CONTRACT: Incorporating Coordination into the IP Network Control Plane

Zheng Cai Florin Dinu Jie Zheng Alan L. Cox T. S. Eugene Ng
Department of Computer Science
Rice University

***Abstract*—This paper presents the CONTRACT framework to address a fundamental deficiency of the IP network control plane, namely the lack of coordination between an IGP and other control functions involved in achieving a high level objective. For example, an IGP's default automatic reaction to a network failure may result in an SLA violation, *even if* the IGP link weights have been carefully chosen. This is because an IGP blindly routes traffic along the shortest paths based on link weights, and it is completely oblivious to the interactions between SLA compliance, load balancing and traffic policing objectives in a network. The CONTRACT framework makes it possible to coordinate these objectives. Under this framework, routers continue to operate autonomously, but they also coordinate their actions with a centralized network controller, which evaluates the impact of routing changes, decides whether the changes are SLA compliant, and performs load rebalancing and/or packet filter reconfiguration as necessary. The key contribution of CONTRACT is a set of coordination algorithms. We show that CONTRACT can effectively coordinate the actions of routing, load balancing and traffic policing to improve a network's SLA compliance.**

***Keywords*-network coordination; traffic control; coordination algorithms; SLA compliance**

I. Introduction

Today, a network operator must carefully handle numerous control tasks to ensure that service level agreements (SLAs) are met. First, the operator must perform careful network capacity planning to ensure that the network has enough bandwidth to meet the traffic demand [1]. Second, load-balanced routing is necessary to mitigate network hot spots and to enhance the network's ability to absorb temporary spikes in traffic [2]. Furthermore, in today's hostile Internet environment where a single DDoS attack could generate more than 40 Gbps of sustained unwanted traffic [3], it is crucial to use traffic filters to stop such unwanted traffic from overwhelming the network.

The possibility of network failures further complicates the network operator's task. This is because when a failure occurs, an Interior Gateway Protocol (IGP) such as IS-IS [4] and OSPF [5] will immediately re-route traffic around the failure. Although automatic failure recovery is generally desirable, the re-routed traffic may congest the network *even if* the IGP link weights have already been carefully chosen by a load balancing mechanism. Furthermore, changing routing without regard to DDoS traffic filtering could mistakenly re-route DDoS flows around the filters that aim to block them. The resulting service level agreement violations can be serious and can persist for over 10 minutes [6], even in a tier-1 backbone network.

The fundamental problem is the lack of coordination in the control plane. Specifically, the IGP is allowed to operate in isolation from the SLA compliance, load balancing and traffic policing functions. In reality, however, these functions are intertwined and need to coordinate their actions. To address this problem, we propose the COordiNated TRAffic ConTrol (CONTRACT) framework. In CONTRACT, routers continue to recover from failures in a distributed autonomous fashion. However, the key difference is that routers coordinate their actions with a centralized network controller who is responsible for network-wide control tasks. Numerous studies have experimentally demonstrated the feasibility of using a centralized controller for a variety of network-wide control tasks even for large networks (e.g. BGP routing decision making [7][8], network-wide access control [9], intra-domain routing and packet filter configuration [10], data center network layer-2 routing [11]). In contrast, the novel focus of CONTRACT is to provide a set of algorithms for achieving coordination, thereby improving SLA compliance in the network.

There are three key mechanisms underlying the CONTRACT framework. First, under CONTRACT, routers participate in a distributed coordination protocol with the network controller. The controller programmatically evaluates the impact of the routing changes, decides whether the changes are SLA compliant, and performs load rebalancing and/or packet filter reconfiguration as necessary. Second, because the overall impact of re-routed traffic cannot be locally determined by a router, under CONTRACT, routers temporarily lower the priority of the re-routed traffic, thus protecting other traffic. The priority will return to normal once the changes are deemed SLA compliant by the controller. Finally, CONTRACT enables routers to autonomously adapt their packet filter configuration as routing changes to retain (when feasible) the packet filtering behavior.

The CONTRACT mechanisms work transparently beneath the IGP. Therefore, they can be deployed without changes to the IGP. The CONTRACT coordination protocol guarantees that all routers in the network partition containing the controller reach a consistent coordinated routing state despite arbitrary network failures. Furthermore, if the controller itself has failed or the network has been partitioned, and coordination is no longer possible, the IGP continues to function autonomously; network survivability is thus not compromised.

CONTRACT therefore seamlessly combines the benefits of a distributed IGP with the benefits of sophisticated centralized network-wide control mechanisms.

While our focus for this paper is ensuring coordination between intra-domain routing, traffic policing and load balancing, the basic ideas underlying the CONTRACT framework are applicable to coordinating additional control tasks. For example, BGP routing decisions can congest parts of the network by changing the paths traversed by inter-domain traffic. To prevent SLA violations, these BGP decisions need to be coordinated with the load-balancing and traffic policing functions. Also, coordination is needed for implementing VPN provisioning and quality of service differentiation in the network. The challenges in these scenarios bear similarity to ensuring SLA compliance in that predictable performance needs to be offered to the client. However, it would be impossible to properly address all these topics in a single paper, thus we save them for a follow-up paper.

To evaluate CONTRACT, we conduct experiments across a wide range of network conditions. We are able to show that CONTRACT can enforce the coordination objectives among the IGP, traffic load balancing, and traffic policing functions even under rapid network changes, while consuming reasonable router resources even for large networks. Furthermore, we show that CONTRACT provides substantial improvements to network performance and SLA compliance during network failures.

The rest of this paper is organized as follows. In the next section, we provide further motivation and discuss the related work. In Section III, we present the proposed CONTRACT mechanisms and analyze their properties. We experimentally evaluate the benefits of CONTRACT in Section IV and conclude in Section V.

II. Motivation and Related Work

A. Need for IGP and Load Balancing Coordination

The load on each individual link is determined by two factors: the traffic demand matrix and routing. Previous studies have demonstrated how the traffic demand matrix can be efficiently measured [12][13]. Routing is determined by an IGP (e.g. OSPF, IS-IS). Each individual network link is assigned a link cost and each router runs the IGP. The IGP exchanges link-state announcements among routers to learn the complete topology and link costs of the network. The IGP then distributedly selects a minimum cost routing path.

Therefore, whether a network has well balanced load depends very much on the link cost assignments. Fortz and Thorup [2] were the first to formalize the problem of optimizing link cost assignment for load balancing and proved that the problem is NP-hard. Fortunately, they also showed that a local search heuristic for finding good link costs can perform very well in practice. Follow-on work includes computing link costs that work well across different traffic demand matrices [14].

The main question is, even if a network's load is well balanced initially, will it continue to behave well when the IGP unilaterally recomputes routes after detecting a failure? In an experiment based on the Sprint North American backbone network, Nucci et al. [15] pointed out that when a single link failure occurs, even an initially well-balanced network with maximum link load of 68% can become overloaded with maximum link load of 135%. Interestingly, this overload is not inevitable. If the IGP were coordinated with the link cost selection mechanism, then the maximum link load after this failure can be kept below 90% [15]. Therefore, the coordination between the IGP and load balancing is crucial for maintaining SLA compliance.

B. Need for IGP and Traffic Policing Coordination

According to a recent survey of network operators [3], from Aug 2007 to Jul 2008, the largest DDoS attacks reached 40 Gbps, with 27% of the attacks reaching 4 Gbps or more. Therefore, without the proper policing of such unwanted traffic, even a tier-1 backbone network could become congested.

The filtering or rate limiting of unwanted traffic is implemented by access control rules in routers (or equivalently in specialized middleboxes). What complicates matters is that a router is limited in the number of access control rules it can handle at wireline speed. Network operators have cited the impact of access control lists on network performance as the most serious infrastructure shortcoming [16]. To get around the performance problem, access control rules often get distributed to internal network links as opposed to being implemented entirely at traffic ingress links. Maltz et al. [17] reported that more than 70% of the operational networks they analyzed have access control rules implemented at internal links.

In this environment, unilateral uncoordinated actions by an IGP could lead to severe network congestion because any change to routing could let a large DoS traffic flow bypass the link where the access control rule is implemented. To quantify the problem caused by this poor coordination, we conduct experiments on the 79-node Rocketfuel topology [18]. The goal is to quantify the likelihood of a flow bypassing its access control rule as a result of the unilateral IGP reconvergence after a single link failure. In these experiments, we only consider flows that have at least 5 hops. The network diameter is 10 hops and the average path length for all these flows is 5.8 hops. We subject 2645 flows to access control rules placed N hops from the ingress link, where N varies from 1 to 3.

For 10% of the link failure scenarios, there are more than 90, 156 and 173 flows bypassing access control rules when the rules are placed at the 1st, 2nd, and 3rd hop respectively. In the worst scenario, there are 373, 666 and 739 flows bypassing access control rules. If an IGP were able to coordinate its actions with the configuration of access control rules, permitting new rules to be configured when routing changes, then a DoS flow need not bypass its access control rule.

C. Related Work

There are a number of routing approaches for improving a network's SLA compliance under failures if coordination is not available. Nucci et al. [15] developed techniques to compute a single set of link costs that achieve good load

balance both during normal operation and after any single link failure. Although this work represents a breakthrough, its scope is restricted to single link failures. The jury is still out on whether a single set of link costs can achieve good load balance for other common types of failures, such as linecard failures and router failures.

If routing is not restricted to link-state IGP, that is, if MPLS routing is employed, then nearly optimal routing that is oblivious to traffic demand can be computed [19][20][21]. Moreover, with MPLS routing, Applegate et al. [22] showed that by carefully choosing the failure restoration paths, nearly optimal performance after a network failure can be achieved even with little knowledge of traffic demand. However, computing restoration paths in advance for all possible failure scenarios is computationally expensive. Furthermore, MPLS routing is not as widely used in practice as IGP routing.

In contrast, the CONTRACT framework is aimed at improving SLA compliance regardless of the type of network failure experienced. Furthermore, the CONTRACT framework takes SLA compliance, routing, load balancing, and traffic policing into account holistically, which is not possible with the previous routing only approaches.

The dependency between traffic policing and an IGP as a potential security problem has been known for a long time [23]. The two can be decoupled if traffic policing is pushed to the very edge of the network where there are natural traffic choke points [24]. However, as discussed, routers have limited ability to support access control rules and these rules in practice are often distributed to internal network links [17]. Implementing redundant access control rules along the potential fail-over paths of traffic may help guard against some problems but will require precious router computation resources that may not be available. Our coordination mechanisms prevent unwanted traffic from bypassing access control rules even when the rules are distributed to internal network links.

It is theoretically possible to avoid the need for coordination by throwing away traditional IGPs and re-engineering routing, load balancing, and traffic policing into one system as advocated by the 4D proposal [25] – in effect re-engineering the entire IP network control plane. However, we believe it is equally important to solve the coordination problems without requiring drastic changes to the control plane. Our motivation is both fundamental and practical. Fundamentally, no solution exists for the coordination problem within today's control plane and therefore it represents an unexplored point in the design space. Furthermore, solving this problem within today's control plane sheds light on how an IGP and other distributed protocols could better co-exist with other functions in the control plane. Practically, throwing away traditional IGPs may not be a viable choice for network operators because of subtle dependencies that may already exist. For example, when businesses are merged, integrating their existing infrastructure requires a significant amount of "glue logic" such as the use of route redistribution [26]. The dependencies entailed by this delicate glue logic make it even more impractical to drastically change the control plane.

III. CONTRACT: The Framework

CONTRACT works with link-state IGPs, including OSPF [5] and IS-IS [4]. It does not modify the IGPs. For simplicity, we will describe CONTRACT in terms of OSPF. We assume the reader is already familiar with OSPF. The purpose of CONTRACT is to ensure that both the load balancing and traffic policing objectives are taken into account during IGP reconvergence. The basic idea of CONTRACT is that, new routing entries generated by the IGP are installed immediately, but put in the unapproved mode. Traffic routed using these unapproved entries is put in low-priority queues, and tends to be dropped first when there is congestion. At the same time, routers send approval requests to the CONTRACT controller for evaluation. Furthermore, routers locally adjust their filter configuration to cope with the routing changes. The controller participates in the link state routing, so it also receives all LSAs (link state advertisement) flooded in the network. Only routing entries which do not violate coordination objectives are approved by the controller, and be brought back to the approved mode (where traffic is routed with a normal priority). In addition, the controller also recomputes the filter configurations for routers accordingly and try to balance the load in the network by optimizing the link cost assignment.

A. IGP and Load Balancing Coordination

CONTRACT assumes that the controller knows the traffic matrix in the network. The traffic matrix is needed to evaluate routing changes and to optimize the link cost assignment. Next we give detailed explanations about the notations we use.

Notations and Explanations:

- $seq_n(t_i)$ denotes the sequence number each router n maintains at time t_i. It increases by 1 when a router's local link state changes. This number is contained in the LSA flooded by each router. For another router m, once it receives such a LSA, it will remember that sequence number in its link state database as $seq_n^m(t_i)$. This is the sequence number of router n from router m's perspective. $seq_n^n(t_i)$ is equivalent to $seq_n(t_i)$. The sequence number serves to uniquely identify each instance of the local link state of each router in the network.
- $x_n(t_i)$ denotes the network-wide link state from router n's perspective at time t_i. $x_n(t_i)$ is the link state database of router n which also contains the $seq_m^n(t_i)$ it has observed from any other router m. If at time t_j, all the routers and the controller reach a consistent state, where $\forall a, b, x_a(t_j) = x_b(t_j)$, we use $X(t_j)$ to denote this consistent network link state.
- $(HASH = SecureHash(x_n(t_i)), SEQSUM = \sum_m seq_m^n(t_i))$ denotes the fingerprint of state $x_n(t_i)$ in router n. Letting routers send actual routing tables to the controller for evaluation is an unnecessary overhead. In CONTRACT it is more efficient for the central controller to evaluate the network link state $x_n(t_i)$ instead. The fingerprint further compresses and identifies each unique network link state, and presents an ordering of network

For Each Router
On local link state changes or receiving new LSAs at time t_i:
Update the local link-state database;
Compute $rt(x_n(t_{i-1}), x_n(t_i))$;
Locally_adjust_filter_configuration(...);
//This function will be expanded in next subsection
Update the router's routing table by $rt(x_n(t_{i-1}), x_n(t_i))$;
For each e in $rt_{insert}(x_n(t_{i-1}), x_n(t_i))$
$fp(e) = (SecureHash(x_n(t_i)), \sum_m seq_m^n(t_i))$;
Flag these entries as $Unapproved$ (traffic will have low priority);
Send $AprReq(x_n(t_i))$ to the controller;

Fig. 1. Local autonomous adaptation algorithm

link states. The first element is generated by a secure hash function (e.g. MD5, SHA-1, SHA-2, etc.) which computes on an array buffer that contains all $seq_m^n(t_i)$. This value uniquely identifies the network link state in router n at time t_i. The value $\sum_m seq_m^n(t_i)$ provides a local ordering of network link states. In any particular node in the network (either a router or the controller), a state with a smaller $\sum_m seq_m^n(t_i)$ is older than a state with a bigger one. This value does not ensure a global ordering. For a fingerprint fn, we use $fn.HASH$ to specify the secure hash value in that fingerprint, and $fn.SEQSUM$ to specify its sum of sequence numbers. $fingerprint()$ denotes the function we use to generate the fingerprint of a network state.

- $rt(x_n(t_i))$ stands for the routing table of router n, generated by OSPF based on state $x_n(t_i)$. $RT(X(t_i))$ denotes all routing tables of all routers in the network, corresponding to a consistent state $X(t_i)$. $rt(x_n(t_{i-1}), x_n(t_i))$ stands for the changes in the routing tables in router n from state $x_n(t_{i-1})$ to $x_n(t_i)$.
- For efficiency, the routing table is modified gradually by insertions and deletions. $rt_{delete}(x_n(t_{i-1}), x_n(t_i))$ denotes the entries bound for deletion, and $rt_{insert}(x_n(t_{i-1}), x_n(t_i))$ denotes new entries that are going to be installed. Updates can be realized by deletions followed by insertions. For each entry in the routing table, we remember the fingerprint of the network link state for which it is inserted. $fp(e)$ denotes such a fingerprint, where e is one entry.
- $AprReq(x_n(t_i))$ denotes the approval request sent to the controller by router n via unicast, for the routing table associated with the new link state $x_n(t_i)$. For brevity, we will loosely refer to this as an approval request for the link state $x_n(t_i)$. It contains the router's ID, and the fingerprint $(SecureHash(x_n(t_i)), \sum_m seq_m^n(t_i))$.
- When the controller approves a routing table associated with some link state, the approval $Apr(X(t_i))$ is reliably flooded hop-by-hop into the network. For brevity, we will loosely refer to $Apr(X(t_i))$ as an approval for the link state $X(t_i)$. The controller only approves consistent state. The approval message contains the fingerprint of that state. For brevity we use approving link state to refer to the approval of the routing table associated with that particular link state.

For Each Router
On receiving $Apr(X(t_i))$
For each entry e in its current routing table
if $(fp(e).SEQSUM <= Apr(X(t_i)).fingerprint.SEQSUM)$
Approve this entry(traffic will be normal priority);
else
Keep it $Unapproved$;

For The Central Controller
On receiving $AprReq(x_n(t_i))$
FingerprintTable[n] = $AprReq(x_n(t_i)).fingerprint$;
Check all fingerprints in FingerprintTable to see whether
they are consistent with the controller's own fingerprint;
if (consistent)
Evaluate($X(t_i)$;
if (approved)
Send out $Apr(X(t_i))$;

On receiving new LSAs
Update the link-state database;
Generate and send out new optimized link weights if necessary;

Fig. 2. Distributed coordination protocol for IGP routing

The Algorithms:

The CONTRACT framework is composed of two algorithms. The first algorithm works locally at a router and allows it to autonomously adapt to network changes. The second algorithm coordinates the routers and the controller.

Figure 1 shows the specifications of the autonomous adaptation algorithm in routers. When one LSA is received, OSPF on each router will compute necessary routing entry changes, update their fingerprint, and put them in the unapproved mode.

Figure 2 shows the specifications of the distributed coordination protocol. When a router receives an approval, it searches through its routing entries, and approves all entries with a fingerprint older than or exactly the same as the one in the approval message. This effectively approves all routing entry changes that have accumulated up to the state specified in the approval message. When the controller receives one approval request, it first checks whether all nodes in the network have reported the same fingerprint (which means they have reached a consistent network link state), and if so it goes ahead and evaluates that network link state to see whether the changes can be approved. In this case, the controller sends out approval messages. When the controller receives new LSAs, it runs an optimization algorithm to generate better link weights if possible. The optimization algorithm can have different objective functions. As an example, in this paper it minimizes the total number of flows that are affected by packet loss. When routers receive the new link weights, they will generate the corresponding routing changes, and the changes will be evaluated and approved by the controller.

Router State Invariant:

Because of the time that the controller takes to evaluate a network link state and the delays in the network, the approval message might take an arbitrary amount of time to reach every router in the network. However, we show that any router's state does not become arbitrarily complex but rather it satisfies a simple invariant at all time. Let us assume that we start with the network state $X(t_0)$ in which

every routing entry is approved. Then, before $Apr(X(t_i))$ arrives, a router could already reach state $X(t_{i+k})$. Based on the coordination protocol, $Apr(X(t_i))$ will only approve the routing entries resulting from states ranging from $X(t_0)$ to $X(t_i)$. The routing entries that are generated corresponding to network state from $X(t_{i+1})$ to $X(t_{i+k})$ will all remain unapproved. Therefore, a router's state satisfies at all time the invariant that it always consists of an approved state followed by zero or more unapproved state changes, no matter how long the approval messages are delayed. In this case, after $Apr(X(t_i))$ has been applied, the router's state is $X(t_i)$ followed by $X(t_{i+1})...X(t_{i+k})$.

In addition, approval messages may arrive out of order. At time t_{i+k}, $Apr(X(t_{i+a})), a \leq k$ may arrive before $Apr(X(t_i))$. $Apr(X(t_{i+a}))$ will approve all the entries that are the results of network state from $X(t_0)$ to $X(t_{i+a})$. When later on $Apr(X(t_i))$ arrives, it becomes a no-op. As a result, such out-of-order approval message processing is equivalent to advancing the router's state to $X(t_{i+a})$ followed by $X(t_{i+a+1})...X(t_{i+k})$. The invariant is still preserved.

Discussion:

In order to evaluate a consistent network link state, CONTRACT requires all routers to report that state. If the network link state changes very fast, such a consistency may not be reached. In this case the controller cannot evaluate and approve any of these states, so eventually all routing entries will become unapproved and all traffic will receive the same low priority. This is one limitation of CONTRACT. We will evaluate this effect in Section IV.

CONTRACT can also be applied to networks where equal-cost multipath routing is used, as long as the ratio with which the traffic is distributed on the equal-cost paths is known by the controller. In this situation, the controller can still predict the traffic distribution in the network.

OSPF creates separate routing entries for each unique destination prefix. Then, the router performs CIDR aggregation on these routing entries and configures the hardware forwarding table entries. Because unapproved routing entries are treated with low priority, when doing the CIDR aggregation, only approved entries can be merged with approved entries, and only unapproved entries can be merged with unapproved entries.

B. IGP and Traffic Policing Coordination

Filter rules are not only used to block malicious traffic, but also configured for traffic shaping. In general, filter rules for specific traffic flows are configured in the network along the path where the flows are routed.

When the network link state changes, traffic flows could be rerouted and thus bypass some filter rules. The controller always tries to adjust filter configurations according to network link state changes. However, since it takes time for the controller to generate and send out new filter configurations, there could be transient periods where the filter rule semantics are not preserved. Therefore, we propose that in addition to coordinating with the controller, on link state changes, routers should locally adjust their filter rule configurations based on the locally observed behavior of traffic policing

At each router, for all the traffic flows that go through the router, the router can observe what filter rules are applied on which traffic flows. This observed traffic flow and filter rule relation defines the *local filter semantics* at the router. A router seeks to preserve these semantics when the network state changes. The *global filter semantics* of the whole network is the traffic flow and filter rule relation that the controller wants to enforce.

Requirements:

First, because filters can be installed on inbound links, to know which inbound link some traffic is going to take, a router needs to know the routing state of the entire network. As a result, a router not only needs to compute the local routing table $rt(x_n(t_i))$, but also needs to compute all-pair shortest path routing state of the entire network, based on its current link state database. This computation can be efficiently performed using a dynamic incremental shortest path algorithm. A router only needs to manage the approval state for its local routing table $rt(x_n(t_i))$, hence the algorithms in the previous section can be readily used. The global routing table is kept separated.

Second, the algorithm requires that the controller always generates exact traffic filters for an approved network link state. By "exact" we mean that the source and destination address ranges of the filter generated by the controller should be equal to or smaller than the address ranges of the traffic that actually travels through the link where the filter is going to be installed. Exact filters precisely define the filter semantics for one router for one routing state. If a filter is exact, and the traffic it matches is rerouted to another link (either inbound or outbound link), when the filter is moved to that new link, it will still match the same traffic. Therefore, the local filter semantics can be preserved by locally adjusting filter configurations.

Notations and Explanations:

- $filter_{current}(x_n(t_i))$ denotes the filter configuration on router n for a network link state $x_n(t_i)$. It can contain filters both generated by the controller and by the router locally.
- $filter_{central}(n, X(t_i))$ denotes the filter configuration generated by the controller for router n for an approved network link state $X(t_i)$. $filter_{central}(X(t_i))$ denotes the collection of filter configurations generated for all the routers. Notice that the controller will only generate filter configuration for an approved state.
- For each filter f in a router, we also associate it with a fingerprint, to remember for which network link state this filter is generated. We use $f.fingerprint$ to denote this fingerprint.
- $f.link$ stands for the link where filter f is installed. $f.toremove$ is a flag used to mark the filters that will be removed. By default it is set to false.

The Algorithms:

Figure 3 expands the function for locally adjusting the filter configuration that was mentioned in figure 1 in the previous

```
Locally_adjust_filter_configuration(new state x_n(t_k))
  For each filter f in filter_current(x_n(t_i))
    fls = all potential traffic matched by f given x_n(t_i);
    fls_changed = all potential traffic in fls that do not go through f.link
        given x_n(t_k);
    fls_unchanged = fls - fls_changed;
    if ( fls_changed != Empty )
      Split f into f_changed for fls_changed and f_unchanged
          for fls_unchanged;
      Install f_unchanged on link f.link;
      if (f.link is an inbound link)
        Install f_changed to the new inbound link(s) of fls_changed;
      else
        Install f_changed to the new outbound link(s) of fls_changed;
      f_changed.fingerprint = fingerprint(x_n(t_k));
      f_unchanged.fingerprint = fingerprint(x_n(t_k));
```

Fig. 3. Specification of Locally_adjust_filter_configuration(...)

```
For Each Router
  On receiving filter configuration filter_central(n, X(t_i))
    For each filter f in filter_current(x_n(t_k))
    //k ≥ i, t_k is the current time
      if (fp(f).SEQSUM < fingerprint(X(t_i)).SEQSUM)
        f.toremove = true;
    For each filter f in filter_central(n, X(t_i))
      Install f on link f.link;
      f.fingerprint = fingerprint(X(t_i));
    Remove all filters with f.toremove == true;
    Locally_adjust_filter_configuration(x_n(t_k));
```

Fig. 4. Actions to be taken when receiving filter configuration

subsection. In this function, on receiving new link state, each router checks each of the filter entries to see whether the flows that they match have been rerouted based on the IGP routing changes. If so the router puts a new filter on the new path (either inbound or outbound). The old entries will be split or removed if necessary, and the new entries will be marked with the fingerprint of the new link state.

Figure 4 specifies the actions to be taken when a router receives filter configuration from the controller. The router removes any filter entries with a fingerprint older than the fingerprint of the new filter configuration from the controller, installs the new filters and locally adjusts them if necessary, using the function shown in figure 3.

Router State Invariant:

At the beginning, in the network state $X(t_0)$, every routing entry is approved, and every filter entry in $filter_{current}(X(t_0))$ is configured by the controller. Before $filter_{central}(X(t_i))$ and $Apr(X(t_i))$ arrive, the network could already reach state $X(t_{i+k})$. Then, after $filter_{central}(X(t_i))$ and $Apr(X(t_i))$ arrive, all filter entries generated for network state $X(t_a), a < i$ will be removed, and $filter_{central}(X(t_i))$ will be installed. Filter entries locally generated for state $X(t_{i+j}), j = 1, 2, ...k$ are locally adjusted based on $filter_{central}(X(t_i))$. These update rules preserve the invariant that a router's state always consists of an approved state followed by zero or more unapproved state changes.

Discussion:

Whether local filter configuration adjustments preserve the global filter semantics depends on where the filter is installed with respect to the location of the routing change. If the routing change happens at a router downstream of the filter rule, then the filter need not be adjusted, and the global filter semantics are preserved. If the routing change happens at the router where the filter rule is installed, then by locally adjusting the filter configuration, the global filter semantics can be preserved. However, if the routing change happens at a router upstream of the filter rule, then even if the filter configuration is locally adjusted, the global filter semantics may not be preserved.

As a result, the local action at a router is only a best effort solution, and it does not always ensure that the global filter semantics are preserved. Nonetheless, new filters are computed by the controller and sent to routers, so the global filter semantics can be re-established. However, it takes time for the controller to reach every router, so the local action at a router helps to reduce the convergence time because it has an immediate effect.

C. CONTRACT Properties

Consistency Property:

CONTRACT ensures that an approval message conveys an endorsement of the routing actions corresponding to a consistent network link state which is known to have been experienced by the controller and all routers in the network. Thus, the resulting approved routing tables in the network are guaranteed to be consistent with the approved link state.

If routers experience different intermediate connectivity states because they experience different connectivity update orderings, the inconsistent intermediate approval requests will never be evaluated by the coordination protocol. Only an eventual set of consistent approval requests would be evaluated.

Furthermore, since the approval message is reliably flooded, routers in any network partition must either all get the approval message or none of them gets the message. Thus, in the event of a network partition during the approval process, every network partition is still internally consistent.

Survivability Property:

Even if the controller fails, or is partitioned from the rest of the network, all the routers will continue to function autonomously, and thus the survivability of the network is not affected. Routers continue to adjust autonomously, such as putting new routing configurations in low-priority mode and trying to preserve local filter semantics while the controller is unavailable. When the controller becomes available again, the CONTRACT coordination mechanisms resume.

IV. Evaluation

In this section, we evaluate the performance and overhead of CONTRACT.

A. Methodology

We use a Java implementation of the CONTRACT framework and an extended version of the ns-2 simulator to conduct packet level simulations. The ns-2 simulator was augmented to operate under the CONTRACT framework. The ns-2 routers

support communication with the controller, and are configurable. Specifically, the controller can install link costs and configure filters. Support for CONTRACT control messages was also added. Since the controller needs to take part in OSPF, it is represented by a router in the ns-2 simulation. We use different Rocketfuel topologies [18] in our evaluation as they provide us with a wide range of scenarios to test our framework.

For the optimization algorithm we use an approach based on a simplex downhill search [27]. Although the results obtained from this algorithm are hardly optimal, we can already see noticeable benefits in fulfilling the objective of the controller. With more sophisticated methods, the performance (both in terms of optimality and computation time) can be further improved. The link cost optimization is a separate process running in parallel to the approval evaluation process.

We put $0.05 \times n \times (n-1)$ randomly chosen best effort traffic flows in the network, where n is the number of nodes in the network. CONTRACT will try to protect these best effort flows from network congestion. At the same time, five malicious flows are set in the network, with a high flow rate (200% of link capacity), to simulate DoS attacks. We introduce different failures in the network, and we compare the performance of CONTRACT to an uncoordinated IGP (OSPF), which we call "No Coordination". For fair comparison, before the failures, we let the controller to generate the same link cost weights and packet filters for both CONTRACT and No Coordination, so at the beginning the network load is balanced, and no malicious flow is leaking.

We use two metrics for evaluating performance. The first metric, "Loss-Num", is the coordination objective of the controller: the number of best effort flows which have packet loss. For the second metric, we assume there is one SLA which covers all best effort flows. This SLA guarantees that the end-to-end delay experienced by packets of these flows is below a threshold. We vary this threshold by multiplying the minimum propagation delay by a variable factor. As the second metric, "SLA-V", we measure the fraction of best effort flows which have SLA violations during the experiments.

In addition to evaluating the performance of CONTRACT, we also evaluate its overhead by varying the size of the network and the frequency of changes to the network.

B. Environment Variables

Here we list all network environment variables that we vary.

- Failure scenario: we try single link and single node failures in the network.
- Average flow rate: source/destination pairs are randomly chosen in the network, and best effort flows with different average rates are created between them. This average flow rate is represented as a percentage of link capacity, and it determines the load level of the network.
- Variance of flow rates: We generate different distributions of the best effort flow rates based on the Pareto distribution. We choose the K value to be 1.1, 2, 4, and 10, where K=10 is closer to a uniform distribution, while K=1.1 is more uneven.
- Noise level of traffic matrix: in a perfect situation, the controller can know exactly the traffic matrix of the flows in the network. However this is not always true, so we introduce Gaussian noise in every non-zero point of the traffic matrix. The standard deviation (as a percentage of the average flow rate) of the Gaussian noise is called the "noise level".
- Optimization time budget: if we allow the optimization algorithm to spend more time balancing the load, it might generate a better link cost assignment that helps reduce congestion, but it also increases the response time. So we give the optimization algorithm a bounded time budget and vary it.
- Link state (LS) routing hold down timer: it is common that a link state routing protocol has a hold down timer to reduce computation overhead and decrease the number of updates to the routing table. Such a timer in our simulated LS routing protocol can also increase the simulation speed. This timer decides the OSPF convergence time.

C. Performance Evaluation

For the performance evaluation we use the 79 node Rocketfuel topology. Since different failure scenarios can cause totally different behavior, in this subsection we analyze all possible failure scenarios we described. We limit the link capacity to 1Mb in order to keep the simulation time tractable. We choose a set of default parameters for the environment variables, and vary one variable at a time in each of experiments to show the effect that variable has on the performance of CONTRACT.

By default we choose 4% of link capacity as the average flow rate, because failures could cause congestion in the network, while the network is not heavily congested; we choose K=10 as the variance of flow rate, which is close to a uniform distribution, but with some variance; we choose a 5% noise level in the traffic matrix, which represents a relatively small noise level; we choose a 2 second optimization time budget because for most cases it can generate good if not optimal link costs; we choose a 1 second hold down timer, which is typical in OSPF.

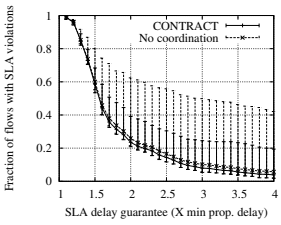
Varying Average Flow Rate:

In this set of experiments we evaluate the effect of different average flow rates on the performance. We use 1%, 2%, 4%, and 10% of link capacity in four groups of experiments.

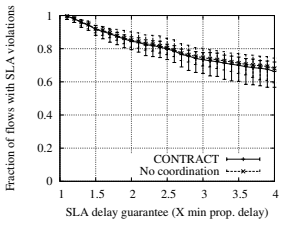
Table I shows the results for the Loss-Num metric. In all these experiments, CONTRACT shows obvious benefits. Specifically, in CONTRACT there is no malicious flow that ever bypasses the filters and gets leaked into the network, while for the No Coordination case, for some failure scenarios there are leaked malicious flows which cause congestion in the network. When the average flow rate is very high (10% of link capacity), the network is so congested that CONTRACT cannot do too much to make the situation better, thus the benefit is reduced.

Scenario	Overall			With leaking malicious flows		
	Avg	Min	Max	Avg	Min	Max
CONTRACT 1%	5.6	0	59	-	-	-
No Coordination 1%	10.9	0	143	67.2	24	143
CONTRACT 2%	6.1	0	63	-	-	-
No Coordination 2%	12.8	0	167	80.9	30	167
CONTRACT 4%	7.2	0	77	-	-	-
No Coordination 4%	15.4	0	176	92.6	34	176
CONTRACT 10%	197.8	178	229	-	-	-
No Coordination 10%	207.9	182	236	219.0	190	236

TABLE I
NUMBER OF FLOWS WITH PACKET LOSS FOR VARYING AVERAGE FLOW RATE



(a) 2% average flow rate



(b) 10% average flow rate

Fig. 5. Number of SLA violations vs. SLA delay guarantee in terms of multiples of minimum propagation delay

Figure 5 plots the results for the SLA-V metric, for average flow rates of 2% and 10% of link capacity. The line is the average value, while the upper and lower bar are the max and min values. In the 2% case, CONTRACT not only reduces the average fraction of violations, but also sharply reduces the maximum fraction of violations, compared to No Coordination. In the 10% case, even though the benefit of CONTRACT is smaller, it is still better than No Coordination, especially in reducing the minimum fraction of violations.

Varying Variance of Flow Rate:

In this set of experiments, we vary the variance of the flow rate distribution with K=1.1, 2, 4 and 10.

Scenario	Overall			With leaking malicious flows		
	Avg	Min	Max	Avg	Min	Max
CONTRACT K=1.1	5.0	0	63	-	-	-
No Coordination K=1.1	8.5	0	110	64.1	13	110
CONTRACT K=2	12.1	0	84	-	-	-
No Coordination K=2	20.3	0	174	88.8	33	174
CONTRACT K=4	7.2	0	89	-	-	-
No Coordination K=4	16.7	0	159	96.8	34	159
CONTRACT K=10	7.2	0	77	-	-	-
No Coordination K=10	15.4	0	176	92.6	34	176

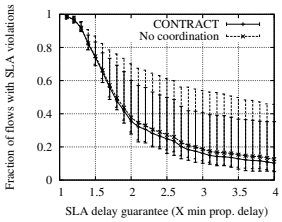
TABLE II
NUMBER OF FLOWS WITH PACKET LOSS FOR VARYING VARIANCE OF FLOW RATE

Table II shows the results for the Loss-Num metric. For different variance in the rates of the traffic flows, CONTRACT always performs better than No Coordination in reducing the number of flows with packet loss.

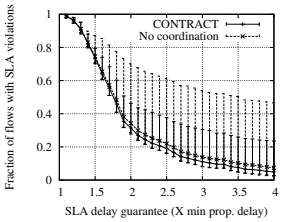
Figure 6 plots the SLA-V, for K=2 and K=10. Again, CONTRACT shows obvious benefits over No Coordination.

Varying Noise Level of Traffic Matrix:

In this set of experiments, we vary the noise level with 5%, 10%, 50%, and 100% of average flow rate. The initial link costs generated for both CONTRACT and No Coordination are based on the traffic matrix with no noise.



(a) K=2



(b) K=10

Fig. 6. Number of SLA violations vs. SLA delay guarantee in terms of multiples of minimum propagation delay

Scenario	Loss-Num			SLA-V, $\times 3$		
	Avg	Min	Max	Avg	Min	Max
CONTRACT 5%	7.2	0	77	0.123	0.099	0.394
CONTRACT 10%	7.4	0	77	0.111	0.081	0.304
CONTRACT 50%	10.4	0	89	0.112	0.084	0.379
CONTRACT 100%	11.1	0	95	0.108	0.075	0.304
No Coordination	15.4	0	176	0.151	0.099	0.558

TABLE III
NUMBER OF FLOWS WITH PACKET LOSS FOR VARYING NOISE LEVEL IN THE TRAFFIC MATRIX

Table III shows the results. Since No Coordination does not optimize link costs for network failures, it is not affected by different noise levels. Also because of limited space, we cannot present the full graphs of the SLA-V results, so we only show the number when the threshold is $3\times$ minimum propagation delay. With a higher noise level in the traffic matrix, the optimization in CONTRACT becomes less effective, and the performance as measured by Loss-Num is worse. But even with the highest level of noise (100% standard deviation), the performance of CONTRACT is still better than No Coordination. Examining the results in terms of SLA-V, it is interesting to see that, although CONTRACT is always better than No Coordination, there is no obvious correlation between SLA-V and the noise level. This is because the controller in our experiments optimizes for the Loss-Num metric, and Loss-Num is not necessarily correlated with SLA-V. A higher Loss-Num could correspond to a lower SLA-V because SLA-V is computed solely based on the delays experienced by those packets that are not lost.

Varying Optimization Time Budget:

In this set of experiments, we vary the optimization time budget for CONTRACT. We use 1, 2, 3, 4, 5, 6, 7, 8 and 16 seconds as the budget values and also present a case where no optimization is performed.

The optimization time budget variation presents a trade-off. If the value is too small, the algorithm cannot generate a good configuration. If the value is too large, then the network stays longer in an unoptimized state thus also leading to bad performance. As the results in Table IV show, a time budget of 5 or 6 seconds leads to good performance in both Loss-Num and SLA-V.

Scenario	Loss-Num			SLA-V, ×3		
	Avg	Min	Max	Avg	Min	Max
CONTRACT no optimization	8.3	0	80	0.119	0.093	0.299
CONTRACT 1 second	8.3	0	80	0.125	0.096	0.346
CONTRACT 2 seconds	7.2	0	77	0.123	0.099	0.394
CONTRACT 3 seconds	7.2	0	73	0.124	0.096	0.394
CONTRACT 4 seconds	7.1	0	72	0.113	0.081	0.304
CONTRACT 5 seconds	7.2	0	71	0.108	0.081	0.281
CONTRACT 6 seconds	7.1	0	72	0.109	0.081	0.290
CONTRACT 7 seconds	7.1	0	72	0.116	0.081	0.296
CONTRACT 8 seconds	7.1	0	69	0.133	0.096	0.331
CONTRACT 16 seconds	7.1	0	69	0.143	0.101	0.328
No Coordination	15.4	0	176	0.151	0.099	0.558

TABLE IV
NUMBER OF FLOWS WITH PACKET LOSS FOR VARYING OPTIMIZATION TIME BUDGET

Varying LS Routing Hold Down Timer:

In this set of experiments, we vary the hold down timer as 0, 0.25, 0.5, 1 and 2 seconds.

Scenario	Loss-Num			SLA-V, ×3		
	Avg	Min	Max	Avg	Min	Max
CONTRACT 0 second	3.5	0	42	0.123	0.096	0.316
No Coordination 0 second	8.6	0	156	0.155	0.096	0.549
CONTRACT 0.25 second	3.8	0	44	0.118	0.096	0.313
No Coordination 0.25 second	10.2	0	162	0.138	0.096	0.591
CONTRACT 0.5 second	5.4	0	59	0.121	0.096	0.394
No Coordination 0.5 second	12.9	0	169	0.146	0.096	0.546
CONTRACT 1 second	7.2	0	77	0.123	0.099	0.394
No Coordination 1 second	15.4	0	176	0.151	0.099	0.558
CONTRACT 2 seconds	8.5	0	82	0.123	0.099	0.316
No Coordination 2 seconds	18.9	0	176	0.155	0.096	0.549

TABLE V
NUMBER OF FLOWS WITH PACKET LOSS FOR VARYING HOLD DOWN TIMER

Table V shows the results. With a smaller LS routing hold down timer, the convergence periods for both OSPF routing, and for CONTRACT to finish approving a state, are shorter so there will be less packet loss in the network, but the routers are more stressed in computing routing tables. With larger LS routing hold down timer, the situation is the opposite. CONTRACT is better than No Coordination in all cases.

D. Overhead Evaluation

Larger Network Topology Size:

In this set of experiments, we evaluate the overhead of CONTRACT by using larger topologies of 161 and 315 nodes. Because these simulations require more time, we only explore a subset of the possible failure cases. We use the default parameters, except for the optimization budget. We use an unlimited optimization budget to see how long it takes to do a full optimization.

In all of these failure cases CONTRACT is better than No Coordination under our two metrics. For the 161 node topology, on average CONTRACT spends 40 milliseconds in evaluating one consistent network link state. The convergence time for CONTRACT to approve one state is on average 1.25 seconds. It is composed of the 1 second LS hold down timer, the OSPF convergence time, the maximum round trip delay between the control station and the farthest node, and the 40 milliseconds.

During one experiment the CONTRACT Java code uses on average 320MB of memory. If we let the optimization algorithm run for an unlimited time, it finishes in 19.8 seconds, but with a budget of 8 seconds it comes up with a reasonably good solution (the average number of flows with packet loss is 4 for a 4 second budget, 2 for 8 seconds, and 2 for an unlimited time budget).

For the 315 node topology, on average CONTRACT spent 173 milliseconds in evaluating one network link state. The convergence for CONTRACT is on average 1.46 seconds. The CONTRACT Java code uses on average 620MB of memory. If we let the optimization algorithm run for an unlimited time, it finishes in 68.4 seconds, but with a budget of 32 seconds, it comes up with a reasonably good solution (the average number of flows with packet loss is 12 for a 16 second budget, 8 for 32 seconds, and 7 for an unlimited budget).

In addition, for the 79 node topology, on average CONTRACT uses 180MB of memory. Therefore the memory consumption of CONTRACT approximately grows linearly with the size of the network (number of nodes and edges).

Higher Network Change Frequency:

We stress CONTRACT by increasing the frequency of changes in the network. We toggle the status of one link between up and down 10 times and choose different frequency values for these toggles.

Toggles frequency (toggles/second)	Number of approvals	Loss-Num		SLA-V, ×3	
		CON	No	CON	No
50	1	11	7	0.146	0.131
10	1	8	5	0.110	0.107
6.6	4	4	4	0.113	0.116
5	10	2	4	0.113	0.116
2.5	10	2	4	0.113	0.119

TABLE VI
PERFORMANCE FOR VARYING CHANGE FREQUENCY

Table VI shows the results ("CON" means CONTRACT, "No" means No Coordination in this table). In the first two extreme cases where the network link state changes very quickly, CONTRACT cannot catch up with all the transient network link states, so there is only one approval at the end of the sequence of toggles. In these two cases the performance of CONTRACT is worse than No Coordination. The reason is as follows. We run simulations with a 1Mbps link bandwidth which is relatively low. When routers are sending approval requests to the central controller frequently in these two extreme cases, the approval requests consume a major fraction of the network bandwidth, thus congesting the network and causing extra packet losses. When we run simulations with a more realistic 10Mbps link bandwidth, the approval requests no longer congest any network link. When the frequency is lower, CONTRACT can approve all the transient states, and the performance is better than No Coordination.

Discussion:

CONTRACT introduces a modest amount of additional overhead on routers which includes running the all-pair shortest path routing algorithm, computing fingerprints, sending approval requests, processing approvals, locally adjusting filter configurations, and processing new filter configurations. The computation required by CONTRACT will be performed by

commercial routers in the control plane. Therefore, this will not cause extra delay in the packet forwarding performed by the separated data plane. Commercial routers also commonly support priority queuing. This functionality can be used by CONTRACT when traffic needs to be placed in a low priority queue.

The link weight optimization algorithm we use is admittedly simple. More sophisticated algorithms can be applied that perform better optimization or improve the computation time. However, the delay introduced by our simple link weight optimization algorithm is not critical in the coordination protocol. The computation is performed in parallel with the process to evaluate network link state, and the network continues to function even if the current state is not approved.

Transient failures could cause temporary link weight and routing changes. Any solution that deals with failure faces an inherent trade-off. If a transient failure is reacted upon then computation might be unnecessarily performed. On the other hand, prompt action is required to limit the effect of any failure. CONTRACT also faces the same trade-off. However, in CONTRACT, the effects caused by a transient failure can be reduced by putting the temporarily rerouted traffic into a low priority queue.

V. Conclusion

We have proposed the CONTRACT framework to incorporate coordination into the IP network control plane. CONTRACT can efficiently and programmatically enforce coordination objectives among distributed IGP, traffic load balancing, and traffic policing functions. Furthermore, CONTRACT provides substantial improvements to network performance and SLA compliance during network failures, with reasonable overhead. While we acknowledge the debate on whether more complexity should be added into the network core, we believe that as more and more critical tasks are performed over the Internet, ensuring predictable performance for some applications needs to be considered as a basic service requirement. The CONTRACT framework trades the addition of some complexity into the IP control plane with improving the SLA compliance of the network. As future work, we plan to extend the CONTRACT controller prototype into a fully programmable, modular network control platform.

References

[1] T. Telkamp, "Traffic Characteristics and Network Planning," in *Proc. Internet Statistics and Metrics Analysis Workshop*, 2002.

[2] F. Bernard and T. Mikkel, "Internet Traffic Engineering by Optimizing OSPF Weights," in *INFOCOM*, March 2000.

[3] Arbor Networks Inc., "Worldwide Infrastructure Security Report, Volume IV," http://www.arbornetworks.com/en/docman/worldwide-infrastructure-securit%y-report-volume-iv-2008-/download.html.

[4] R. Callon, *RFC 1195 - Use of OSI IS-IS for routing in TCP/IP and dual environments*, 1990.

[5] J. Moy, *RFC 2178 - OSPF Version 2*, 1997.

[6] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of link failures in an IP backbone," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM New York, NY, USA, 2002, pp. 237–242.

[7] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," in *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 2004.

[8] J. V. der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. McMillan, D. Moloney, J. Mulligan, H. Nguyen, M. Nguyen, A. Ramarajan, S. Saad, M. Satterlee, T. Spencer, D. Toll, and S. Zelingher, "Dynamic connectivity management with an intelligent route service control point," *in* Proceedings of ACM SIGCOMM Workshop on Internet Netwo rk Management (INM), September 2006.

[9] M. Casado, M. Freedman, J. Pettit, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proc. ACM SIGCOMM*, August 2007.

[10] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai, "Tesseract: A 4D network control plane," in *Proc. NSDI*, April 2007.

[11] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the Datacenter," in *Eighth ACM Workshop on Hot Topics in Networks (HotNets-VIII)*, October 2009.

[12] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: Existing techniques compared and new directions," in *Proc. ACM SIGCOMM*, August 2002.

[13] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, "Fast, accurate computation of large-scale IP traffic matrices from link loads," in *Proc. ACM SIGMETRICS*, June 2003.

[14] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 4, pp. 756–767, 2002.

[15] A. Nucci, S. Bhattacharyya, N. Taft, and C. Diot, "IGP link weight assignment for operational Tier-1 backbones," *IEEE/ACM Transactions on Networking (TON)*, vol. 15, no. 4, pp. 789–802, 2007.

[16] Arbor Networks Inc., "Worldwide Infrastructure Security Report, Volume III," http://www.arbornetworks.com/index.php?option=com_docman&task=doc_downl%oad&gid=218.

[17] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg, "Routing design in operational networks: A look from the inside," in *Proc. ACM SIGCOMM*, August 2004.

[18] N. Spring, R. Mahajan, and D. Wetheral, "Measuring ISP topologies with RocketFuel," in *Proc. ACM SIGCOMM*, August 2002.

[19] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM New York, NY, USA, 2003, pp. 313–324.

[20] ——, "Making routing robust to changing traffic demands: Algorithms and evaluation," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. 6, p. 1206, 2006.

[21] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. R
"acke, "Optimal oblivious routing in polynomial time," *Journal of Computer and System Sciences*, vol. 69, no. 3, pp. 383–394, 2004.

[22] D. Applegate, L. Breslau, and E. Cohen, "Coping with network failures: Routing strategies for optimal demand oblivious restoration," in *Proceedings of the joint international conference on Measurement and modeling of computer systems*. ACM New York, NY, USA, 2004, pp. 270–281.

[23] D. Chapman, "Network (In) Security Through IP Packet Filtering," in *Proceedings of the Third UNIX Security Symposium*. September, 1992.

[24] S. Bellovin, "Distributed Firewalls," *Journal of Login*, vol. 24, no. 5, pp. 37–39, 1999.

[25] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM Computer Communication Review*, October 2005.

[26] F. Le, G. Xie, D. Pei, J. Wang, and H. Zhang, "Shedding Light on the Glue Logic of Internet Routing Architecture," *Proc. ACM SIGCOMM*, 2008.

[27] J. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–313, 1965.