009, pages 363–378. ISBN: 9783642044731. DOI: 10.1007/978-3-642-04474-8_29 (cited on page 169).
- [34] Georgios Chinis, Polyvios Pratikakis, Sotiris Ioannidis, and Elias Athanasopoulos. “Practical Information Flow for Legacy Web Applications”. In: *Proceedings of the 8th Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems*. ICPOOLPS’13. Montpellier, France: ACM, 2013, pages 17–28. ISBN: 9781450320450. DOI: 10.1145/2491404.2491410 (cited on page 159).

- [35] Jan Chomicki. “Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding”. In: *ACM Transactions on Database Systems* 20.2 (June 1995), pages 149–186. ISSN: 0362-5915. DOI: 10.1145/210197.210200 (cited on page 156).
- [36] CipherShed project. *CipherShed — Secure Encryption Software*. 2015. URL: <https://ciphershed.org/> (visited on 09/02/2015) (cited on page 96).
- [37] Maurizio Colombo, Fabio Martinelli, Paolo Mori, and Aliaksandr Lazouski. “On Usage Control for GRID Services”. In: *International Joint Conference on Computational Sciences and Optimization*. Volume 1. Apr. 2009, pages 47–51. DOI: 10.1109/CSO.2009.479 (cited on page 157).
- [38] Contributors to the cURL project. *curl and libcurl*. 2015. URL: <http://curl.haxx.se/> (visited on 09/02/2015) (cited on page 105).
- [39] Graham Cormode. “The Continuous Distributed Monitoring Model”. In: *SIGMOD Rec.* 42.1 (May 2013), pages 5–14. ISSN: 0163-5808. DOI: 10.1145/2481528.2481530 (cited on pages 18, 167, 169).
- [40] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. “The Ponder Policy Specification Language”. English. In: *Policies for Distributed Systems and Networks*. Edited by Morris Sloman, EmilC. Lupu, and Jorge Lobo. Volume 1995. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pages 18–38. ISBN: 9783540416104. DOI: 10.1007/3-540-44569-2_2. URL: http://dx.doi.org/10.1007/3-540-44569-2_2 (cited on page 156).
- [41] DataStax, Inc. *DataStax - NoSQL Cassandra Database — Fastest, Most Scalable*. 2015. URL: <http://www.datastax.com/> (visited on 09/02/2015) (cited on page 87).
- [42] DataStax, Inc. *Lightweight transactions in Cassandra 2.0 : DataStax*. 2015. URL: <http://www.datastax.com/dev/blog/lightweight-transactions-in-cassandra-2-0> (visited on 09/02/2015) (cited on page 88).
- [43] DataStax, Inc. *Timeuuid functions — DataStax CQL 3.0 Documentation*. 2015. URL: http://docs.datastax.com/en/cql/3.0/cql/cql_reference/timeuuid_functions_r.html (visited on 09/02/2015) (cited on page 88).
- [44] Benjamin Davis and Hao Chen. “DBTaint: Cross-application Information Flow Tracking via Databases”. In: *Proceedings of the 2010 USENIX Conference on Web Application Development*. WebApps’10. Boston, MA: USENIX Association, 2010. URL: <http://dl.acm.org/citation.cfm?id=1863166.1863178> (cited on page 159).
- [45] Eric Diehl. “A Four-Layer Model for Security of Digital Rights Management”. In: *Proceedings of the 8th ACM Workshop on Digital Rights Management*. DRM ’08. Alexandria, Virginia, USA: ACM, Oct. 2008, pages 19–28. ISBN: 9781605582900. DOI: 10.1145/1456520.1456527 (cited on page 172).

- [46] Tim Dierks and Eric Rescorla. *RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2*. 2008. URL: <https://tools.ietf.org/html/rfc5246> (cited on pages 71, 96).
- [47] Peter Djalaliev and JoséCarlos Brustoloni. “Secure Web-Based Retrieval of Documents with Usage Controls”. In: *Proceedings of the 2009 ACM Symposium on Applied Computing*. SAC ’09. Honolulu, Hawaii: ACM, 2009, pages 2062–2069. ISBN: 9781605581668. DOI: 10.1145/1529282.1529738 (cited on page 172).
- [48] Ralph Droms. *RFC 2131: Dynamic Host Configuration Protocol*. 1997. URL: <https://tools.ietf.org/html/rfc2131> (cited on page 54).
- [49] David Durham, Jim Boyle, Ron Cohen, Raju Rajan, Shai Herzog, and Arun Sastry. *RFC 2748: The COPS (Common Open Policy Service) Protocol*. 2000. URL: <https://tools.ietf.org/html/rfc2748> (cited on pages 35, 72).
- [50] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, and Robert Morris. “Labels and Event Processes in the Asbestos Operating System”. In: *ACM SIGOPS Operating Systems Review* 39.5 (Oct. 2005), pages 17–30. ISSN: 0163-5980. DOI: 10.1145/1095809.1095813 (cited on pages 159, 163).
- [51] Paul England, John D. DeTreville, and Butler W. Lampson. “Digital Rights Management Operating System”. Patent US6330670 (US). Dec. 2001. URL: <http://www.google.com/patents/US6330670> (cited on page 97).
- [52] Chris Evans. *vsftpd - Secure, fast FTP server for UNIX-like systems*. 2015. URL: <https://security.appspot.com/vsftpd.html> (visited on 09/02/2015) (cited on page 105).
- [53] David Evans and David M. Eyers. “Efficient Policy Checking across Administrative Domains”. In: *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*. July 2010, pages 146–153. DOI: 10.1109/POLICY.2010.36 (cited on pages 17, 18, 163, 164).
- [54] David F. Ferraiolo, Janet A. Cugini, and D. Richard Kuhn. “Role-Based Access Control (RBAC): Features and Motivations”. In: *Proceedings of the 11th Annual Computer Security Application Conference*. New Orleans, LA, USA, 1995, pages 242–248 (cited on page 168).
- [55] Denis Feth and Alexander Pretschner. “Flexible Data-Driven Security for Android”. In: *IEEE Sixth International Conference on Software Security and Reliability*. SERE. June 2012, pages 41–50. DOI: 10.1109/SERE.2012.14 (cited on pages 16, 36, 43, 77, 178).

- [56] Michael J. Fischer. “The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)”. In: *Foundations of Computation Theory*. Edited by Marek Karpinski. Volume 158. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1983, pages 127–140. ISBN: 9783540126898. DOI: 10.1007/3-540-12689-9_99 (cited on page 88).
- [57] Ian Foster. “The Virtual Data Grid: A New Model and Architecture for Data-intensive Collaboration”. In: *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*. SSDBM ’03. Cambridge, MA: IEEE Computer Society, 2003. ISBN: 0769519644. DOI: 10.1109/SSDM.2003.1214945 (cited on page 159).
- [58] Ian Foster and Carl Kesselman. “Globus: A Metacomputing Infrastructure Toolkit”. In: *International Journal of High Performance Computing Applications* 11.2 (1997), pages 115–128 (cited on page 159).
- [59] Free Software Foundation, Inc. *GNU Wget*. 2015. URL: <http://www.gnu.org/software/wget/> (visited on 09/02/2015) (cited on page 105).
- [60] Alexander Fromm. *Data Protection in Composed Service Oriented Computing*. Technical report. Doctoral Thesis Proposal. Technische Universität München, 2014 (cited on page 98).
- [61] Alexander Fromm, Florian Kelbert, and Alexander Pretschner. “Data Protection in a Cloud-Enabled Smart Grid”. In: *Smart Grid Security*. Volume 7823. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pages 96–107. ISBN: 9783642380297. DOI: 10.1007/978-3-642-38030-3_7 (cited on pages 25, 36, 77, 158).
- [62] Dov Gabbay. “The Declarative Past and Imperative Future”. In: *Temporal Logic in Specification*. Edited by B. Banieqbal, H. Barringer, and A. Pnueli. Volume 398. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1989, pages 409–448. ISBN: 9783540518037. DOI: 10.1007/3-540-51803-7_36 (cited on page 32).
- [63] Richard Gay, Jinwei Hu, and Heiko Mantel. “CliSeAu: Securing Distributed Java Programs by Cooperative Dynamic Enforcement”. In: *Information Systems Security*. Edited by Atul Prakash and Rudrapatna Shyamasundar. Volume 8880. Lecture Notes in Computer Science. Springer International Publishing, 2014, pages 378–398. ISBN: 9783319138404. DOI: 10.1007/978-3-319-13841-1_21 (cited on pages 18, 36, 77, 165, 169).
- [64] Richard Gay, Heiko Mantel, and Barbara Sprick. “Service Automata”. In: *Formal Aspects of Security and Trust*. Volume 7140. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pages 148–163. ISBN: 9783642294198. DOI: 10.1007/978-3-642-29420-4_10 (cited on pages 18, 19, 165, 169).

- [65] David Geer. “Digital Rights Technology Sparks Interoperability Concerns”. In: *Computer* 37.12 (2004), pages 20–22. ISSN: 0018-9162. DOI: 10.1109/MC.2004.242 (cited on pages 19, 172).
- [66] Ashish Gehani and Dawood Tariq. “SPADE: Support for Provenance Auditing in Distributed Environments”. In: *Proceedings of the 13th International Middleware Conference*. Middleware ’12. Montreal, Quebec, Canada: Springer-Verlag New York, Inc., 2012, pages 101–120. ISBN: 9783642351693. URL: <http://dl.acm.org/citation.cfm?id=2442626.2442634> (cited on page 164).
- [67] Eleni Gessiou, Vasilis Pappas, Elias Athanasopoulos, Angelos D. Keromytis, and Sotiris Ioannidis. “Towards a Universal Data Provenance Framework Using Dynamic Instrumentation”. In: *Information Security and Privacy Research*. Volume 376. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2012, pages 103–114. ISBN: 9783642304354. DOI: 10.1007/978-3-642-30436-1_9 (cited on page 158).
- [68] Seth Gilbert and Nancy Lynch. “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services”. In: *SIGACT News* 33.2 (June 2002). ISSN: 0163-5700. DOI: 10.1145/564585.564601 (cited on page 88).
- [69] Google Inc. *Chrome 26 Beta: Template Element & Unprefixed CSS Transitions*. 2013. URL: <http://blog.chromium.org/2013/02/chrome-26-beta-template-element.html> (visited on 09/04/2015) (cited on page 173).
- [70] Valient Gough. *EncFS*. 2014. URL: <http://www.arg0.net/#!/encfs/c1awt> (visited on 09/02/2015) (cited on page 96).
- [71] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. “Lest We Remember: Cold-boot Attacks on Encryption Keys”. In: *Communications of the ACM* 52.5 (May 2009), pages 91–98. ISSN: 0001-0782. DOI: 10.1145/1506409.1506429 (cited on page 100).
- [72] Jiho Han and Deog-Kyoon Jeong. “A Practical Implementation of IEEE 1588-2008 Transparent Clock for Distributed Measurement and Control Systems”. In: *IEEE Transactions on Instrumentation and Measurement* 59.2 (Feb. 2010), pages 433–439. ISSN: 0018-9456. DOI: 10.1109/TIM.2009.2024371 (cited on page 89).
- [73] Weili Han and Chang Lei. “A Survey on Policy Languages in Network and Security Management”. In: *Computer Networks* 56.1 (2012), pages 477–489. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2011.09.014 (cited on page 156).
- [74] Matús Harvan and Alexander Pretschner. “State-Based Usage Control Enforcement with Data Flow Tracking using System Call Interposition”. In: *Third International Conference on Network and System Security*. NSS. Oct. 2009,

- pages 373–380. DOI: 10.1109/NSS.2009.51 (cited on pages 15–17, 27, 30, 31, 36, 43, 46, 77, 155, 178).
- [75] Gregory L. Heileman and Pramod A. Jamkhedkar. “DRM Interoperability Analysis from the Perspective of a Layered Framework”. In: *Proceedings of the 5th ACM Workshop on Digital Rights Management*. DRM ’05. Alexandria, VA, USA: ACM, Nov. 2005, pages 17–26. ISBN: 1595932305. DOI: 10.1145/1102546.1102551 (cited on page 173).
- [76] Mike Hendry. *Smart Card Security and Applications, Second Edition*. 2nd. Norwood, MA, USA: Artech House, Inc., 2001. ISBN: 1580531563 (cited on page 180).
- [77] Michael Henson and Stephen Taylor. “Memory Encryption: A Survey of Existing Techniques”. In: *ACM Computing Surveys (CSUR)* 46.4 (Mar. 2014), 53:1–53:26. ISSN: 0360-0300. DOI: 10.1145/2566673 (cited on page 100).
- [78] Manuel Hilty, David Basin, and Alexander Pretschner. “On Obligations”. In: *Computer Security — ESORICS 2005*. Edited by Sabrinade Capitani di Vimercati, Paul Syverson, and Dieter Gollmann. Volume 3679. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pages 98–117. ISBN: 9783540289630. DOI: 10.1007/11555827_7 (cited on page 155).
- [79] Manuel Hilty, Alexander Pretschner, David Basin, Christian Schaefer, and Thomas Walter. “A Policy Language for Distributed Usage Control”. In: *Computer Security – ESORICS 2007*. Volume 4734. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pages 531–546. ISBN: 9783540748342. DOI: 10.1007/978-3-540-74835-9_35 (cited on pages 15, 27, 32, 155, 178).
- [80] Manuel Hilty, Alexander Pretschner, David Basin, Christian Schaefer, and Thomas Walter. “Monitors for Usage Control”. In: *Trust Management*. Edited by Sandro Etalle and Stephen Marsh. Volume 238. IFIP International Federation for Information Processing. Springer US, 2007, pages 411–414. ISBN: 9780387736549. DOI: 10.1007/978-0-387-73655-6_29 (cited on page 32).
- [81] D. Richard Hipp. *SQLite Home Page*. 2015. URL: <https://www.sqlite.org/> (visited on 09/02/2015) (cited on page 158).
- [82] Renato Iannella. “Digital Rights Management (DRM) Architectures”. In: *D-Lib Magazine* 7.6 (2001). ISSN: 1082-9873. DOI: 10.1045/june2001-iannella (cited on pages 172, 173).
- [83] Renato Iannella. *Open Digital Rights Language (ODRL) Version 1.1*. 2002. URL: <http://www.w3.org/TR/odr1/> (visited on 09/04/2015) (cited on page 173).
- [84] Renato Iannella. *Open Digital Rights Management*. Technical report. A Position Paper for the W3C DRM Workshop. IPR Systems Pty Ltd, 2000 (cited on page 173).

- [85] Pramod A. Jamkhedkar and Gregory L. Heileman. “Digital Rights Management Architectures”. In: *Computers and Electrical Engineering* 35.2 (Mar. 2009), pages 376–394. ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2008.06.012 (cited on page 172).
- [86] Pramod A. Jamkhedkar and Gregory L. Heileman. “DRM as a Layered System”. In: *Proceedings of the 4th ACM Workshop on Digital Rights Management*. DRM ’04. Washington DC, USA: ACM, Oct. 2004, pages 11–21. ISBN: 1581139691. DOI: 10.1145/1029146.1029151 (cited on page 173).
- [87] Pramod A. Jamkhedkar, Gregory L. Heileman, and Chris C. Lamb. “An Interoperable Usage Management Framework”. In: *Proceedings of the tenth Annual ACM Workshop on Digital Rights Management*. DRM ’10. Chicago, Illinois, USA: ACM, Oct. 2010, pages 73–88. ISBN: 9781450300919. DOI: 10.1145/1866870.1866885 (cited on page 173).
- [88] Helge Janicke, Antonio Cau, François Siewe, and Hussein Zedan. “Concurrent Enforcement of Usage Control Policies”. In: *IEEE Workshop on Policies for Distributed Systems and Networks*. POLICY ’08. 2008, pages 111–118. DOI: 10.1109/POLICY.2008.44 (cited on pages 18, 50, 167).
- [89] Helge Janicke, Mohamed Sarrab, and Hamza Aldabbas. “Controlling Data Dissemination”. In: *Data Privacy Management and Autonomous Spontaneous Security*. Edited by Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Nora Cuppens-Bouahia, and Sabrina de Capitani di Vimercati. Volume 7122. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pages 303–309. ISBN: 9783642288784. DOI: 10.1007/978-3-642-28879-1_21 (cited on pages 17, 18, 163, 164).
- [90] Georgios Karopoulos, Paolo Mori, and Fabio Martinelli. “Usage Control in SIP-based Multimedia Delivery”. In: *Computers & Security* 39, Part B (2013), pages 406–418. ISSN: 0167-4048. DOI: 10.1016/j.cose.2013.09.005 (cited on page 157).
- [91] Basel Katt, Xinwen Zhang, Ruth Breu, Michael Hafner, and Jean-Pierre Seifert. “A General Obligation Model and Continuity: Enhanced Policy Enforcement Engine for Usage Control”. In: *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*. SACMAT ’08. Estes Park, CO, USA: ACM, 2008, pages 123–132. ISBN: 9781605581293. DOI: 10.1145/1377836.1377856 (cited on pages 15, 157).
- [92] Florian Kelbert. “Data Usage Control for the Cloud”. In: *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. CCGrid ’13. Delft, The Netherlands: IEEE, 2013, pages 156–159. ISBN: 9781467364652. DOI: 10.1109/CCGrid.2013.35 (cited on page 24).

- [93] Florian Kelbert and Alexander Pretschner. “A Fully Decentralized Data Usage Control Enforcement Infrastructure”. In: *Applied Cryptography and Network Security*. Edited by Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis. Volume 9092. Lecture Notes in Computer Science. Springer International Publishing, 2015, pages 409–430. ISBN: 9783319281650. DOI: 10.1007/978-3-319-28166-7_20 (cited on pages 21, 22, 24, 25, 27, 49, 84, 124).
- [94] Florian Kelbert and Alexander Pretschner. “Data Usage Control Enforcement in Distributed Systems”. In: *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*. CODASPY ’13. San Antonio, Texas, USA: ACM, 2013, pages 71–82. ISBN: 9781450318907. DOI: 10.1145/2435349.2435358 (cited on pages 24, 27, 49, 53, 78, 98, 103).
- [95] Florian Kelbert and Alexander Pretschner. “Decentralized Distributed Data Usage Control”. In: *Cryptology and Network Security*. Edited by Dimitris Gritzalis, Aggelos Kiayias, and Ioannis Askoxylakis. Volume 8813. Lecture Notes in Computer Science. Springer International Publishing, 2014, pages 353–369. ISBN: 9783319122793. DOI: 10.1007/978-3-319-12280-9_23 (cited on pages 22, 24, 27, 49, 60, 223, 233).
- [96] Florian Kelbert and Alexander Pretschner. “Towards a Policy Enforcement Infrastructure for Distributed Usage Control”. In: *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*. SACMAT ’12. Newark, New Jersey, USA: ACM, 2012, pages 119–122. ISBN: 9781450312950. DOI: 10.1145/2295136.2295159 (cited on page 24).
- [97] Vasileios P. Kemerlis, Georgios Portokalidis, Kangkook Jee, and Angelos D. Keromytis. “libdft: Practical Dynamic Data Flow Tracking for Commodity Systems”. In: *Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*. VEE ’12. London, England, UK: ACM, Mar. 2012, pages 121–132. ISBN: 9781450311762. DOI: 10.1145/2151024.2151042 (cited on page 158).
- [98] David Kempe, Jon Kleinberg, and Alan Demers. “Spatial Gossip and Resource Location Protocols”. In: *Journal of the ACM (JACM)* 51.6 (Nov. 2004), pages 943–967. ISSN: 0004-5411. DOI: 10.1145/1039488.1039491 (cited on page 91).
- [99] Stephen Kent and Karen Seo. *RFC 4301: Security Architecture for the Internet Protocol*. 2005. URL: <https://tools.ietf.org/html/rfc4301> (cited on page 99).
- [100] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. “Communication-Efficient Distributed Monitoring of Thresholded Counts”. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management*

- of Data*. SIGMOD '06. Chicago, IL, USA: ACM, 2006, pages 289–300. ISBN: 1595934340. DOI: 10.1145/1142473.1142507 (cited on page 167).
- [101] Rob H. Koenen, Jack Lacy, Michael Mackay, and Steve Mitchell. “The Long March to Interoperable Digital Rights Management”. In: *Proceedings of the IEEE* 92.6 (June 2004), pages 883–897. ISSN: 0018-9219. DOI: 10.1109/JPROC.2004.827357 (cited on page 173).
- [102] Paul Kranenburg and Dmitry Levin. *strace*. 2015. URL: <http://sourceforge.net/projects/strace/> (visited on 09/02/2015) (cited on page 77).
- [103] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. “Information Flow Control for Standard OS Abstractions”. In: *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*. SOSP '07. Stevenson, Washington, USA: ACM, 2007, pages 321–334. ISBN: 9781595935915. DOI: 10.1145/1294261.1294293 (cited on pages 159, 163).
- [104] Amol Kulkarni and Jesse Walker. *RFC 4261: Common Open Policy Service (COPS) Over Transport Layer Security (TLS)*. 2005. URL: <https://tools.ietf.org/html/rfc4261> (cited on pages 35, 72).
- [105] Ponnurangam Kumaraguru, Jorge Lobo, Lorrie Faith Cranor, and Seraphin B. Calo. “A Survey of Privacy Policy Languages”. In: *Proceedings of the 3rd Symposium on Usable Privacy and Security / Workshop on Usable IT Security Management*. ACM, 2007 (cited on page 156).
- [106] Prachi Kumari. “Model-Based Policy Derivation for Usage Control”. PhD thesis. Technische Universität München, Garching b. München, Germany, 2015 (cited on pages 32, 42, 78, 89, 156, 177–179).
- [107] Prachi Kumari, Florian Kelbert, and Alexander Pretschner. “Data Protection in Heterogeneous Distributed Systems: A Smart Meter Example”. In: *Proceedings der 41. GI-Jahrestagung / Lecture Notes in Informatics (LNI)*. INFORMATIK 2011: Dependable Software for Critical Infrastructures. Berlin, Germany: Gesellschaft für Informatik e.V., Oct. 2011. ISBN: 9783885792864 (cited on page 24).
- [108] Prachi Kumari and Alexander Pretschner. “Deriving Implementation-level Policies for Usage Control Enforcement”. In: *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*. CODASPY. San Antonio, Texas, USA: ACM, 2012, pages 83–94. ISBN: 9781450310918. DOI: 10.1145/2133601.2133612 (cited on pages 32, 156, 178).
- [109] Prachi Kumari and Alexander Pretschner. “Model-Based Usage Control Policy Derivation”. In: *Engineering Secure Software and Systems*. Volume 7781. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pages 58–74. ISBN: 9783642365621. DOI: 10.1007/978-3-642-36563-8_5 (cited on pages 32, 156, 178).

- [110] Prachi Kumari, Alexander Pretschner, Jonas Peschla, and Jens-Michael Kuhn. “Distributed Data Usage Control for Web Applications: A Social Network Implementation”. In: *Proceedings of the First ACM Conference on Data and Application Security and Privacy*. CODASPY. San Antonio, TX, USA: ACM, 2011, pages 85–96. ISBN: 9781450304665. DOI: 10.1145/1943513.1943526 (cited on pages 17, 35, 36, 77, 178, 179).
- [111] David Kyle and JoséCarlos Brustoloni. “UCLinux: a Linux Security Module for Trusted-Computing-based Usage Controls Enforcement”. In: *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*. STC ’07. Alexandria, Virginia, USA: ACM, 2007, pages 63–70. ISBN: 9781595938886. DOI: 10.1145/1314354.1314371 (cited on page 171).
- [112] Avinash Lakshman and Prashant Malik. “Cassandra: A Decentralized Structured Storage System”. In: *ACM SIGOPS Operating Systems Review* 44.2 (Apr. 2010), pages 35–40. ISSN: 0163-5980. DOI: 10.1145/1773912.1773922 (cited on page 87).
- [113] Leslie Lamport. “Paxos Made Simple”. In: *SIGACT News Distributed Computing Column* 5 32.4 (Dec. 2001), pages 51–58. ISSN: 0163-5700. DOI: 10.1145/568425.568433 (cited on page 88).
- [114] Leslie Lamport. “The Part-time Parliament”. In: *ACM Transactions on Computer Systems* 16.2 (May 1998), pages 133–169. ISSN: 0734-2071. DOI: 10.1145/279227.279229 (cited on page 88).
- [115] Aliaksandr Lazouski, Gaetano Mancini, Fabio Martinelli, and Paolo Mori. “Architecture, Workflows, and Prototype for Stateful Data Usage Control in Cloud”. In: *IEEE Security and Privacy Workshops (SPW)*. May 2014, pages 23–30. DOI: 10.1109/SPW.2014.13 (cited on pages 15, 17, 18, 35, 36, 77, 162, 165, 166, 169).
- [116] Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. “Usage Control in Computer Security: A Survey”. In: *Computer Science Review* 4.2 (2010), pages 81–99. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2010.02.002 (cited on page 156).
- [117] Wlodzimierz Lewandowski and Claudine Thomas. “GPS Time Transfer”. In: *Proceedings of the IEEE* 79.7 (July 1991), pages 991–1000. ISSN: 0018-9219. DOI: 10.1109/5.84976 (cited on page 89).
- [118] Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. “The Glory of the Past”. In: *Logics of Programs*. Edited by Rohit Parikh. Volume 193. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1985, pages 196–218. ISBN: 9783540156482. DOI: 10.1007/3-540-15648-8_16 (cited on page 32).
- [119] Daniel Lienert. “Distributed Usage Control for the MySQL Database Server”. Diploma Thesis. Karlsruhe Institute of Technology, Germany, 2012 (cited on pages 31, 36, 53, 77, 158, 178).

- [120] Linux man pages. *ptrace(2): process trace*. 2015. URL: <http://linux.die.net/man/2/ptrace> (visited on 09/02/2015) (cited on pages 44, 77).
- [121] Linux man pages. *scp(1): secure copy*. 2015. URL: <http://linux.die.net/man/1/scp> (visited on 09/02/2015) (cited on page 105).
- [122] Linux man pages. *sendfile(2)*. 2015. URL: <http://linux.die.net/man/2/sendfile> (visited on 09/02/2015) (cited on page 123).
- [123] Linux man pages. *time - time a simple command or give resource usage*. 2015. URL: <http://linux.die.net/man/1/time> (visited on 09/02/2015) (cited on page 107).
- [124] Qiong Liu, Reihaneh Safavi-Naini, and Nicholas Paul Sheppard. “Digital Rights Management for Content Distribution”. In: *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21*. ACSW Frontiers '03. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2003, pages 49–58. ISBN: 1920682007. URL: <http://dl.acm.org/citation.cfm?id=827987.827994> (cited on pages 19, 172).
- [125] Michael Lörcher. “Data Usage Control for the Thunderbird Mail Client”. Master’s thesis. University of Kaiserslautern, Germany, 2012 (cited on pages 17, 31, 36, 77, 158, 178, 179).
- [126] Enrico Lovat. “Cross-layer Data-centric Usage Control”. PhD thesis. Technische Universität München, Garching b. München, Germany, 2015 (cited on pages 32, 78, 158, 159, 178).
- [127] Enrico Lovat and Florian Kelbert. “Structure Matters – A new Approach for Data Flow Tracking”. In: *IEEE Security and Privacy Workshops*. May 2014 (cited on page 25).
- [128] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. “Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation”. In: *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '05. Chicago, IL, USA: ACM, 2005, pages 190–200. ISBN: 1595930566. DOI: 10.1145/1065010.1065034 (cited on pages 158, 161).
- [129] Alexander Lukyanov. *LFTP - sophisticated file transfer program*. 2015. URL: <http://lftp.yar.ru/> (visited on 09/02/2015) (cited on page 105).
- [130] Tanu Malik, Ligia Nistor, and Ashish Gehani. “Tracking and Sketching Distributed Data Provenance”. In: *IEEE Sixth International Conference on e-Science*. Dec. 2010, pages 190–197. DOI: 10.1109/eScience.2010.51 (cited on page 164).

- [131] Masoud Mansouri-Samani and Morris Sloman. “Monitoring Distributed Systems”. In: *Network, IEEE* 7.6 (Nov. 1993), pages 20–30. ISSN: 0890-8044. DOI: 10.1109/65.244791 (cited on page 87).
- [132] Daniel W. Margo and Margo Seltzer. “The Case for Browser Provenance”. In: *First Workshop on Theory and Practice of Provenance*. TAPP’09. San Francisco, CA: USENIX Association, 2009. URL: <http://dl.acm.org/citation.cfm?id=1525932.1525941> (cited on page 159).
- [133] Nicolas Markey. “Past is for Free: on the Complexity of Verifying Linear Temporal Properties with Past”. In: *Electronic Notes in Theoretical Computer Science* 68.2 (72002). EXPRESS’02, 9th International Workshop on Expressiveness in Concurrency, pages 87–104. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(05)80366-4 (cited on page 32).
- [134] Fabio Martinelli, Paolo Mori, and Anna Vaccarelli. “Towards Continuous Usage Control on Grid Computational Services”. In: *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*. Oct. 2005, pages 82–82. DOI: 10.1109/ICAS-ICNS.2005.93 (cited on page 157).
- [135] Matt Buchanan. *DRM Officially Dead: Last Major Label Sony BMG Plans to Finally Drop DRM*. 2008. URL: <http://gizmodo.com/340598/drm-officially-dead-last-major-label-sony-bmg-plans-to-finally-drop-drm> (visited on 09/02/2015) (cited on page 178).
- [136] Sam Michiels, Kristof Verslype, Wouter Joosen, and Bart De Decker. “Towards a Software Architecture for DRM”. In: *Proceedings of the 5th ACM Workshop on Digital Rights Management*. DRM ’05. Alexandria, VA, USA: ACM, Nov. 2005, pages 65–74. ISBN: 1595932305. DOI: 10.1145/1102546.1102559 (cited on page 173).
- [137] Microsoft Corporation. *Architecture of Windows Media Rights Manager*. 2004. URL: <http://www.microsoft.com/windows/windowsmedia/howto/articles/drmarchitecture.aspx> (visited on 09/02/2015) (cited on pages 17, 19).
- [138] Microsoft Corporation. *Architecture of Windows Media Rights Manager*. 2004. URL: <http://www.microsoft.com/windows/windowsmedia/howto/articles/drmarchitecture.aspx> (visited on 09/08/2015) (cited on page 172).
- [139] Microsoft Corporation. *BitLocker Drive Encryption Overview*. 2015. URL: <http://windows.microsoft.com/en-us/windows-vista/bitlocker-drive-encryption-overview> (visited on 09/02/2015) (cited on page 96).

- [140] Microsoft Corporation. *Next-Generation Secure Computing Base*. 2015. URL: <http://www.microsoft.com/resources/ngscb/default.msp> (visited on 09/02/2015) (cited on page 97).
- [141] Microsoft Corporation. *Supporting Encrypted Media Extensions with Microsoft PlayReady DRM in web browsers*. URL: <https://msdn.microsoft.com/en-us/library/windows/apps/dn466732.aspx> (visited on 09/04/2015) (cited on page 173).
- [142] David L. Mills, Jim Martin, Jack Burbank, and William Kasch. *RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification*. 2010. URL: <https://tools.ietf.org/html/rfc5905> (cited on page 87).
- [143] Marco Casassa Mont, Siani Pearson, and Robert Thyne. “A Systematic Approach to Privacy Enforcement and Policy Compliance Checking in Enterprises”. In: *Trust and Privacy in Digital Business*. Edited by Simone Fischer-Hübner, Stevel Furnell, and Costas Lambrinoudakis. Volume 4083. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pages 91–102. ISBN: 9783540377504. DOI: 10.1007/11824633_10 (cited on page 157).
- [144] Marco Casassa Mont and Robert Thyne. “Privacy Policy Enforcement in Enterprises with Identity Management Solutions”. In: *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*. PST ’06. Markham, Ontario, Canada: ACM, 2006, 25:1–25:12. ISBN: 1595936041. DOI: 10.1145/1501434.1501465 (cited on page 157).
- [145] John Morrissey, Michael Renner, Daniel Roesen, and TJ Saunders. *The ProFTPD Project*. 2015. URL: <http://www.proftpd.org/> (visited on 09/02/2015) (cited on page 105).
- [146] Ben Moszkowski. “Executing Temporal Logic Programs”. In: *Seminar on Concurrency*. Edited by Stephen D. Brookes, Andrew William Roscoe, and Glynn Winskel. Volume 197. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1985, pages 111–130. ISBN: 9783540156703. DOI: 10.1007/3-540-15670-4_6 (cited on page 167).
- [147] Cornelius Moucha, Enrico Lovat, and Alexander Pretschner. “A Hypervisor-Based Bus System for Usage Control”. In: *Sixth International Conference on Availability, Reliability and Security*. ARES. Aug. 2011, pages 254–259. DOI: 10.1109/ARES.2011.44 (cited on page 16).
- [148] Mozilla Corporation. *DRM and the Challenge of Serving Users*. 2014. URL: <https://blog.mozilla.org/blog/2014/05/14/drm-and-the-challenge-of-serving-users/> (visited on 09/04/2015) (cited on page 173).

- [149] Mozilla Corporation. *Update on Digital Rights Management and Firefox*. 2015. URL: <https://blog.mozilla.org/blog/2015/05/12/update-on-digital-rights-management-and-firefox/> (visited on 09/04/2015) (cited on page 173).
- [150] Tilo Müller, Felix C. Freiling, and Andreas Dewald. “TRESOR Runs Encryption Securely Outside RAM”. In: *Proceedings of the 20th USENIX Conference on Security*. SEC’11. San Francisco, CA: USENIX Association, 2011. URL: <http://dl.acm.org/citation.cfm?id=2028067.2028084> (cited on page 100).
- [151] Zsolt Müller. *How to compile strace for use on an Android phone (running an ARM CPU)*. 2012. URL: <http://muzso.hu/2012/04/21/how-to-compile-strace-for-use-on-an-android-phone-running-an-arm-cpu> (visited on 09/02/2015) (cited on page 78).
- [152] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana Maclean, Daniel Margo, Margo Seltzer, and Robin Smogor. “Layering in Provenance Systems”. In: *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*. USENIX’09. San Diego, California: USENIX Association, 2009. URL: <http://dl.acm.org/citation.cfm?id=1855807.1855817> (cited on page 159).
- [153] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. “Provenance-Aware Storage Systems”. In: *Proceedings of the Annual Conference on USENIX ’06 Annual Technical Conference*. ATEC ’06. Boston, MA: USENIX Association, 2006. URL: <http://dl.acm.org/citation.cfm?id=1267359.1267363> (cited on page 159).
- [154] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo Seltzer. “Making a Cloud Provenance-aware”. In: *First Workshop on on Theory and Practice of Provenance*. TAPP’09. San Francisco, CA: USENIX Association, 2009, 12:1–12:10. URL: <http://dl.acm.org/citation.cfm?id=1525932.1525944> (cited on page 164).
- [155] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo Seltzer. “Provenance for the Cloud”. In: *Proceedings of the 8th USENIX Conference on File and Storage Technologies*. FAST’10. San Jose, California: USENIX Association, 2010. URL: <http://dl.acm.org/citation.cfm?id=1855511.1855526> (cited on page 164).
- [156] Ricardo Neisse, Alexander Pretschner, and Valentina Di Giacomo. “A Trustworthy Usage Control Enforcement Framework”. In: *Sixth International Conference on Availability, Reliability and Security (ARES)*. Aug. 2011, pages 230–235. DOI: 10.1109/ARES.2011.40 (cited on pages 19, 32, 35).
- [157] NGINX, Inc. *nginx*. 2015. URL: <http://www.nginx.org> (visited on 09/02/2015) (cited on page 105).

- [158] Åsmund Ahlmann Nyre. “Usage Control Enforcement - A Survey”. In: *Availability, Reliability and Security for Business, Enterprise and Health Information Systems*. Edited by AMin Tjoa, Gerald Quirchmayr, Ilsun You, and Lida Xu. Volume 6908. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pages 38–49. ISBN: 9783642232992. DOI: 10.1007/978-3-642-23300-5_4 (cited on page 157).
- [159] Danny O’Brien. *Lowering Your Standards: DRM and the Future of the W3C*. 2013. URL: <https://www.eff.org/deeplinks/2013/10/lowering-your-standards> (visited on 09/04/2015) (cited on page 173).
- [160] Organization for the Advancement of Structured Information Standards (OASIS). “eXtensible Access Control Markup Language (XACML) Version 3.0”. In: *OASIS Standard* (Jan. 2013), pages 1–154. URL: <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf> (visited on 09/02/2015) (cited on pages 35, 72, 156).
- [161] Ioannis Papagiannis and Peter Pietzuch. “CloudFilter: Practical Control of Sensitive Data Propagation to the Cloud”. In: *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*. CCSW ’12. Raleigh, North Carolina, USA: ACM, Oct. 2012, pages 97–102. ISBN: 9781450316651. DOI: 10.1145/2381913.2381931 (cited on pages 17, 18, 161, 164).
- [162] Vasilis Pappas, Vasileios P. Kemerlis, Angeliki Zavou, Michalis Polychronakis, and Angelos D. Keromytis. “CloudFence: Data Flow Tracking as a Cloud Service”. In: *Research in Attacks, Intrusions, and Defenses*. Edited by Salvatore J. Stolfo, Angelos Stavrou, and Charles V. Wright. Volume 8145. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pages 411–431. ISBN: 9783642412837. DOI: 10.1007/978-3-642-41284-4_21 (cited on pages 17, 18, 161, 164).
- [163] Anthony Park and Mark Watson. *HTML5 Video in Safari on OS X Yosemite*. 2014. URL: <http://techblog.netflix.com/2014/06/html5-video-in-safari-on-os-x-yosemite.html> (visited on 09/08/2015) (cited on page 173).
- [164] Jaehong Park and Ravi Sandhu. “Originator Control in Usage Control”. In: *Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks*. 2002, pages 60–66. DOI: 10.1109/POLICY.2002.1011294 (cited on page 155).
- [165] Jaehong Park and Ravi Sandhu. “The UCON_{ABC} Usage Control Model”. In: *ACM Transactions on Information and System Security* 7.1 (Feb. 2004), pages 128–174. ISSN: 1094-9224. DOI: 10.1145/984334.984339 (cited on pages 15, 155, 156, 173).

- [166] Jaehong Park and Ravi Sandhu. “Towards Usage Control Models: Beyond Traditional Access Control”. In: *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*. SACMAT '02. Monterey, California, USA: ACM, 2002, pages 57–64. ISBN: 1581134967. DOI: 10.1145/507711.507722 (cited on pages 15, 155).
- [167] Siani Pearson and Boris Balacheff. *Trusted Computing Platforms: TCPA Technology in Context*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2002. ISBN: 0130092207 (cited on page 19).
- [168] Jonas Peschla. “Information Flow Tracking for JavaScript in Chromium”. Master’s Thesis. University of Kaiserslautern, Germany, 2012 (cited on pages 36, 53, 77, 158, 178).
- [169] Peter A.H. Peterson. “Cryptkeeper: Improving Security With Encrypted RAM”. In: *IEEE International Conference on Technologies for Homeland Security*. Nov. 2010, pages 120–126. DOI: 10.1109/THS.2010.5655081 (cited on page 100).
- [170] Alexander Pretschner, Matthias Büchler, Matúš Harvan, Christian Schaefer, and Thomas Walter. “Usage Control Enforcement with Data Flow Tracking for X11”. In: *5th International Workshop on Security and Trust Management*. STM. 2009 (cited on pages 16, 30, 31, 36, 53, 77, 155, 158, 178).
- [171] Alexander Pretschner, Manuel Hilty, and David Basin. “Distributed Usage Control”. In: *Communications of the ACM* 49.9 (Sept. 2006), pages 39–44. ISSN: 0001-0782. DOI: 10.1145/1151030.1151053 (cited on pages 15, 16, 155).
- [172] Alexander Pretschner, Manuel Hilty, David Basin, Christian Schaefer, and Thomas Walter. “Mechanisms for Usage Control”. In: *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*. ASI-ACCS. Tokyo, Japan: ACM, 2008, pages 240–244. ISBN: 9781595939791. DOI: 10.1145/1368310.1368344 (cited on pages 15, 27, 32, 155).
- [173] Alexander Pretschner, Manuel Hilty, Florian Schütz, Christian Schaefer, and Thomas Walter. “Usage Control Enforcement: Present and Future”. In: *IEEE Security & Privacy* 6.4 (2008), pages 44–53. ISSN: 1540-7993. DOI: 10.1109/MSP.2008.101 (cited on page 15).
- [174] Alexander Pretschner, Enrico Lovat, and Matthias Büchler. “Representation-Independent Data Usage Control”. In: *Data Privacy Management and Autonomous Spontaneous Security*. Volume 7122. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pages 122–140. ISBN: 9783642288784. DOI: 10.1007/978-3-642-28879-1_9 (cited on pages 15–17, 27, 30–32, 35, 53, 158, 178).

- [175] Alexander Pretschner, Fabio Massacci, and Manuel Hilty. “Usage Control in Service-Oriented Architectures”. In: *Trust, Privacy and Security in Digital Business*. Edited by Costas Lambrinoudakis, Günther Pernul, and Amin Tjoa. Volume 4657. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pages 83–93. ISBN: 9783540744085. DOI: 10.1007/978-3-540-74409-2_11 (cited on page 155).
- [176] Alexander Pretschner, Judith Rüesch, Christian Schaefer, and Thomas Walter. “Formal Analyses of Usage Control Policies”. In: *International Conference on Availability, Reliability and Security*. ARES. Mar. 2009, pages 98–105. DOI: 10.1109/ARES.2009.100 (cited on page 27).
- [177] Alexander Pretschner, Florian Schütz, Christian Schaefer, and Thomas Walter. “Policy Evolution in Distributed Usage Control”. In: *Proceedings of the 4th International Workshop on Security and Trust Management*. Volume 244. 2009, pages 109–123. DOI: 10.1016/j.entcs.2009.07.042 (cited on page 156).
- [178] Niels Provos. “Improving Host Security with System Call Policies”. In: *Proceedings of the 12th Conference on USENIX Security Symposium*. Washington, DC: USENIX Association, 2003. URL: http://static.usenix.org/publications/library/proceedings/sec03/tech/full_papers/provos/provos_html/ (cited on page 44).
- [179] QEMU Team. *QEMU: Open Source Processor Emulator*. 2015. URL: <http://www.qemu.org> (visited on 09/02/2015) (cited on page 162).
- [180] Siegfried Rasthofer, Steven Arzt, Enrico Lovat, and Eric Bodden. “DroidForce: Enforcing Complex, Data-centric, System-wide Policies in Android”. In: *Ninth International Conference on Availability, Reliability and Security (ARES)*. Sept. 2014, pages 40–49. DOI: 10.1109/ARES.2014.13 (cited on pages 36, 77).
- [181] Giovanni Russello and Naranker Dulay. “xDUCON: Coordinating Usage Control Policies in Distributed Domains”. In: *Third International Conference on Network and System Security*. NSS ’09. Oct. 2009, pages 246–253. DOI: 10.1109/NSS.2009.77 (cited on page 169).
- [182] Giovanni Russello, Enrico Scalavino, Naranker Dulay, and Emil C. Lupu. “Coordinating Data Usage Control in Loosely-Connected Networks”. In: *IEEE International Symposium on Policies for Distributed Systems and Networks*. POLICY ’10. July 2010, pages 30–39. DOI: 10.1109/POLICY.2010.20 (cited on page 169).
- [183] Reiner Sailer, Trent Jaeger, Xiaolan Zhang, and Leendert van Doorn. “Attestation-based Policy Enforcement for Remote Access”. In: *Proceedings of the 11th ACM Conference on Computer and Communications Security*. CCS ’04. Washington DC, USA: ACM, 2004, pages 308–317. ISBN: 1581139616. DOI: 10.1145/1030083.1030125 (cited on page 19).

- [184] David Samyde, Sergei Skorobogatov, Ross Anderson, and Jean-Jacques Quisquater. “On a New Way to Read Data from Memory”. In: *Proceedings of the First International IEEE Security in Storage Workshop*. Dec. 2002, pages 65–69. DOI: 10.1109/SISW.2002.1183512 (cited on page 100).
- [185] Ravi Sandhu and Jaehong Park. “Usage Control: A Vision for Next Generation Access Control”. In: *Computer Network Security*. Edited by Vladimir Gorodetsky, Leonard Popyack, and Victor Skormin. Volume 2776. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pages 17–31. ISBN: 9783540407973. DOI: 10.1007/978-3-540-45215-7_2 (cited on page 155).
- [186] Nuno Santos, Krishna P. Gummadi, and Rodrigo Rodrigues. “Towards Trusted Cloud Computing”. In: *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*. HotCloud’09. San Diego, California: USENIX Association, 2009. URL: <http://dl.acm.org/citation.cfm?id=1855533.1855536> (cited on page 19).
- [187] Can Sar and Pei Cao. *Lineage File System*. URL: <http://crypto.stanford.edu/~cao/lineage.html> (visited on 09/02/2015) (cited on page 159).
- [188] Shakti Saxena. “Data Usage Control In Office Application”. Master’s Thesis. Technische Universität München, Germany, 2014 (cited on pages 31, 36, 53, 77, 158, 178, 179).
- [189] Dries Schellekens, Brecht Wyseur, and Bart Preneel. “Remote Attestation on Legacy Operating Systems with Trusted Platform Modules”. In: *Science of Computer Programming* 74.1-2 (2008). Special Issue on Security and Trust, pages 13–22. ISSN: 0167-6423. DOI: 10.1016/j.scico.2008.09.005 (cited on page 19).
- [190] Spencer Shepler, Mike Eisler, and David Noveck. *RFC 5661: Network File System (NFS) Version 4 Minor Version 1 Protocol*. 2010. URL: <https://tools.ietf.org/html/rfc5661> (cited on page 164).
- [191] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. *A Survey of Data Provenance Techniques*. Technical report. Computer Science Department, Indiana University, Bloomington IN, 2005 (cited on page 159).
- [192] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. *Thrift: Scalable Cross-Language Services Implementation*. Technical report. Facebook Inc., 2007 (cited on page 72).
- [193] Mark Stamp. “Digital Rights Management: The Technology Behind the Hype”. In: *Journal of Electronic Commerce Research* 4.3 (2003), pages 102–112 (cited on page 172).

- [194] Maicon Stihler, Altair Olivo Santin, Alcides Calsavara, and Arlindo L. Marcon Jr. “Distributed Usage Control Architecture for Business Coalitions”. In: *IEEE International Conference on Communications*. ICC '09. June 2009, pages 708–713. ISBN: 9781424434343. DOI: 10.1109/ICC.2009.5198940 (cited on page 170).
- [195] Linying Su, David W. Chadwick, Andrew Basden, and James A. Cunningham. “Automated Decomposition of Access Control Policies”. In: *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*. June 2005, pages 3–13. DOI: 10.1109/POLICY.2005.10 (cited on page 168).
- [196] S.R. Subramanya and Byung K. Yi. “Digital Rights Management”. In: *Potentials, IEEE* 25.2 (Mar. 2006), pages 31–34. ISSN: 0278-6648. DOI: 10.1109/MP.2006.1649008 (cited on pages 19, 172).
- [197] Gelareh Taban, Alvaro A. Cárdenas, and Virgil D. Gligor. “Towards a Secure and Interoperable DRM Architecture”. In: *Proceedings of the ACM Workshop on Digital Rights Management*. DRM '06. Alexandria, Virginia, USA: ACM, Oct. 2006, pages 69–78. ISBN: 159593555X. DOI: 10.1145/1179509.1179524 (cited on page 173).
- [198] The Apache Software Foundation. *Apache Thrift*. 2015. URL: <https://thrift.apache.org/> (visited on 09/02/2015) (cited on page 72).
- [199] The Apache Software Foundation. *The Apache Cassandra Project*. 2015. URL: <http://cassandra.apache.org/> (visited on 09/02/2015) (cited on page 87).
- [200] The Apache Software Foundation. *The Apache HTTP Server Project*. 2015. URL: <http://httpd.apache.org/> (visited on 09/02/2015) (cited on page 105).
- [201] The GnuPG Project. *The GNU Privacy Guard*. 2015. URL: <https://gnupg.org/> (visited on 09/02/2015) (cited on page 99).
- [202] The OpenBSD Project. *OpenSSH*. 2015. URL: <http://www.openssh.com/> (visited on 09/02/2015) (cited on page 105).
- [203] Danan Thilakanathan, Rafael Calvo, Shiping Chen, and Surya Nepal. “Secure and Controlled Sharing of Data in Distributed Computing”. In: *16th IEEE International Conference on Computational Science and Engineering (CSE)*. Dec. 2013, pages 825–832. DOI: 10.1109/CSE.2013.125 (cited on page 170).
- [204] Danan Thilakanathan, Shiping Chen, Surya Nepal, Rafael Calvo, and Leila Alem. “A Platform for Secure Monitoring and Sharing of Generic Health Data in the Cloud”. In: *Future Generation Computer Systems* 35 (2014). Special Section: Integration of Cloud Computing and Body Sensor Networks; Guest Editors: Giancarlo Fortino and Mukaddim Pathan, pages 102–113. ISSN: 0167-739X. DOI: 10.1016/j.future.2013.09.011 (cited on page 170).

- [205] Slim Trabelsi, Akram Njeh, Laurent Bussard, and Gregory Neven. “PPL Engine: A Symmetric Architecture for Privacy Policy Handling”. In: *W3C Workshop on Privacy and data usage control*. W3C, 2010, pages 1–5. ISBN: 9788897253013 (cited on page 156).
- [206] Slim Trabelsi and Jakub Sendor. “Sticky Policies for Data Control in the Cloud”. In: *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*. July 2012, pages 75–80. DOI: 10.1109/PST.2012.6297922 (cited on page 162).
- [207] TrueCrypt Foundation. *TrueCrypt*. 2014. URL: <http://www.truecrypt.org> (visited on 09/02/2015) (cited on page 96).
- [208] Trusted Computing Group. *TPM Main Specification*. 2014. URL: http://www.trustedcomputinggroup.org/resources/tpm_main_specification (visited on 09/02/2015) (cited on pages 97, 180).
- [209] Tatsuhiko Tsujikawa. *aria2*. 2015. URL: <http://aria2.sourceforge.net/> (visited on 09/02/2015) (cited on page 105).
- [210] John Turek and Dennis Shasha. “The Many Faces of Consensus in Distributed Systems”. In: *IEEE Computer* 25.6 (June 1992), pages 8–17. ISSN: 0018-9162. DOI: 10.1109/2.153253 (cited on page 88).
- [211] Kevin Twidle, Naranker Dulay, Emil Lupu, and Morris Sloman. “Ponder2: A Policy System for Autonomous Pervasive Environments”. In: *Autonomic and Autonomous Systems, 2009. ICAS '09. Fifth International Conference on*. Apr. 2009, pages 330–335. DOI: 10.1109/ICAS.2009.42 (cited on page 156).
- [212] Werner Vogels. “Eventually Consistent”. In: *Communications of the ACM* 52.1 (Jan. 2009), pages 40–44. ISSN: 0001-0782. DOI: 10.1145/1435417.1435432 (cited on page 88).
- [213] W3C. *HTML 5*. 2014. URL: <http://www.w3.org/TR/html5/> (visited on 09/04/2015) (cited on page 173).
- [214] Patrick Wenz. “Data Usage Control for ChromiumOS”. Diploma Thesis. Karlsruhe Institute of Technology, Germany, 2012 (cited on pages 17, 36, 43, 77, 158, 178).
- [215] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, and Greg Kroah-Hartman. “Linux Security Modules: General Security Support for the Linux Kernel”. In: *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2002, pages 17–31. ISBN: 1931971005. URL: <http://dl.acm.org/citation.cfm?id=647253.720287> (cited on page 171).
- [216] Tobias Wüchner and Alexander Pretschner. “Data Loss Prevention Based on Data-Driven Usage Control”. In: *IEEE 23rd International Symposium on Software Reliability Engineering*. ISSRE. Nov. 2012, pages 151–160. DOI: 10.1109/ISSRE.2012.10 (cited on pages 16, 17, 31, 36, 43, 53, 77, 158, 178).

- [217] Min Xu, Xuxian Jiang, Ravi Sandhu, and Xinwen Zhang. “Towards a VMM-based Usage Control Framework for OS Kernel Integrity Protection”. In: *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*. SACMAT '07. Sophia Antipolis, France: ACM, 2007, pages 71–80. ISBN: 9781595937452. DOI: 10.1145/1266840.1266852 (cited on page 157).
- [218] Chih-Ta Yen, Horng-Twu Liaw, and Nai-Wei Lo. “Digital Rights Management System with User Privacy, Usage Transparency, and Superdistribution Support”. In: *International Journal of Communication Systems* 27.10 (2014), pages 1714–1730. ISSN: 1099-1131. DOI: 10.1002/dac.2431 (cited on page 172).
- [219] Angeliki Zavou, Vasilis Pappas, Vasileios P. Kemerlis, Michalis Polychronakis, Georgios Portokalidis, and Angelos D. Keromytis. “Cloudopsy: An Autopsy of Data Flows in the Cloud”. In: *Human Aspects of Information Security, Privacy, and Trust*. Edited by Louis Marinos and Ioannis Askoxylakis. Volume 8030. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pages 366–375. ISBN: 9783642393440. DOI: 10.1007/978-3-642-39345-7_39 (cited on page 161).
- [220] Angeliki Zavou, Georgios Portokalidis, and Angelos Keromytis. “Taint-Exchange: A Generic System for Cross-Process and Cross-Host Taint Tracking”. In: *Advances in Information and Computer Security*. Volume 7038. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pages 113–128. ISBN: 9783642251405. DOI: 10.1007/978-3-642-25141-2_8 (cited on pages 17, 18, 160, 164).
- [221] Nikolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. “Making Information Flow Explicit in HiStar”. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*. OSDI '06. Seattle, Washington: USENIX Association, 2006, pages 263–278. ISBN: 1931971471. URL: <http://dl.acm.org/citation.cfm?id=1298455.1298481> (cited on pages 159, 163).
- [222] Nikolai Zeldovich, Silas Boyd-Wickizer, and David Mazières. “Securing Distributed Systems with Information Flow Control”. In: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. NSDI'08. San Francisco, California: USENIX Association, 2008, pages 293–308. URL: <http://dl.acm.org/citation.cfm?id=1387589.1387610> (cited on pages 17, 18, 163, 164).
- [223] Olive Qing Zhang, Markus Kirchberg, Ryan KL Ko, and Bu Sung Lee. “How to Track Your Data: The Case for Cloud Computing Provenance”. In: *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*. Nov. 2011, pages 446–453. DOI: 10.1109/CloudCom.2011.66 (cited on pages 18, 164).

- [224] Qing Zhang, John McCullough, Justin Ma, Nabil Schear, Michael Vrable, Amin Vahdat, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. “Neon: System Support for Derived Data Management”. In: *Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. VEE '10. Pittsburgh, Pennsylvania, USA: ACM, 2010, pages 63–74. ISBN: 9781605589107. DOI: 10.1145/1735997.1736008 (cited on pages 17, 162, 164).
- [225] Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi Sandhu. “A Usage-based Authorization Framework for Collaborative Computing Systems”. In: *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*. SACMAT '06. Lake Tahoe, California, USA: ACM, 2006, pages 180–189. ISBN: 1595933530. DOI: 10.1145/1133058.1133084 (cited on page 157).
- [226] Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. “Formal Model and Policy Specification of Usage Control”. In: *ACM Transactions on Information and System Security* 8.4 (Nov. 2005), pages 351–387. ISSN: 1094-9224. DOI: 10.1145/1108906.1108908 (cited on page 15).
- [227] Xinwen Zhang, Jean-Pierre Seifert, and Ravi Sandhu. “Security Enforcement Model for Distributed Usage Control”. In: *International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing* (2008), pages 10–18. DOI: 10.1109/SUTC.2008.79 (cited on pages 15, 19, 171).

INDICES

Index

Symbols

\mathcal{A} (Addresses)	54
\mathcal{C} (Containers)	30
\mathcal{D} (Data)	30
\mathcal{E} (Events)	27
\mathcal{E}^A (Actual events)	28
\mathcal{E}_F (Data flow events)	28
\mathcal{E}^I (Intended events)	28
\mathcal{E}_U (Data usage events)	28
\mathcal{I} (Identifiers)	30
\mathcal{N} (Parameter names)	27
Ω (State-based operators)	32
\mathcal{P} (Principals)	30
\mathcal{R} (Transition relation)	30
<i>refines</i> (Event refinement)	28
<i>refines$_{\Sigma}$</i> (Event refinement using states)	31
<i>relevant</i>	60
Σ (Data flow states)	30
\mathcal{S} (System events)	28
\mathcal{S}^A (Actual system events)	28
\mathcal{S}_F^A (Actual data flow system events)	28
\mathcal{S}_F (Data flow system events)	28
\mathcal{S}^I (Intended system events)	28
\mathcal{S}_U (Data usage system events)	28
<i>Sat</i>	64
\mathcal{T} (Traces)	29
\mathcal{V} (Parameter values)	27
\mathcal{Y} (Systems)	50

A

Action	32
Actual data flow system events (\mathcal{S}_F^A)	28
Actual events (\mathcal{E}^A)	28
Actual system events (\mathcal{S}^A)	28
Addresses (\mathcal{A})	54

Alias function (a)	30
C	
Cardinality operators	33
Cluster	87
Condition	32
Consistency Level	90
Containers (C)	30
D	
Data (\mathcal{D})	30
Data flow events (\mathcal{E}_F)	28
Data flow states (Σ)	30
Data flow system events (\mathcal{S}_F)	28
Data usage events (\mathcal{E}_U)	28
Data usage system events (\mathcal{S}_U)	28
E	
ECA rule	32
Event refinement (<i>refines</i>)	28
Event refinement using states (<i>refines$_{\Sigma}$</i>)	31
Events (\mathcal{E})	27
Expression tree	37
F	
Formula projections	64
G	
Global data flow state	52
Global policy	24
I	
Identifiers (\mathcal{I})	30
Initial State	30
Intended events (\mathcal{E}^I)	28
Intended system events (\mathcal{S}^I)	28
K	
Keyspace	88
L	
Local socket name	55
N	
Naming function (n)	30

O

Overlay of traces 52

P

Parameter names (\mathcal{N}) 27

Parameter values (\mathcal{V}) 27

Principals (\mathcal{P}) 30

Propositional operators 33

R

Refinement (*refines*) 28

Remote socket name 55

Replication strategy 88

S

State-based operators 32

Storage function (*s*) 30

System events (\mathcal{S}) 28

Systems (\mathcal{Y}) 50

T

Temporal operators 33

Timestep 29

Traces (\mathcal{T}) 29

Transition relation (\mathcal{R}) 30

Trigger event 32

List of Figures

1.1	Sequence of events in the running example.	23
2.1	Interactions of PEP, PDP, PIP and PMP upon observation of system event e	36
2.2	Expression tree of the condition of ECA rule 1a.	38
2.3	Policy evaluation by the PDP in the presence of event e at timestep i	39
3.1	Two independent systems, each featuring three PEPs.	51
3.2	Three systems running in parallel and the overlay of their traces.	52
3.3	Sequence of TCP-related system calls.	55
4.1	High-level component diagram and its most important interfaces.	70
4.2	Modeling the establishment of a <i>remote</i> TCP channel. ¹	80
4.3	Modeling the establishment of a <i>local</i> TCP channel. ¹	81
4.4	Cross-system data flow tracking and policy propagation. ¹	83
4.5	Four systems connected via the Cassandra database.	87
5.1	Transfer times for a 1KB file.	108
5.2	Transfer times for a 1MB file.	110
5.3	Transfer times for a 128MB file.	114
5.4	Transfer times for a 512MB file.	118
5.5	Transfer times for a bit rate of 10Mbps.	121
5.6	Recap of policy 1 and the corresponding ECA rules.	126
5.7	Communication overhead when enforcing ECA rule 1a on three systems.	128
5.8	Communication overhead when enforcing ECA rule 1b on three systems.	129
5.9	Communication overhead enforcing ECA rule 1a on three of seven systems.	130
5.10	Recap of policy 2 and the corresponding ECA rule.	133
5.11	Communication overhead when enforcing ECA rule 2 on seven systems.	134
5.12	Recap of policy 3 and the corresponding ECA rule.	135
5.13	Communication overhead when enforcing ECA rule 3 on eleven systems.	137
5.14	Communication overhead enforcing ECA rule 3 on five out of eleven systems.	138
5.15	Performance overheads for ECA rule 1a on three systems.	140
5.16	Performance overheads for ECA rule 1b on three systems.	141
5.17	Performance overheads for ECA rule 1a on three out of seven systems.	143
5.18	Performance overheads for ECA rule 2 on seven systems.	146

5.19 Performance overheads for ECA rule 3 on eleven systems.	147
5.20 Performance overheads for ECA rule 3 on five out of eleven systems. . .	148
C.1 Legend for boxplots in Appendix C.	243
C.2 Performance measurement results when transferring a 1KB file.	244
C.3 Measurement results when transferring a 1MB file with 50Mbps.	246
C.4 Measurement results when transferring a 1MB file with 100Mbps.	246
C.5 Measurement results when transferring a 1MB file with 300Mbps.	247
C.6 Measurement results when transferring a 128MB file with 50Mbps.	249
C.7 Measurement results when transferring a 128MB file with 100Mbps.	249
C.8 Measurement results when transferring a 128MB file with 300Mbps.	250
C.9 Measurement results when transferring a 512MB file with 50Mbps.	252
C.10 Measurement results when transferring a 512MB file with 100Mbps.	252
C.11 Measurement results when transferring a 512MB file with 300Mbps.	253

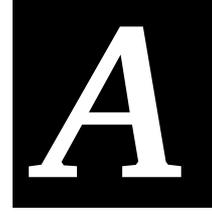
List of Tables

2.1	Example event trace spanning several timesteps.	29
2.2	Specification of the example policies from Section 1.5 as ECA rules. . . .	34
2.3	Evaluation of operators Ψ by the PDP; Java-like syntax.	38
2.4	Evaluation of operators Φ by the PDP; Java-like syntax.	40
2.5	Evaluation of operators Ω by the PIP; Java-like syntax.	43
5.1	Amount of intercepted and signaled system calls.	106
5.2	Time and standard deviation [ms] to transfer a 1KB file.	108
5.3	Time and standard deviation [ms] to transfer a 1MB file.	112
5.4	Time and standard deviation [s] to transfer a 128MB file.	116
5.5	Time and standard deviation [s] to transfer a 512MB file.	119
5.6	Time and standard deviation to transfer files at a bit rate of 10Mbps. . .	122
5.7	Median KB per second and standard deviation for ECA rule 1a on three systems.	132
5.8	Median KB per second and standard deviation for ECA rule 1b on three systems.	132
5.9	Median KB per second for ECA rule 1a on three out of seven systems. . .	132
5.10	Median KB per second and standard deviation for ECA rule 2 on seven systems.	135
5.11	Median KB per second and standard deviation for ECA rule 3 on eleven systems.	139
C.1	Absolute and relative overall median overhead when transferring a 1KB file.	244
C.2	Absolute and relative overall overheads when transferring a 1MB file. . .	247
C.3	Absolute and relative overall overheads when transferring a 128MB file.	250
C.4	Absolute and relative overall overheads when transferring a 512MB file.	253

List of Listings

4.1	Interfaces provided by the PDP.	73
4.2	Interfaces provided by the PIP.	74
4.3	Interfaces provided by the PMP.	74
4.4	Interfaces provided by the DMP.	75

APPENDICES



Correctness of Function *relevant*

Contents of this appendix have been published in [95].

Assuming ECA conditions to be given in disjunctive normal form (DNF), this appendix shows that function *relevant* as defined in Section 3.3.1 is correct in the following sense: For any tuple of concurrently executing traces $\tau \in \prod \mathcal{T}$, point in time $i \in \mathbb{N}$, formula $\varphi \in \Phi$, set of systems $Y = \text{relevant}(\varphi, i, \tau)$ and $X \subseteq \mathcal{Y} \setminus Y$ it holds that

$$(t_Y^\tau, i) \models \varphi \iff (t_{Y \cup X}^\tau, i) \models \varphi.$$

In other words, the set of systems $Y = \text{relevant}(\varphi, i, \tau)$ is sufficient to evaluate φ at time i given τ . Adding any other set of systems $X \subseteq \mathcal{Y} \setminus Y$ to the evaluation process does not change the evaluation's result. For each of the following proofs, part a) shows $(t_Y^\tau, i) \models \varphi \implies (t_{Y \cup X}^\tau, i) \models \varphi$, while part b) shows $(t_Y^\tau, i) \models \varphi \iff (t_{Y \cup X}^\tau, i) \models \varphi$.

Due to the uniqueness of sets \mathcal{C}_y and \mathcal{I}_y for any system $y \in \mathcal{Y}$ and the definition of the data flow state of sets of systems $X, Y \subseteq \mathcal{Y}$ (Section 3.1.2), it follows that

$$\begin{aligned} \forall Y \subseteq \mathcal{Y}, X \subseteq \mathcal{Y} \setminus Y, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, c \in \mathcal{C} \setminus \{nil\}, j \in \mathcal{I}, t \in \mathcal{T}: \\ \sigma_{t_Y^\tau}^i \cdot s(c) \subseteq \sigma_{t_{Y \cup X}^\tau}^i \cdot s(c) \\ \wedge \sigma_{t_Y^\tau}^i \cdot a(c) \subseteq \sigma_{t_{Y \cup X}^\tau}^i \cdot a(c) \\ \wedge \sigma_{t_Y^\tau}^i \cdot n(j) = c \implies \sigma_{t_{Y \cup X}^\tau}^i \cdot n(j) = c \\ \wedge \sigma_{t_Y^\tau}^i \cdot n(j) = nil \iff \sigma_{t_{Y \cup X}^\tau}^i \cdot n(j) = nil \end{aligned}$$

These relations between sets of systems $Y \subseteq \mathcal{Y}$ and $X \subseteq \mathcal{Y} \setminus Y$ will be leveraged throughout the following proofs.

Proof. For $\varphi = e; e \in \mathcal{E}$

$$\begin{aligned}
 \text{a) } & \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, e \in \mathcal{E}, \varphi \in \Phi, \varphi = e, Y = \text{relevant}(\varphi, i, \tau), \\
 & X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}: \\
 & (t_Y^\tau, i) \models \varphi \\
 \iff & \exists e' \in t_Y^\tau(i) : (e', \sigma_{t_Y^\tau}^i) \text{refines}_\Sigma e \\
 \implies & \exists e' \in t_{Y \cup X}^\tau(i) : (e', \sigma_{t_{Y \cup X}^\tau}^i) \text{refines}_\Sigma e \\
 \iff & (t_{Y \cup X}^\tau, i) \models \varphi
 \end{aligned}$$

$$\begin{aligned}
 \text{b) Assume: } & \exists \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, e \in \mathcal{E}, \varphi \in \Phi, \varphi = e, \\
 & Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}: \\
 & (t_{Y \cup X}^\tau, i) \models \varphi \not\Rightarrow (t_Y^\tau, i) \models \varphi \\
 \iff & \exists e' \in t_{Y \cup X}^\tau(i) : (e', \sigma_{t_{Y \cup X}^\tau}^i) \text{refines}_\Sigma e \\
 & \wedge \nexists e'' \in t_Y^\tau(i) : (e'', \sigma_{t_Y^\tau}^i) \text{refines}_\Sigma e \\
 \implies & \exists e' \in t_X^\tau(i) : (e', \sigma_{t_X^\tau}^i) \text{refines}_\Sigma e \\
 \implies & \text{happens}(e, i, \tau) \cap X \neq \emptyset \\
 \implies & Y \cap X \neq \emptyset \wedge Y \cap X = \emptyset \text{ (**Contradiction**)}
 \end{aligned}$$

since $Y = \text{relevant}(\varphi, i, \tau) = \text{relevant}(e, i, \tau) = \text{happens}(e, i, \tau)$

and $X \subseteq \mathcal{Y} \setminus Y$.

Hence $\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, e \in \mathcal{E}, \varphi \in \Phi, \varphi = e, Y = \text{relevant}(\varphi, i, \tau),$

$X \subseteq \mathcal{Y} \setminus Y :$

$$(t_{Y \cup X}^\tau, i) \models \varphi \implies (t_Y^\tau, i) \models \varphi$$

□

Proof. For $\varphi = \underline{isNotIn}(d, C); d \in \mathcal{D}, C \subseteq \mathcal{C}$

a) Assume: $\exists \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, d \in \mathcal{D}, C \subseteq \mathcal{C}, \varphi \in \Phi, \varphi = \underline{isNotIn}(d, C),$

$Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}:$

$$(t_Y^\tau, i) \models \varphi \not\Rightarrow (t_{Y \cup X}^\tau, i) \models \varphi$$

$$\Leftrightarrow \forall c \in C : d \notin \sigma_{t_Y^\tau}^i \cdot s(c) \wedge \exists c' \in C : d \in \sigma_{t_{Y \cup X}^\tau}^i \cdot s(c')$$

$$\Leftrightarrow \nexists c \in C : d \in \sigma_{t_Y^\tau}^i \cdot s(c) \wedge \exists c' \in C : d \in \sigma_{t_{Y \cup X}^\tau}^i \cdot s(c')$$

$$\Rightarrow \exists c \in C : d \in \sigma_{t_X^\tau}^i \cdot s(c)$$

$$\Rightarrow \text{knowD}(\{d\}, i, \tau) \cap \text{knowC}(C) \cap X \neq \emptyset$$

$$\Rightarrow Y \cap X \neq \emptyset \wedge Y \cap X = \emptyset \text{ (Contradiction)}$$

since $Y = \text{relevant}(\varphi, i, \tau) = \text{relevant}(\underline{isNotIn}(d, C), i, \tau)$

$$= \text{knowD}(\{d\}, i, \tau) \cap \text{knowC}(C)$$

and $X \subseteq \mathcal{Y} \setminus Y.$

Hence $\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, d \in \mathcal{D}, C \subseteq \mathcal{C}, \varphi \in \Phi, \varphi = \underline{isNotIn}(d, C),$

$Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y :$

$$(t_Y^\tau, i) \models \varphi \Rightarrow (t_{Y \cup X}^\tau, i) \models \varphi$$

b) $\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, d \in \mathcal{D}, C \subseteq \mathcal{C}, \varphi \in \Phi, \varphi = \underline{isNotIn}(d, C),$

$Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}:$

$$(t_{Y \cup X}^\tau, i) \models \varphi$$

$$\Leftrightarrow \forall c \in C : d \notin \sigma_{t_{Y \cup X}^\tau}^i \cdot s(c)$$

$$\Rightarrow \forall c \in C : d \notin \sigma_{t_Y^\tau}^i \cdot s(c)$$

$$\Leftrightarrow (t_Y^\tau, i) \models \varphi$$

□

Proof. For $\varphi = \underline{isMaxIn}(d, m, C)$; $d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C}$

a) Assume: $\exists \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C}, \varphi \in \Phi,$

$\varphi = \underline{isMaxIn}(d, m, C), Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}:$

$(t_Y^\tau, i) \models \varphi \not\Rightarrow (t_{Y \cup X}^\tau, i) \models \varphi$

$\Leftrightarrow |\{c \in C \mid d \in \sigma_{t_Y^\tau}^i \cdot s(c)\}| \leq m \wedge |\{c \in C \mid d \in \sigma_{t_{Y \cup X}^\tau}^i \cdot s(c)\}| > m$

$\Rightarrow |\{c \in C \mid d \in \sigma_{t_{Y \cup X}^\tau}^i \cdot s(c)\}| - |\{c \in C \mid d \in \sigma_{t_Y^\tau}^i \cdot s(c)\}| > 0$

$\Rightarrow \exists c \in C : d \in \sigma_{t_X^\tau}^i \cdot s(c)$

$\Rightarrow \text{knowD}(\{d\}, i, \tau) \cap \text{knowC}(C) \cap X \neq \emptyset$

$\Rightarrow Y \cap X \neq \emptyset \wedge Y \cap X = \emptyset$ (**Contradiction**)

since $Y = \text{relevant}(\varphi, i, \tau) = \text{relevant}(\underline{isMaxIn}(d, m, C), i, \tau)$

$= \text{knowD}(\{d\}, i, \tau) \cap \text{knowC}(C)$

and $X \subseteq \mathcal{Y} \setminus Y.$

Hence $\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C}, \varphi \in \Phi,$

$\varphi = \underline{isMaxIn}(d, m, C), Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y :$

$(t_Y^\tau, i) \models \varphi \Rightarrow (t_{Y \cup X}^\tau, i) \models \varphi$

b) $\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, d \in \mathcal{D}, C \subseteq \mathcal{C}, \varphi \in \Phi, \varphi = \underline{isNotIn}(d, C),$

$Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}:$

$(t_{Y \cup X}^\tau, i) \models \varphi$

$\Leftrightarrow \forall c \in C : d \notin \sigma_{t_{Y \cup X}^\tau}^i \cdot s(c)$

$\Rightarrow \forall c \in C : d \notin \sigma_{t_Y^\tau}^i \cdot s(c)$

$\Leftrightarrow (t_Y^\tau, i) \models \varphi$

□

Proof. For $\varphi = \underline{isCombined}(d_1, d_2, C); d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C}$

$$\text{a) } \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C}, \varphi \in \Phi, \varphi = \underline{isCombined}(d_1, d_2, C),$$

$$Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}:$$

$$(t_Y^\tau, i) \models \varphi$$

$$\iff \exists c \in C : \{d_1, d_2\} \subseteq \sigma_{t_Y^\tau}^i \cdot s(c)$$

$$\implies \exists c \in C : \{d_1, d_2\} \subseteq \sigma_{t_{Y \cup X}^\tau}^i \cdot s(c)$$

$$\iff (t_{Y \cup X}^\tau, i) \models \varphi$$

$$\text{b) Assume: } \exists \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C}, \varphi \in \Phi,$$

$$\varphi = \underline{isCombined}(d_1, d_2, C), Y = \text{relevant}(\varphi, i, \tau),$$

$$X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}:$$

$$(t_{Y \cup X}^\tau, i) \models \varphi \not\Rightarrow (t_Y^\tau, i) \models \varphi$$

$$\iff \exists c \in C : \{d_1, d_2\} \subseteq \sigma_{t_{Y \cup X}^\tau}^i \cdot s(c) \wedge \nexists c' \in C : \{d_1, d_2\} \subseteq \sigma_{t_Y^\tau}^i \cdot s(c')$$

$$\implies \exists c \in C : \{d_1, d_2\} \subseteq \sigma_{t_X^\tau}^i \cdot s(c)$$

$$\implies \text{knowD}(\{d_1, d_2\}, i, \tau) \cap \text{knowC}(C) \cap X \neq \emptyset$$

$$\implies Y \cap X \neq \emptyset \wedge Y \cap X = \emptyset \text{ (Contradiction)}$$

$$\text{since } Y = \text{relevant}(\varphi, i, \tau) = \text{relevant}(\underline{isCombined}(d_1, d_2, C), i, \tau)$$

$$= \text{knowD}(\{d_1, d_2\}, i, \tau) \cap \text{knowC}(C)$$

$$\text{and } X \subseteq \mathcal{Y} \setminus Y.$$

$$\text{Hence } \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C}, \varphi \in \Phi,$$

$$\varphi = \underline{isCombined}(d_1, d_2, C), Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y :$$

$$(t_{Y \cup X}^\tau, i) \models \varphi \implies (t_Y^\tau, i) \models \varphi$$

□

Proof. For $\varphi = \underline{\text{not}}(\alpha); \alpha \in \Phi$

$$\text{a) \& b) } \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, \alpha, \varphi \in \Phi, \varphi = \underline{\text{not}}(\alpha), Y = \text{relevant}(\varphi, i, \tau),$$

$$A = \text{relevant}(\alpha, i, \tau), X \subseteq \mathcal{Y} \setminus Y, \tilde{A} \subseteq \mathcal{Y} \setminus A, t \in \mathcal{T}:$$

$$(t_Y^\tau, i) \models \varphi$$

$$\iff \neg((t_Y^\tau, i) \models \alpha)$$

$$\iff \neg((t_A^\tau, i) \models \alpha)$$

$$\text{since } A = \text{relevant}(\alpha, i, \tau) = \text{relevant}(\underline{\text{not}}(\alpha), i, \tau)$$

$$= \text{relevant}(\varphi, i, \tau) = Y$$

$$\iff \neg((t_{A \cup \tilde{A}}^\tau, i) \models \alpha)$$

$$\iff \neg((t_{Y \cup X}^\tau, i) \models \alpha)$$

$$\text{since } A = Y \text{ and } \tilde{A} = X$$

$$\iff (t_{Y \cup X}^\tau, i) \models \varphi$$

□

Proof. For $\varphi = \alpha \text{ and } \beta; \alpha, \beta \in \Phi$

$$\text{a) \& b) } \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, \alpha, \beta, \varphi \in \Phi, \varphi = \alpha \text{ and } \beta, Y = \text{relevant}(\varphi, i, \tau),$$

$$A = \text{relevant}(\alpha, i, \tau), B = \text{relevant}(\beta, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}:$$

$$(t_Y^\tau, i) \models \varphi$$

$$\iff (t_Y^\tau, i) \models \alpha \wedge (t_Y^\tau, i) \models \beta$$

$$\iff (t_A^\tau, i) \models \alpha \wedge (t_B^\tau, i) \models \beta$$

$$\text{since } A = \text{relevant}(\alpha, i, \tau)$$

$$\subseteq \text{relevant}(\alpha, i, \tau) \cup \text{relevant}(\beta, i, \tau) = \text{relevant}(\varphi, i, \tau) = Y$$

$$\text{and } B = \text{relevant}(\beta, i, \tau)$$

$$\subseteq \text{relevant}(\alpha, i, \tau) \cup \text{relevant}(\beta, i, \tau) = \text{relevant}(\varphi, i, \tau) = Y$$

$$\iff (t_{A \cup \tilde{A}}^\tau, i) \models \alpha \wedge (t_{B \cup \tilde{B}}^\tau, i) \models \beta$$

$$\text{with } \tilde{A} = (Y \setminus A) \cup X$$

$$\text{and } \tilde{B} = (Y \setminus B) \cup X$$

$$\iff (t_{A \cup ((Y \setminus A) \cup X)}^\tau, i) \models \alpha \wedge (t_{B \cup ((Y \setminus B) \cup X)}^\tau, i) \models \beta$$

$$\iff (t_{Y \cup X}^\tau, i) \models \alpha \wedge (t_{Y \cup X}^\tau, i) \models \beta$$

$$\iff (t_{Y \cup X}^\tau, i) \models \varphi$$

□

Proof. For $\varphi = \alpha$ or β ; $\alpha, \beta \in \Phi$

$$\begin{aligned}
\text{a) \& b) } & \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, \alpha, \beta, \varphi \in \Phi, \varphi = \alpha \text{ or } \beta, Y = \text{relevant}(\varphi, i, \tau), \\
& A = \text{relevant}(\alpha, i, \tau), B = \text{relevant}(\beta, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}: \\
& (t_Y^\tau, i) \models \varphi \\
& \iff (t_Y^\tau, i) \models \alpha \vee (t_Y^\tau, i) \models \beta \\
& \iff (t_A^\tau, i) \models \alpha \vee (t_B^\tau, i) \models \beta \\
& \text{since } A = \text{relevant}(\alpha, i, \tau) \\
& \quad \subseteq \text{relevant}(\alpha, i, \tau) \cup \text{relevant}(\beta, i, \tau) = \text{relevant}(\varphi, i, \tau) = Y \\
& \text{and } B = \text{relevant}(\beta, i, \tau) \\
& \quad \subseteq \text{relevant}(\alpha, i, \tau) \cup \text{relevant}(\beta, i, \tau) = \text{relevant}(\varphi, i, \tau) = Y \\
& \iff (t_{A \cup \tilde{A}}^\tau, i) \models \alpha \vee (t_{B \cup \tilde{B}}^\tau, i) \models \beta \\
& \text{with } \tilde{A} = (Y \setminus A) \cup X \\
& \text{and } \tilde{B} = (Y \setminus B) \cup X \\
& \iff (t_{A \cup ((Y \setminus A) \cup X)}^\tau, i) \models \alpha \vee (t_{B \cup ((Y \setminus B) \cup X)}^\tau, i) \models \beta \\
& \iff (t_{Y \cup X}^\tau, i) \models \alpha \vee (t_{Y \cup X}^\tau, i) \models \beta \\
& \iff (t_{Y \cup X}^\tau, i) \models \varphi
\end{aligned}$$

□

Proof. For $\varphi = \alpha$ before j ; $\alpha \in \Phi, j \in \mathbb{N}$

$$\begin{aligned}
\text{a) \& b) } & \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i, j \in \mathbb{N}, \alpha, \varphi \in \Phi, \varphi = \alpha \text{ before } j, Y = \text{relevant}(\varphi, i, \tau), \\
& A = \text{relevant}(\alpha, i - j, \tau), X \subseteq \mathcal{Y} \setminus Y, \tilde{A} \subseteq \mathcal{Y} \setminus A, t \in \mathcal{T}: \\
& (t_Y^\tau, i) \models \varphi \\
& \iff (t_Y^\tau, i - j) \models \alpha \\
& \iff (t_A^\tau, i - j) \models \alpha \\
& \text{since } A = \text{relevant}(\alpha, i - j, \tau) = \text{relevant}(\alpha \text{ before } j, i, \tau) \\
& \quad = \text{relevant}(\varphi, i, \tau) = Y \\
& \iff (t_{A \cup \tilde{A}}^\tau, i - j) \models \alpha \\
& \iff (t_{Y \cup X}^\tau, i - j) \models \alpha \\
& \text{since } A = Y \text{ and } \tilde{A} = X \\
& \iff (t_{Y \cup X}^\tau, i) \models \varphi
\end{aligned}$$

□

Proof. For $\varphi = \alpha$ since $\beta; \alpha, \beta \in \Phi$

$$\begin{aligned}
\text{a) \& b) } & \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, \alpha, \beta, \varphi \in \Phi, \varphi = \alpha \text{ since } \beta, Y = \bigcup_{l=0}^i \text{relevant}(\varphi, l, \tau), \\
& A = \bigcup_{l=0}^i \text{relevant}(\alpha, l, \tau), B = \bigcup_{l=0}^i \text{relevant}(\beta, l, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}: \\
& (t_Y^\tau, i) \models \varphi \\
& \iff \exists j \in [0, i] : ((t_Y^\tau, j) \models \beta \wedge \forall k \in (j, i] : (t_Y^\tau, k) \models \alpha) \\
& \quad \vee \forall k \in [0, i] : (t_Y^\tau, k) \models \alpha \\
& \iff \exists j \in [0, i] : ((t_{B_j}^\tau, j) \models \beta \wedge \forall k \in (j, i] : (t_{A_k}^\tau, k) \models \alpha) \\
& \quad \vee \forall k \in [0, i] : (t_{A_k}^\tau, k) \models \alpha \\
& \text{with } B_j = \text{relevant}(\beta, j, \tau) \subseteq \bigcup_{l=0}^i \text{relevant}(\beta, l, \tau) \\
& \quad \subseteq \bigcup_{l=0}^i (\text{relevant}(\alpha, l, \tau) \cup \text{relevant}(\beta, l, \tau)) \\
& \quad = \bigcup_{l=0}^i \text{relevant}(\varphi, l, \tau) = Y; \forall j \in [0, i] \\
& \text{and } A_k = \text{relevant}(\alpha, k, \tau) \subseteq \bigcup_{l=0}^i \text{relevant}(\alpha, l, \tau) \\
& \quad \subseteq \bigcup_{l=0}^i (\text{relevant}(\alpha, l, \tau) \cup \text{relevant}(\beta, l, \tau)) \\
& \quad = \bigcup_{l=0}^i \text{relevant}(\varphi, l, \tau) = Y; \forall k \in [0, i] \\
& \iff \exists j \in [0, i] : ((t_{B_j \cup \tilde{B}_j}^\tau, j) \models \beta \wedge \forall k \in (j, i] : (t_{A_k \cup \tilde{A}_k}^\tau, k) \models \alpha) \\
& \quad \vee \forall k \in [0, i] : (t_{A_k \cup \tilde{A}_k}^\tau, k) \models \alpha \\
& \text{with } \tilde{B}_j = (Y \setminus B_j) \cup X; \forall j \in [0, i] \\
& \text{and } \tilde{A}_k = (Y \setminus A_k) \cup X; \forall k \in [0, i] \\
& \iff \exists j \in [0, i] : ((t_{B_j \cup ((Y \setminus B_j) \cup X)}^\tau, j) \models \beta \\
& \quad \wedge \forall k \in (j, i] : (t_{A_k \cup ((Y \setminus A_k) \cup X)}^\tau, k) \models \alpha) \\
& \quad \vee \forall k \in [0, i] : (t_{A_k \cup ((Y \setminus A_k) \cup X)}^\tau, k) \models \alpha \\
& \iff \exists j \in [0, i] : ((t_{Y \cup X}^\tau, j) \models \beta \wedge \forall k \in (j, i] : (t_{Y \cup X}^\tau, k) \models \alpha) \\
& \quad \vee \forall k \in [0, i] : (t_{Y \cup X}^\tau, k) \models \alpha \\
& \iff (t_{Y \cup X}^\tau, i) \models \varphi
\end{aligned}$$

□

Proof. For $\varphi = \underline{repmin}(j, m, e); j, m \in \mathbb{N}, e \in \mathcal{E}$

$$\begin{aligned}
& \text{a) } \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i, j, m \in \mathbb{N}, e \in \mathcal{E}, \varphi \in \Phi, \varphi = \underline{repmin}(j, m, e), \\
& \quad Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}: \\
& \quad (t_Y^\tau, i) \models \varphi \\
& \iff m \leq \sum_{k=0}^{\min\{i,j\}-1} |\{e' \in t_Y^\tau(i-k) \mid (e', \sigma_{t_Y^\tau}^{i-k}) \text{refines}_\Sigma e\}| \\
& \implies m \leq \sum_{k=0}^{\min\{i,j\}-1} |\{e' \in t_{Y \cup X}^\tau(i-k) \mid (e', \sigma_{t_{Y \cup X}^\tau}^{i-k}) \text{refines}_\Sigma e\}| \\
& \iff (t_{Y \cup X}^\tau, i) \models \varphi
\end{aligned}$$

$$\begin{aligned}
& \text{b) Assume: } \exists \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i, j, m \in \mathbb{N}, e \in \mathcal{E}, \varphi \in \Phi, \varphi = \underline{repmin}(j, m, e), \\
& \quad Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y, t \in \mathcal{T}: \\
& \quad (t_{Y \cup X}^\tau, i) \models \varphi \not\Rightarrow (t_Y^\tau, i) \models \varphi \\
& \iff m \leq \sum_{k=0}^{\min\{i,j\}-1} |\{e' \in t_{Y \cup X}^\tau(i-k) \mid (e', \sigma_{t_{Y \cup X}^\tau}^{i-k}) \text{refines}_\Sigma e\}| \\
& \quad \wedge m > \sum_{k=0}^{\min\{i,j\}-1} |\{e' \in t_Y^\tau(i-k) \mid (e', \sigma_{t_Y^\tau}^{i-k}) \text{refines}_\Sigma e\}| \\
& \implies \exists k \in [0, \min\{i, j\} - 1], e' \in t_X^\tau(i-k) : (e', \sigma_X^{i-k}) \text{refines}_\Sigma e \\
& \implies \left(\bigcup_{k=0}^{\min\{i,j\}-1} \text{happens}(e, i-k, \tau) \right) \cap X \neq \emptyset \\
& \implies Y \cap X \neq \emptyset \wedge Y \cap X = \emptyset \text{ (**Contradiction**)}
\end{aligned}$$

since $Y = \text{relevant}(\varphi, i, \tau) = \text{relevant}(\underline{repmin}(j, m, e), i, \tau)$

$$= \bigcup_{k=0}^{\min\{i,j\}-1} \text{happens}(e, i-k, \tau)$$

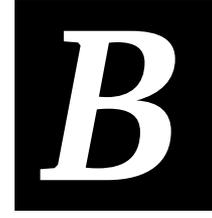
and $X \subseteq \mathcal{Y} \setminus Y$.

Hence $\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i, j, m \in \mathbb{N}, e \in \mathcal{E}, \varphi \in \Phi, \varphi = \underline{repmin}(j, m, e),$

$Y = \text{relevant}(\varphi, i, \tau), X \subseteq \mathcal{Y} \setminus Y :$

$$(t_{Y \cup X}^\tau, i) \models \varphi \implies (t_Y^\tau, i) \models \varphi$$

□



Correctness of Predicate *Sat*

Contents of this appendix have been published in [95].

This appendix proves that predicate *Sat* as defined in Section 3.3.2 is correct in the following sense: For any tuple of concurrently executing traces $\tau \in \prod \mathcal{T}$, set of systems $Y \subseteq \mathcal{Y}$, point in time $i \in \mathbb{N}$ and formula $\varphi \in \Phi$ in DNF, it holds that

$$(t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi) \implies (t_Y^\tau, i) \models \varphi.$$

In other words, if $\text{Sat}(\tau, Y, i, \varphi)$ holds then the set of systems Y can conclusively infer the global evaluation result of formula φ by only evaluating the corresponding formula projection φ_Y locally.

Proof. For $\text{relevant}(\varphi, i, \tau) \subseteq Y$ with $\varphi \in \Phi, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, Y \subseteq \mathcal{Y}$.

$$\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, Y \subseteq \mathcal{Y}, \text{relevant}(\varphi, i, \tau) \subseteq Y, t \in \mathcal{T}, \varphi \in \Phi :$$

$$(t_Y^\tau, i) \models \varphi_Y \implies (t_Y^\tau, i) \models \varphi.$$

This follows immediately with the claims and proofs provided in Section 3.3.2 and Appendix B, which essentially state:

$$\forall Y', X \subseteq \mathcal{Y}, Y' = \text{relevant}(\varphi, i, \tau), X = \mathcal{Y} \setminus Y' :$$

$$(t_{Y'}^\tau, i) \models \varphi \iff (t_{Y' \cup X}^\tau, i) \models \varphi$$

In particular, $Y' = \text{relevant}(\varphi, i, \tau) \subseteq Y$, and $t_{Y'}^\tau = t_{\text{relevant}(\varphi, i, \tau)}^\tau = t_Y^\tau$,

and thus

$$(t_{Y'}^\tau, i) \models \varphi \iff (t_Y^\tau, i) \models \varphi \iff (t_{Y' \cup X}^\tau, i) \models \varphi \iff (t_Y^\tau, i) \models \varphi.$$

□

Proof. For $\varphi = e$ with $e \in \mathcal{E}$.

$$\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi \in \Phi, e \in \mathcal{E}, \varphi = e :$$

$$\begin{aligned} & (t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi) \\ \iff & \exists e' \in t_Y^\tau(i) : (e', \sigma_{t_Y^\tau}^i) \text{refines}_\Sigma e \wedge \text{Sat}(\tau, Y, i, e) \\ \implies & \exists e' \in t_Y^\tau(i) : (e', \sigma_{t_Y^\tau}^i) \text{refines}_\Sigma e \end{aligned}$$

Note: $\text{Sat}(\tau, Y, i, e) = \text{true}$ by construction (Section 3.3.2).

$$\iff (t_Y^\tau, i) \models \varphi$$

□

Proof. For $\varphi = \text{not}(\text{isNotIn}(d, C))$ with $d \in \mathcal{D}, C \subseteq \mathcal{C}$.

$$\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi \in \Phi, d \in \mathcal{D}, C \subseteq \mathcal{C},$$

$$\varphi = \text{not}(\text{isNotIn}(d, C)) :$$

$$\begin{aligned} & (t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi) \\ \iff & (t_Y^\tau, i) \models \text{not}(\text{isNotIn}(d, C \cap \mathcal{C}_Y)) \\ & \quad \wedge \text{Sat}(\tau, Y, i, \text{not}(\text{isNotIn}(d, C))) \\ \iff & \neg((t_Y^\tau, i) \models \text{isNotIn}(d, C \cap \mathcal{C}_Y)) \end{aligned}$$

Note: $\text{Sat}(\tau, Y, i, \text{not}(\text{isNotIn}(d, C))) = \text{true}$

by construction (Section 3.3.2).

$$\begin{aligned} \iff & \neg(\forall c \in C \cap \mathcal{C}_Y : d \notin \sigma_{t_Y^\tau}^i.s(c)) \\ \iff & \exists c \in C \cap \mathcal{C}_Y : d \in \sigma_{t_Y^\tau}^i.s(c) \\ \implies & \exists c \in C : d \in \sigma_{t_Y^\tau}^i.s(c) \\ \iff & \neg(\forall c \in C : d \notin \sigma_{t_Y^\tau}^i.s(c)) \\ \iff & \neg((t_Y^\tau, i) \models \text{isNotIn}(d, C)) \\ \iff & (t_Y^\tau, i) \models \text{not}(\text{isNotIn}(d, C)) \\ \iff & (t_Y^\tau, i) \models \varphi \end{aligned}$$

□

Proof. For $\varphi = \underline{isCombined}(d_1, d_2, C)$ with $d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C}$.

$$\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi \in \Phi, d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C},$$

$$\varphi = \underline{isCombined}(d_1, d_2, C) :$$

$$(t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi)$$

$$\iff (t_Y^\tau, i) \models \underline{isCombined}(d_1, d_2, C \cap \mathcal{C}_Y)$$

$$\wedge \text{Sat}(\tau, Y, i, \underline{isCombined}(d_1, d_2, C))$$

$$\iff \exists c \in C \cap \mathcal{C}_Y : \{d_1, d_2\} \subseteq \sigma_{t_Y^\tau}^i \cdot s(c)$$

$$\text{Note: } \text{Sat}(\tau, Y, i, \underline{isCombined}(d_1, d_2, C)) = \text{true}$$

by construction (Section 3.3.2).

$$\implies \exists c \in C : \{d_1, d_2\} \subseteq \sigma_{t_Y^\tau}^i \cdot s(c)$$

$$\iff (t_Y^\tau, i) \models \underline{isCombined}(d_1, d_2, C)$$

$$\iff (t_Y^\tau, i) \models \varphi$$

□

Proof. For $\varphi = \underline{not(isMaxIn}(d, m, C))$ with $d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C}$.

$$\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i, m \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi \in \Phi, d \in \mathcal{D}, C \subseteq \mathcal{C},$$

$$\varphi = \underline{not(isMaxIn}(d, m, C)) :$$

$$(t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi)$$

$$\iff (t_Y^\tau, i) \models \underline{not(isMaxIn}(d, m, C \cap \mathcal{C}_Y))$$

$$\wedge \text{Sat}(\tau, Y, i, \underline{not(isMaxIn}(d, m, C)))$$

$$\iff \neg((t_Y^\tau, i) \models \underline{isMaxIn}(d, m, C \cap \mathcal{C}_Y))$$

$$\text{Note: } \text{Sat}(\tau, Y, i, \underline{not(isMaxIn}(d, m, C))) = \text{true}$$

by construction (Section 3.3.2).

$$\iff \neg(|\{c \in C \cap \mathcal{C}_Y \mid d \in \sigma_{t_Y^\tau}^i \cdot s(c)\}| \leq m)$$

$$\iff |\{c \in C \cap \mathcal{C}_Y \mid d \in \sigma_{t_Y^\tau}^i \cdot s(c)\}| > m$$

$$\implies |\{c \in C \mid d \in \sigma_{t_Y^\tau}^i \cdot s(c)\}| > m$$

$$\iff \neg(|\{c \in C \mid d \in \sigma_{t_Y^\tau}^i \cdot s(c)\}| \leq m)$$

$$\iff \neg((t_Y^\tau, i) \models \underline{isMaxIn}(d, m, C))$$

$$\iff (t_Y^\tau, i) \models \underline{not(isMaxIn}(d, m, C))$$

$$\iff (t_Y^\tau, i) \models \varphi$$

□

Proof. For $\varphi = \alpha$ and β with $\alpha, \beta \in \Phi$.

$$\begin{aligned}
& \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi, \alpha, \beta \in \Phi, \varphi = \alpha \text{ and } \beta : \\
& \quad (t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi) \\
& \iff (t_Y^\tau, i) \models \alpha_Y \text{ and } \beta_Y \wedge \text{Sat}(\tau, Y, i, \alpha \text{ and } \beta) \\
& \iff (t_Y^\tau, i) \models \alpha_Y \text{ and } \beta_Y \wedge \text{Sat}(\tau, Y, i, \alpha) \wedge \text{Sat}(\tau, Y, i, \beta) \\
& \iff (t_Y^\tau, i) \models \alpha_Y \wedge (t_Y^\tau, i) \models \beta_Y \wedge \text{Sat}(\tau, Y, i, \alpha) \wedge \text{Sat}(\tau, Y, i, \beta) \\
& \iff (t_Y^\tau, i) \models \alpha_Y \wedge \text{Sat}(\tau, Y, i, \alpha) \wedge (t_Y^\tau, i) \models \beta_Y \wedge \text{Sat}(\tau, Y, i, \beta) \\
& \implies (t_{\mathcal{Y}}^\tau, i) \models \alpha \wedge (t_{\mathcal{Y}}^\tau, i) \models \beta \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \alpha \text{ and } \beta \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \varphi
\end{aligned}$$

□

Proof. For $\varphi = \alpha$ or β with $\alpha, \beta \in \Phi$.

$$\begin{aligned}
& \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi, \alpha, \beta \in \Phi, \varphi = \alpha \text{ or } \beta : \\
& \quad (t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi) \\
& \iff (t_Y^\tau, i) \models \alpha_Y \text{ or } \beta_Y \wedge \text{Sat}(\tau, Y, i, \alpha \text{ or } \beta) \\
& \iff (t_Y^\tau, i) \models \alpha_Y \text{ or } \beta_Y \\
& \quad \wedge ((t_Y^\tau, i) \models \alpha_Y \wedge \text{Sat}(\tau, Y, i, \alpha) \\
& \quad \vee (t_Y^\tau, i) \models \beta_Y \wedge \text{Sat}(\tau, Y, i, \beta)) \\
& \iff ((t_Y^\tau, i) \models \alpha_Y \vee (t_Y^\tau, i) \models \beta_Y) \\
& \quad \wedge ((t_Y^\tau, i) \models \alpha_Y \wedge \text{Sat}(\tau, Y, i, \alpha) \\
& \quad \vee (t_Y^\tau, i) \models \beta_Y \wedge \text{Sat}(\tau, Y, i, \beta)) \\
& \iff (t_Y^\tau, i) \models \alpha_Y \wedge \text{Sat}(\tau, Y, i, \alpha) \vee (t_Y^\tau, i) \models \beta_Y \wedge \text{Sat}(\tau, Y, i, \beta) \\
& \implies (t_{\mathcal{Y}}^\tau, i) \models \alpha \vee (t_{\mathcal{Y}}^\tau, i) \models \beta \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \alpha \text{ or } \beta \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \varphi
\end{aligned}$$

□

Proof. For $\varphi = \alpha$ since β with $\alpha, \beta \in \Phi$.

$$\begin{aligned}
& \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi, \alpha, \beta \in \Phi, \varphi = \alpha \text{ since } \beta : \\
& \quad (t_Y^\tau, i) \models \varphi_Y \\
& \quad \quad \wedge ((\exists j \in [0, i] : (t_Y^\tau, j) \models \beta_Y \wedge \text{Sat}(\tau, Y, j, \beta)) \\
& \quad \quad \quad \wedge \forall k \in (j, i] : (t_Y^\tau, k) \models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \alpha)) \\
& \quad \quad \quad \vee (\forall k \in [0, i] : (t_Y^\tau, k) \models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \alpha))) \\
& \quad (t_Y^\tau, i) \models \alpha_Y \text{ since } \beta_Y \\
& \quad \quad \wedge ((\exists j \in [0, i] : (t_Y^\tau, j) \models \beta_Y \wedge \text{Sat}(\tau, Y, j, \beta)) \\
& \quad \quad \quad \wedge \forall k \in (j, i] : (t_Y^\tau, k) \models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \alpha)) \\
& \quad \quad \quad \vee (\forall k \in [0, i] : (t_Y^\tau, k) \models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \alpha))) \\
& \iff ((\exists j \in [0, i] : (t_Y^\tau, j) \models \beta_Y \wedge \forall k \in (j, i] : (t_Y^\tau, k) \models \alpha_Y) \\
& \quad \quad \vee (\forall k \in [0, i] : (t_Y^\tau, k) \models \alpha_Y)) \\
& \quad \quad \wedge ((\exists j \in [0, i] : (t_Y^\tau, j) \models \beta_Y \wedge \text{Sat}(\tau, Y, j, \beta)) \\
& \quad \quad \quad \wedge \forall k \in (j, i] : (t_Y^\tau, k) \models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \alpha)) \\
& \quad \quad \quad \vee (\forall k \in [0, i] : (t_Y^\tau, k) \models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \alpha))) \\
& \iff ((\exists j \in [0, i] : (t_Y^\tau, j) \models \beta_Y \wedge \text{Sat}(\tau, Y, j, \beta)) \\
& \quad \quad \wedge \forall k \in (j, i] : (t_Y^\tau, k) \models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \alpha)) \\
& \quad \quad \vee (\forall k \in [0, i] : (t_Y^\tau, k) \models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \alpha))) \\
& \implies ((\exists j \in [0, i] : (t_{\mathcal{Y}}^\tau, j) \models \beta \wedge \forall k \in (j, i] : (t_{\mathcal{Y}}^\tau, k) \models \alpha) \\
& \quad \quad \vee (\forall k \in [0, i] : (t_{\mathcal{Y}}^\tau, k) \models \alpha)) \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \alpha \text{ since } \beta \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \varphi
\end{aligned}$$

□

Proof. For $\varphi = \underline{\text{not}}(\alpha \text{ since } \beta)$ with $\alpha, \beta \in \Phi$.

$$\begin{aligned}
& \forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi, \alpha, \beta \in \Phi, \varphi = \underline{\text{not}}(\alpha \text{ since } \beta) : \\
& \quad (t_Y^\tau, i) \models \varphi_Y \\
& \quad \wedge ((\forall j \in [0, i] : (t_Y^\tau, j) \not\models \beta_Y \wedge \text{Sat}(\tau, Y, j, \underline{\text{not}}(\beta))) \\
& \quad \quad \wedge \exists k \in (j, i] : (t_Y^\tau, k) \not\models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \underline{\text{not}}(\alpha))) \\
& \quad \quad \vee (\exists k \in [0, i] : (t_Y^\tau, k) \models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \underline{\text{not}}(\alpha)))) \\
& \iff (t_Y^\tau, i) \models \underline{\text{not}}(\alpha_Y \text{ since } \beta_Y) \\
& \quad \wedge ((\forall j \in [0, i] : (t_Y^\tau, j) \not\models \beta_Y \wedge \text{Sat}(\tau, Y, j, \underline{\text{not}}(\beta))) \\
& \quad \quad \wedge \exists k \in (j, i] : (t_Y^\tau, k) \not\models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \underline{\text{not}}(\alpha))) \\
& \quad \quad \vee (\exists k \in [0, i] : (t_Y^\tau, k) \not\models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \underline{\text{not}}(\alpha)))) \\
& \iff ((\forall j \in [0, i] : (t_Y^\tau, j) \not\models \beta_Y \wedge \exists k \in (j, i] : (t_Y^\tau, k) \not\models \alpha_Y \\
& \quad \quad \vee (\exists k \in [0, i] : (t_Y^\tau, k) \not\models \alpha_Y)) \\
& \quad \quad \wedge ((\forall j \in [0, i] : (t_Y^\tau, j) \not\models \beta_Y \wedge \text{Sat}(\tau, Y, j, \underline{\text{not}}(\beta))) \\
& \quad \quad \quad \wedge \exists k \in (j, i] : (t_Y^\tau, k) \not\models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \underline{\text{not}}(\alpha))) \\
& \quad \quad \quad \vee (\exists k \in [0, i] : (t_Y^\tau, k) \not\models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \underline{\text{not}}(\alpha)))) \\
& \iff ((\forall j \in [0, i] : (t_Y^\tau, j) \not\models \beta_Y \wedge \text{Sat}(\tau, Y, j, \underline{\text{not}}(\beta))) \\
& \quad \quad \wedge \exists k \in (j, i] : (t_Y^\tau, k) \not\models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \underline{\text{not}}(\alpha))) \\
& \quad \quad \vee (\exists k \in [0, i] : (t_Y^\tau, k) \not\models \alpha_Y \wedge \text{Sat}(\tau, Y, k, \underline{\text{not}}(\alpha)))) \\
& \implies ((\forall j \in [0, i] : (t_Y^\tau, j) \not\models \beta \wedge \exists k \in (j, i] : (t_Y^\tau, k) \not\models \alpha \\
& \quad \quad \vee (\exists k \in [0, i] : (t_Y^\tau, k) \not\models \alpha)) \\
& \iff (t_Y^\tau, i) \models \underline{\text{not}}(\alpha \text{ since } \beta) \\
& \iff (t_Y^\tau, i) \models \varphi
\end{aligned}$$

□

Proof. For $\varphi = \alpha$ before j with $\alpha \in \Phi, j \in \mathbb{N}$.

$$\begin{aligned}
\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i, j \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi, \alpha \in \Phi, \varphi = \alpha \text{ before } j : \\
& (t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi) \\
& \iff (t_Y^\tau, i) \models \alpha_Y \text{ before } j \wedge \text{Sat}(\tau, Y, i, \alpha \text{ before } j) \\
& \iff (t_Y^\tau, i - j) \models \alpha_Y \wedge \text{Sat}(\tau, Y, i - j, \alpha) \\
& \implies (t_{\mathcal{Y}}^\tau, i - j) \models \alpha \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \alpha \text{ before } j \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \varphi
\end{aligned}$$

□

Proof. For $\varphi = \text{not}(\alpha \text{ before } j)$ with $\alpha \in \Phi, j \in \mathbb{N}$.

$$\begin{aligned}
\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i, j \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, \varphi, \alpha \in \Phi, \varphi = \text{not}(\alpha \text{ before } j) : \\
& (t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi) \\
& \iff (t_Y^\tau, i) \models \text{not}(\alpha_Y \text{ before } j) \wedge \text{Sat}(\tau, Y, i, \text{not}(\alpha \text{ before } j)) \\
& \iff \neg((t_Y^\tau, i) \models \alpha_Y \text{ before } j) \wedge \text{Sat}(\tau, Y, i, \text{not}(\alpha \text{ before } j)) \\
& \iff \neg((t_Y^\tau, i - j) \models \alpha_Y) \wedge \text{Sat}(\tau, Y, i - j, \text{not}(\alpha)) \\
& \iff (t_Y^\tau, i - j) \models \text{not}(\alpha_Y) \wedge \text{Sat}(\tau, Y, i - j, \text{not}(\alpha)) \\
& \implies (t_{\mathcal{Y}}^\tau, i - j) \models \text{not}(\alpha) \\
& \iff \neg((t_{\mathcal{Y}}^\tau, i - j) \models \alpha) \\
& \iff \neg((t_{\mathcal{Y}}^\tau, i) \models \alpha \text{ before } j) \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \text{not}(\alpha \text{ before } j) \\
& \iff (t_{\mathcal{Y}}^\tau, i) \models \varphi
\end{aligned}$$

□

Proof. For $\varphi = \text{repmi}n(j, m, e)$ with $e \in \mathcal{E}, j, m \in \mathbb{N}$.

$$\forall \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, i, j, m \in \mathbb{N}, Y \subseteq \mathcal{Y}, t \in \mathcal{T}, e \in \mathcal{E}, \varphi = \text{repmi}n(j, m, e) :$$

$$(t_Y^\tau, i) \models \varphi_Y \wedge \text{Sat}(\tau, Y, i, \varphi)$$

$$\iff (t_Y^\tau, i) \models \text{repmi}n(j, m, e) \wedge \text{Sat}(\tau, Y, i, \text{repmi}n(j, m, e))$$

$$\iff m \leq \sum_{k=0}^{\min\{i,j\}-1} |\{e' \in t_Y^\tau(i-k) \mid (e', \sigma_{t_Y^\tau}^{i-k}) \text{refines}_\Sigma e\}|$$

Note: $\text{Sat}(\tau, Y, i, \text{repmi}n(j, m, e)) = \text{true}$

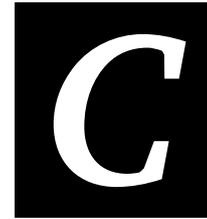
by construction (Section 3.3.2).

$$\implies m \leq \sum_{k=0}^{\min\{i,j\}-1} |\{e' \in t_Y^\tau(i-k) \mid (e', \sigma_{t_Y^\tau}^{i-k}) \text{refines}_\Sigma e\}|$$

$$\iff (t_Y^\tau, i) \models \text{repmi}n(j, m, e)$$

$$\iff (t_Y^\tau, i) \models \varphi$$

□



Evaluation: Cross-System Data Flow Tracking and Policy Propagation

This appendix complements Section 5.2 by providing more accurate and detailed measurement results. The legend for all boxplots in this appendix is given in Figure C.1.

C.1 Transferring Files of Size 1KB

Figure C.2 plots the measurement results presented in Table 5.2 (Section 5.2.1) as boxplots. Hence, the medians of the plotted boxes correspond to the values within the aforementioned table. Further, Figure C.2 shows for each set of measurements those two quartiles that are most close to the computed median.

In addition, Table C.1 shows for each client/server combination both the absolute and the relative performance overhead imposed by a fully functional usage control infrastructure, i.e. when signaling system calls to the Controller, and performing both local and remote data flow tracking (■ ■).

Figure C.1: Legend for boxplots in Appendix C.

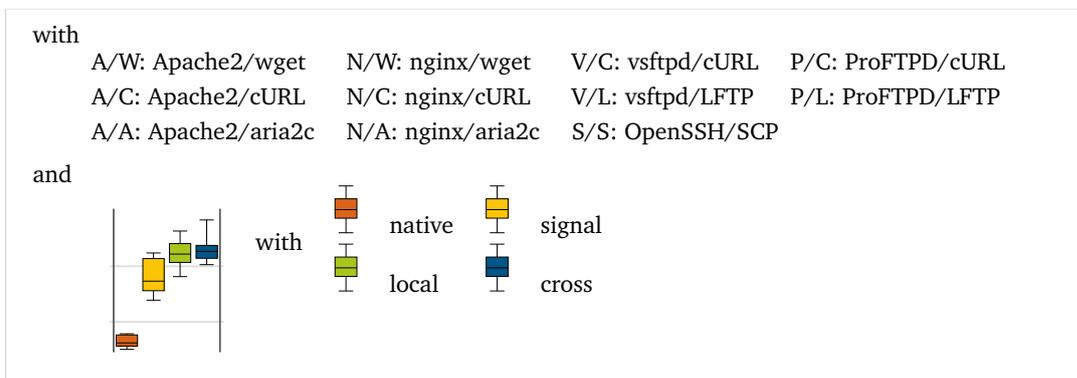


Figure C.2: Performance measurement results when transferring a 1KB file.

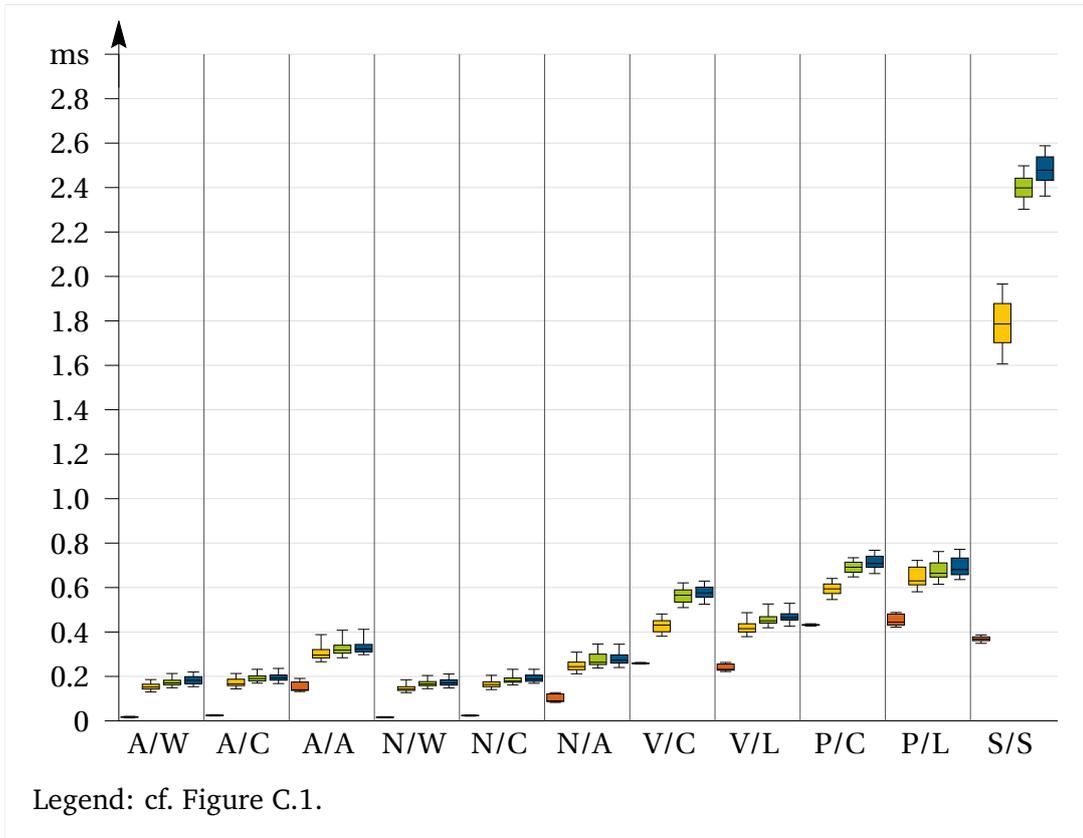


Table C.1: Absolute and relative overall median overhead when transferring a 1KB file.

Protocol	Server	Client	Overall overhead (■ ■)
			(any bit rate)
HTTPS	Apache2	wget	165ms (971%)
		cURL	168ms (672%)
		aria2c	184ms (131%)
	nginx	wget	154ms (906%)
		cURL	164ms (683%)
		aria2c	183ms (203%)
FTPS	vsftpd	cURL	317ms (122%)
		LFTP	232ms (100%)
	ProFTPD	cURL	276ms (64%)
		LFTP	237ms (53%)
SSH	OpenSSH	SCP	2110ms (573%)

C.2 Transferring Files of Size 1MB

Figures C.3 to C.5 on pages 246 and 247 visualize the performance measurement results presented in Table 5.3 as boxplots. For a bit rate of 50Mbps (Figure C.3) each boxplot is based on 30 repeated measurements, for a bit rate of 100Mbps (Figure C.3) each boxplot is based on 30 repeated measurements, and for a bit rate of 300Mbps (Figure C.3) each boxplot is based on 40 repeated measurements. The plotted medians correspond to the median values presented in Table 5.3.

In addition, Table C.2 (page 247) shows for each client/server combination and for different bit rates both the absolute and the relative performance overhead imposed by a fully functional usage control infrastructure, i.e. when signaling system calls to the Controller, and performing both local and remote data flow tracking (■ ■ ■).

Figure C.3: Measurement results when transferring a 1MB file with 50Mbps.

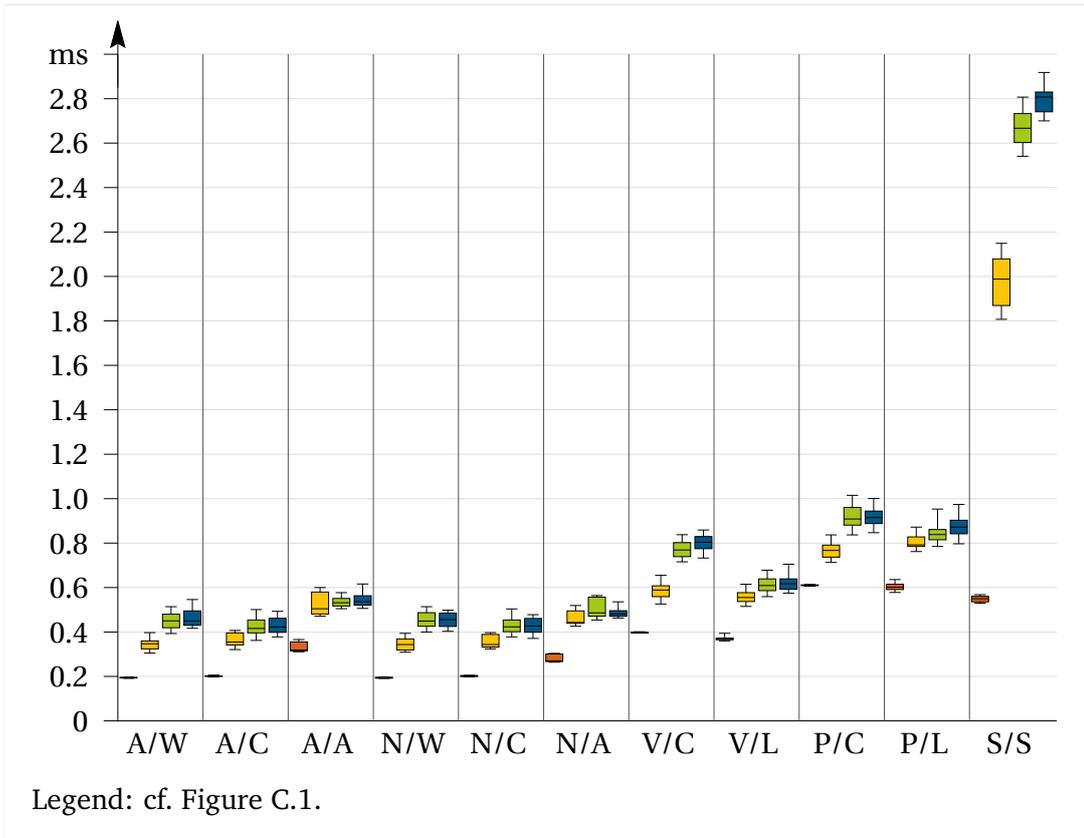


Figure C.4: Measurement results when transferring a 1MB file with 100Mbps.

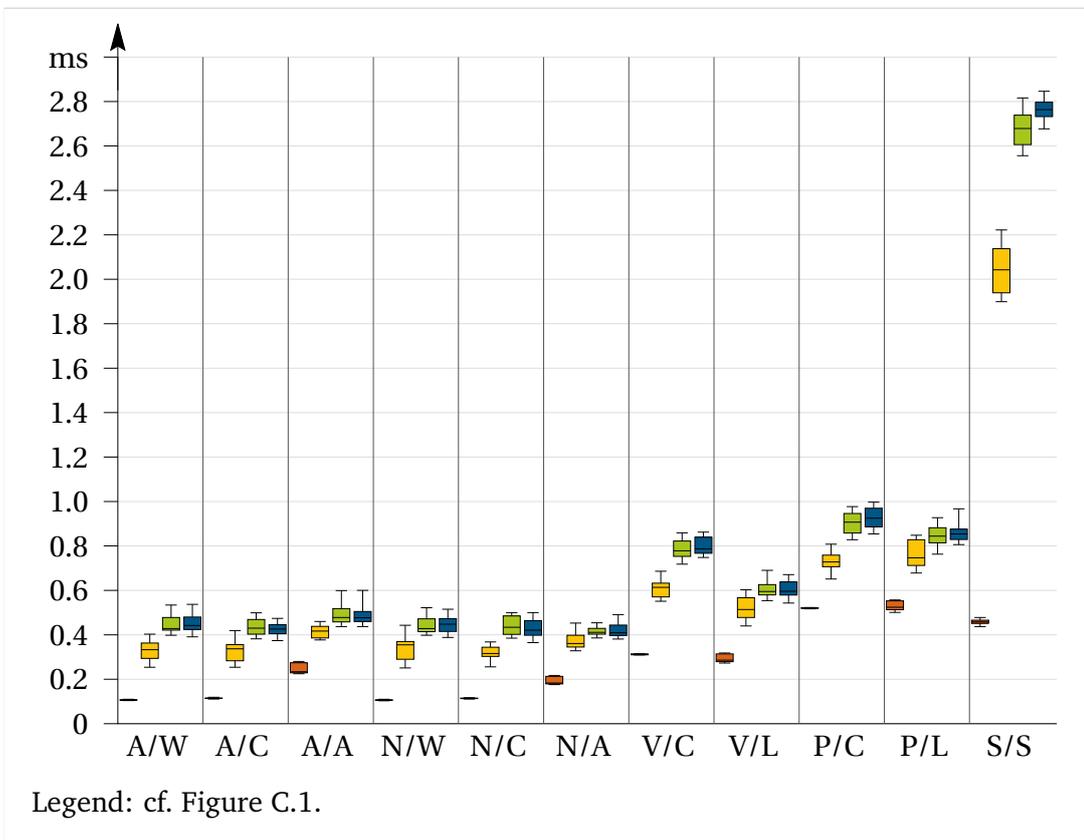


Figure C.5: Measurement results when transferring a 1MB file with 300Mbps.

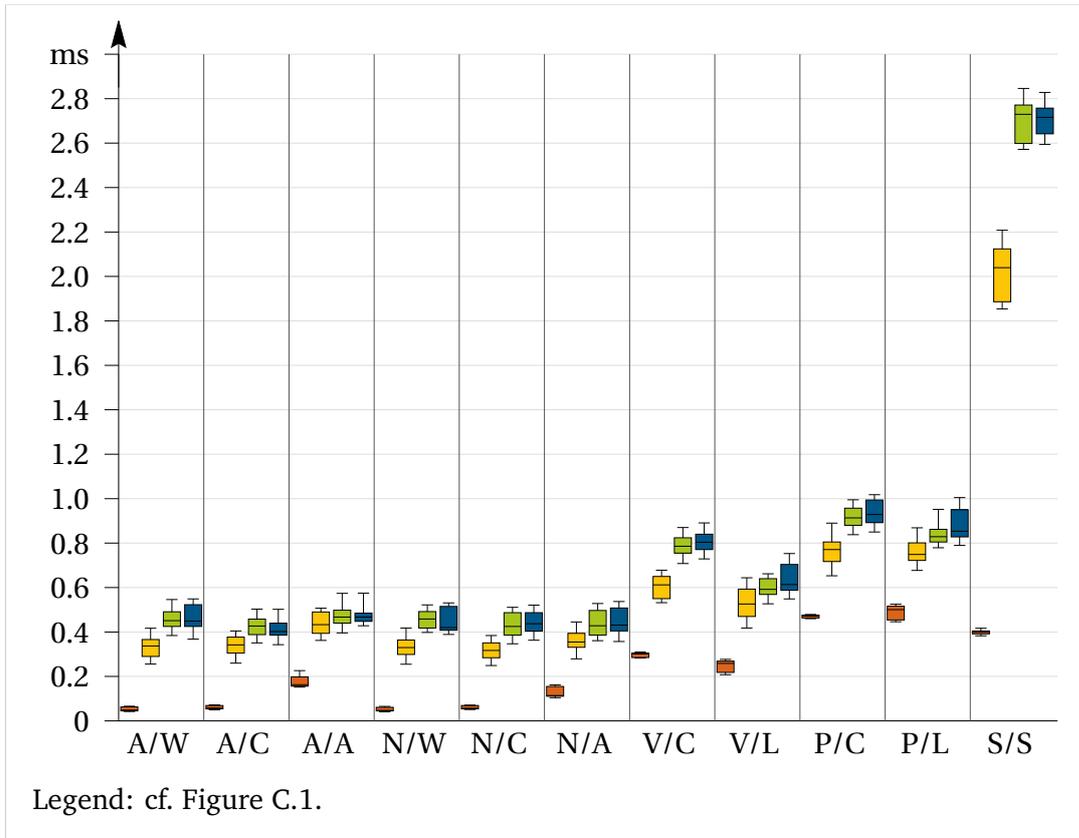


Table C.2: Absolute and relative overall overheads when transferring a 1MB file.

Protocol	Server	Client	Overall overhead () for a bit rate of		
			50Mbps	100Mbps	300Mbps
HTTPS	Apache2	wget	256ms (132%)	335ms (313%)	400ms (816%)
		cURL	220ms (109%)	311ms (270%)	344ms (593%)
		aria2c	217ms (68%)	243ms (104%)	304ms (187%)
	nginx	wget	262ms (135%)	341ms (319%)	373ms (794%)
		cURL	226ms (112%)	307ms (269%)	380ms (667%)
		aria2c	212ms (79%)	226ms (123%)	317ms (278%)
FTPS	vsftpd	cURL	407ms (103%)	475ms (152%)	503ms (167%)
		LFTP	248ms (67%)	310ms (108%)	356ms (138%)
	ProFTPD	cURL	305ms (50%)	405ms (78%)	456ms (97%)
		LFTP	270ms (45%)	330ms (63%)	354ms (71%)
SSH	OpenSSH	SCP	2257ms (410%)	2306ms (503%)	2317ms (581%)

C.3 Transferring Files of Size 128MB

Figures C.6 to C.8 on pages 249 and 250 visualize the performance measurement results presented in Table 5.4 as boxplots. For a bit rate of 50Mbps (Figure C.6) each boxplot is based on 30 repeated measurements, for a bit rate of 100Mbps (Figure C.6) each boxplot is based on 30 repeated measurements, and for a bit rate of 300Mbps (Figure C.6) each boxplot is based on 40 repeated measurements. The plotted medians correspond to the median values presented in Table 5.4.

In addition, Table C.3 (page 250) shows for each client/server combination and for different bit rates both the absolute and the relative performance overhead imposed by a fully functional usage control infrastructure, i.e. when signaling system calls to the Controller, and performing both local and remote data flow tracking (■).

Figure C.6: Measurement results when transferring a 128MB file with 50Mbps.

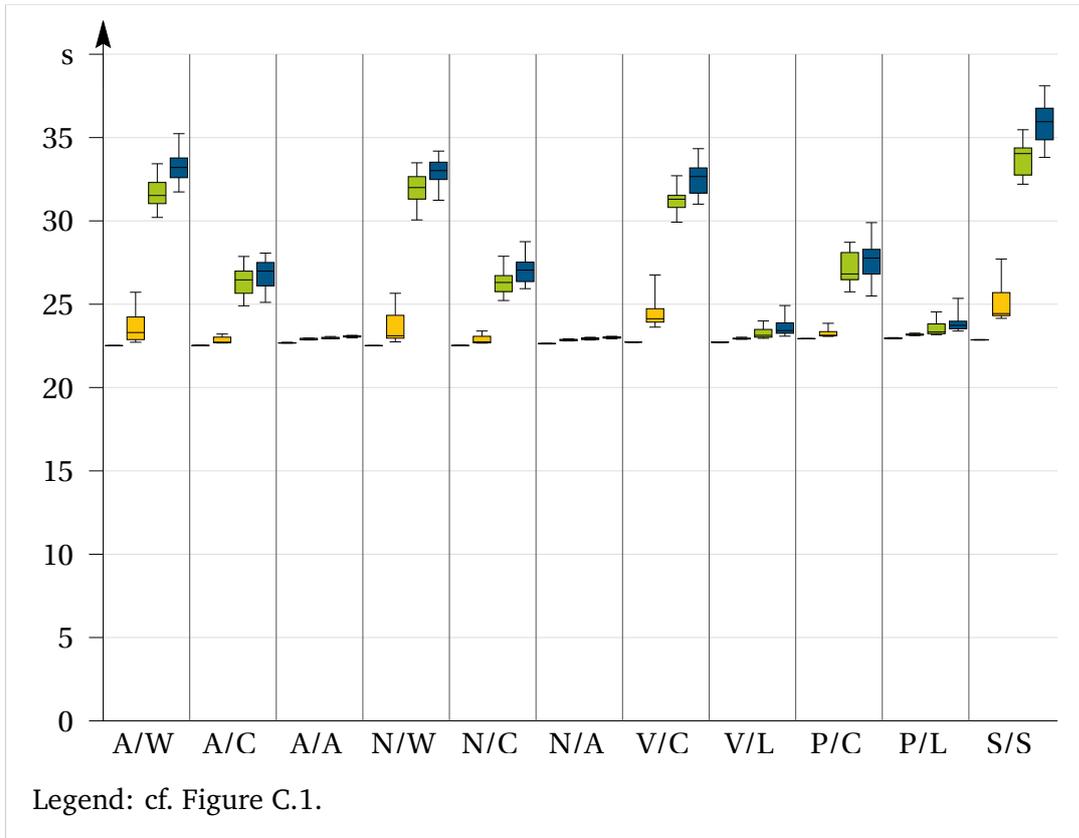


Figure C.7: Measurement results when transferring a 128MB file with 100Mbps.

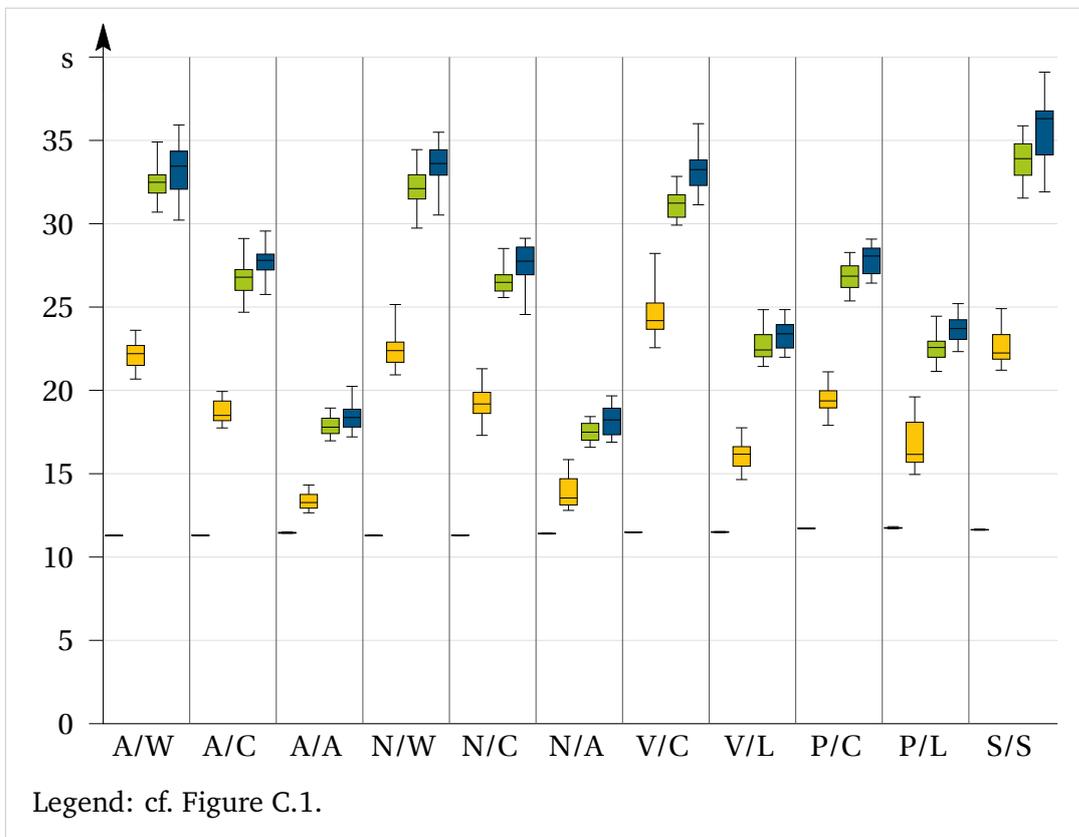


Figure C.8: Measurement results when transferring a 128MB file with 300Mbps.

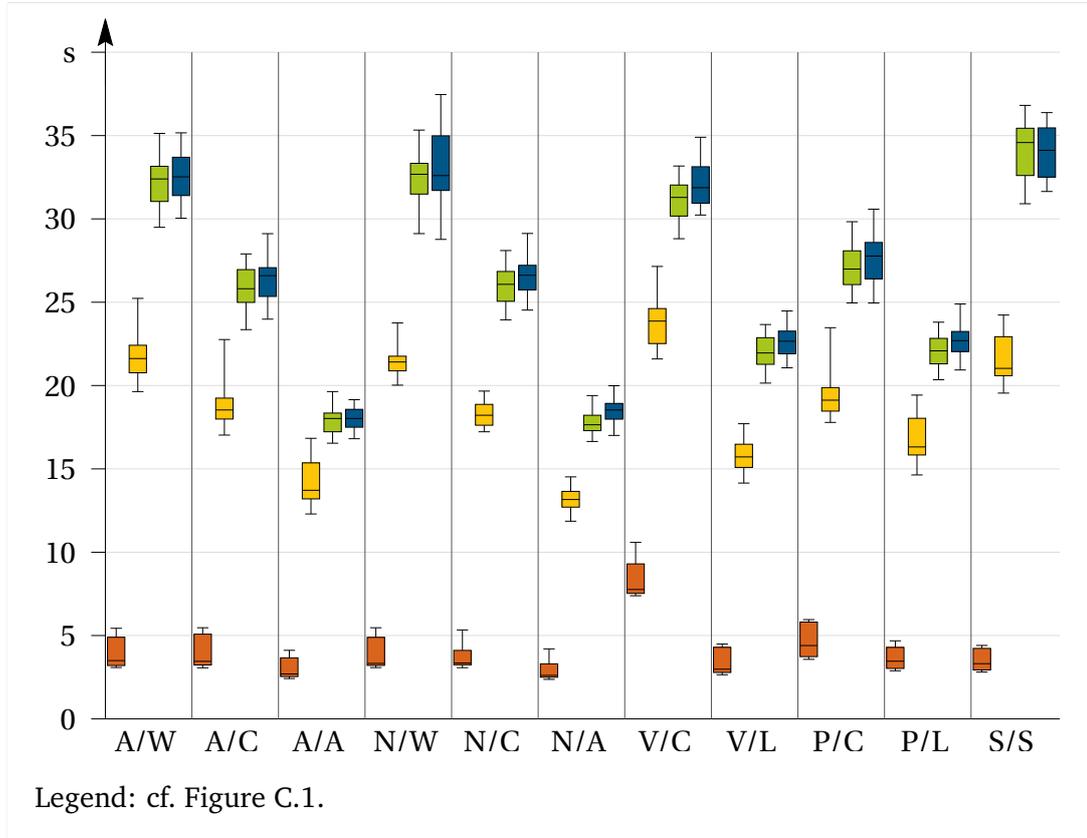


Table C.3: Absolute and relative overall overheads when transferring a 128MB file.

Protocol	Server	Client	Overall overhead () for a bit rate of		
			50Mbps	100Mbps	300Mbps
HTTPS	Apache2	wget	10.69s (47%)	22.15s (196%)	29.03s (829%)
		cURL	4.46s (20%)	16.50s (146%)	23.14s (671%)
		aria2c	0.38s (2%)	6.92s (60%)	15.33s (570%)
	nginx	wget	10.50s (47%)	22.32s (198%)	29.27s (879%)
		cURL	4.53s (20%)	16.45s (146%)	23.27s (695%)
		aria2c	0.38s (2%)	6.83s (60%)	15.92s (610%)
FTPS	vsftpd	cURL	9.93s (44%)	21.76s (189%)	24.10s (310%)
		LFTP	0.70s (3%)	11.90s (103%)	19.68s (660%)
	ProFTPD	cURL	4.84s (21%)	16.34s (139%)	23.37s (531%)
		LFTP	0.77s (3%)	11.97s (102%)	19.23s (556%)
SSH	OpenSSH	SCP	13.09s (57%)	24.67s (212%)	30.80s (931%)

C.4 Transferring Files of Size 512MB

Figures C.9 to C.11 on pages 252 and 253 visualize the performance measurement results presented in Table 5.5 as boxplots. For a bit rate of 50Mbps (Figure C.9) each boxplot is based on 30 repeated measurements, for a bit rate of 100Mbps (Figure C.9) each boxplot is based on 30 repeated measurements, and for a bit rate of 300Mbps (Figure C.9) each boxplot is based on 40 repeated measurements. The plotted medians correspond to the median values presented in Table 5.5.

In addition, Table C.4 (page 253) shows for each client/server combination and for different bit rates both the absolute and the relative performance overhead imposed by a fully functional usage control infrastructure, i.e. when signaling system calls to the Controller, and performing both local and remote data flow tracking (■ ■ ■).

Figure C.9: Measurement results when transferring a 512MB file with 50Mbps.

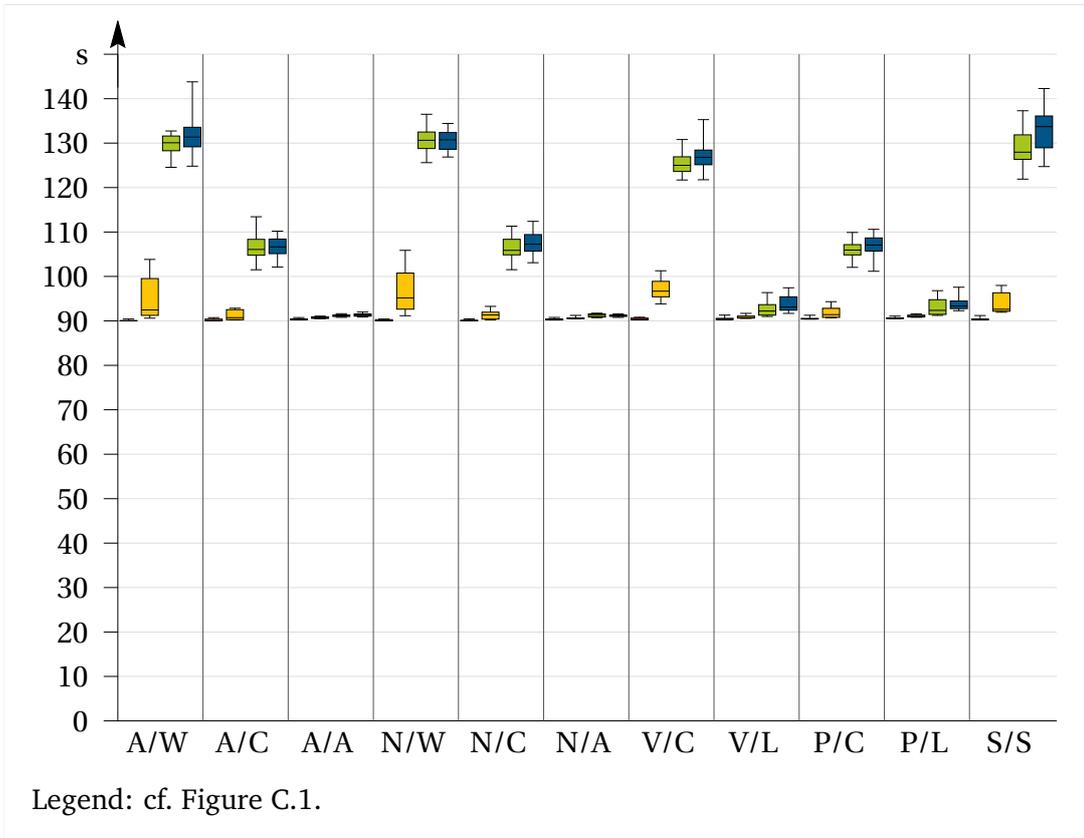


Figure C.10: Measurement results when transferring a 512MB file with 100Mbps.

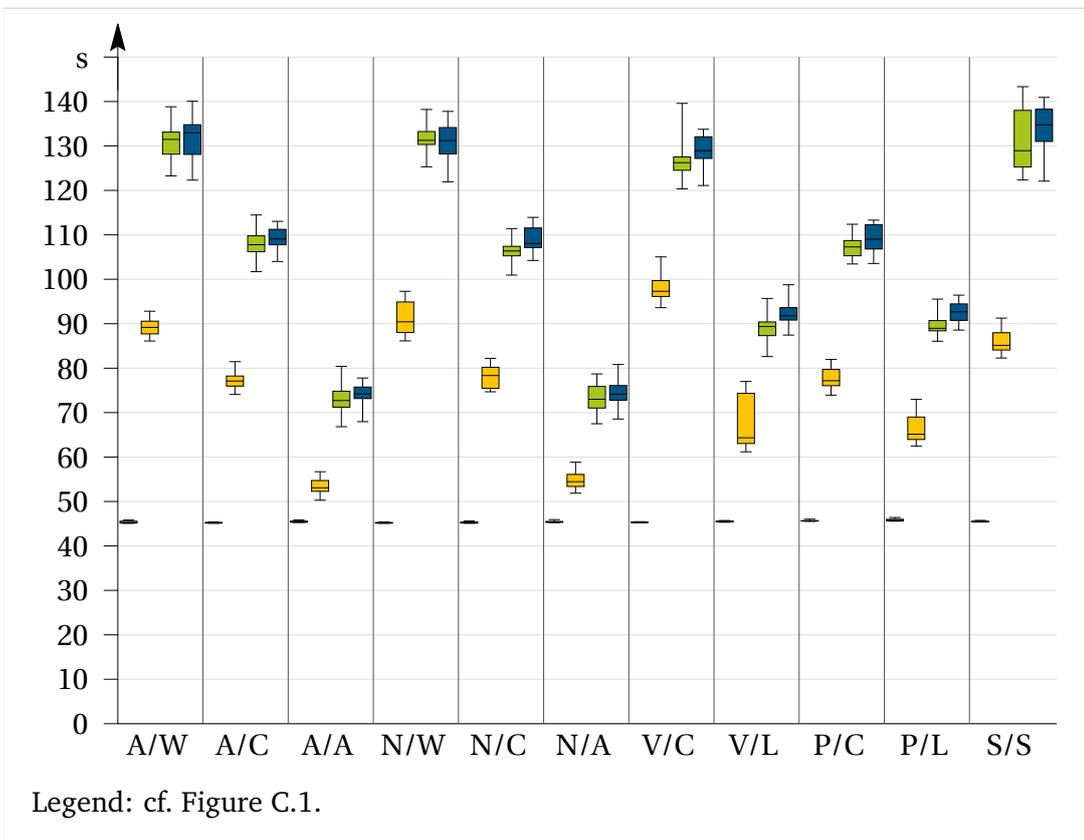


Figure C.11: Measurement results when transferring a 512MB file with 300Mbps.

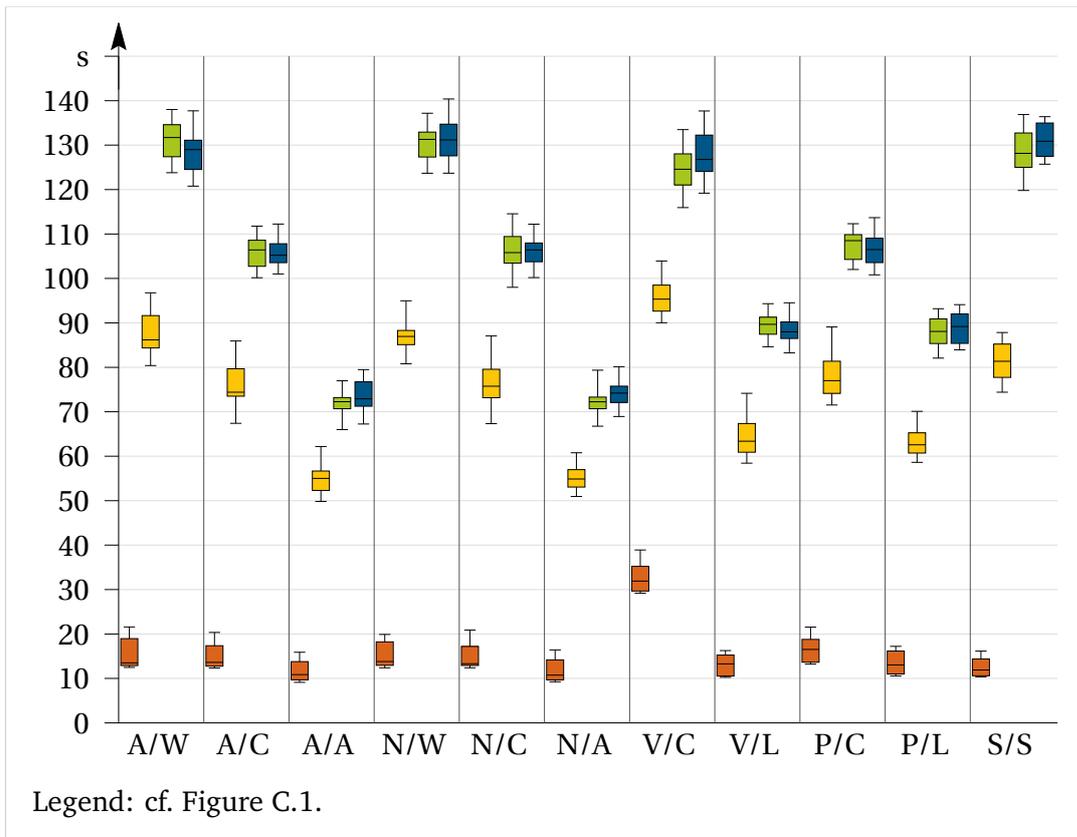


Table C.4: Absolute and relative overall overheads when transferring a 512MB file.

Protocol	Server	Client	Overall overhead () for a bit rate of		
			50Mbps	100Mbps	300Mbps
HTTPS	Apache2	wget	41.36s (46%)	87.71s (194%)	115.50s (855%)
		cURL	16.61s (18%)	63.89s (141%)	91.61s (673%)
		aria2c	0.97s (1%)	28.79s (63%)	62.07s (572%)
	nginx	wget	40.72s (45%)	86.01s (190%)	117.35s (850%)
		cURL	17.23s (19%)	62.89s (139%)	93.09s (699%)
		aria2c	0.85s (1%)	28.77s (63%)	63.46s (590%)
FTPS	vsftpd	cURL	36.48s (40%)	83.62s (185%)	94.87s (297%)
		LFTP	2.77s (3%)	46.31s (102%)	74.71s (563%)
	ProFTPD	cURL	16.64s (18%)	63.37s (139%)	89.96s (544%)
		LFTP	2.78s (3%)	46.95s (103%)	76.13s (583%)
SSH	OpenSSH	SCP	43.39s (48%)	89.32s (197%)	118.98s (1000%)