Using Simulations to Study the Efficiency of Update Control Protocols

Bernd Pfitzinger[†‡], Tommy Baumann[*], Dragan Macos[+], Thomas Jestädt[†]
[†]Toll Collect GmbH, Linkstr. 4, 10785 Berlin, Germany. Email: {bernd.pfitzinger|thomas.jestaedt}@toll-collect.de
[‡]FOM Hochschule für Oekonomie & Management, Zeltnerstraße 19, 90443 Nürnberg, Germany.
[*]Andato GmbH & Co. KG, Ehrenbergstraße 11, 98693 Ilmenau, Germany. Email: tommy.baumann@andato.com
[+]Beuth Hochschule für Technik, Luxemburger Str. 10, 13353 Berlin, Germany. dmacos@beuth-hochschule.de

Abstract

Taking an existing large-scale realistic discrete event simulation model [1, 2] of a mobile distributed system with more than 700 000 units we investigate the efficiency of control protocols for the software and data updates of on-board-units (OBUs). Adding four different closed-loop control protocols to the existing simulation model we explain and investigate the technical implications of the control protocols: OBUs that allow incoming network connections can be reached at any time (Always-On or Sometimes-On). If the OBU is designed without a TCP/IP server, signaling is implemented either as polling (Sometimes-On with polling) or using a dedicated GSM signaling channel (On-Demand-On). The efficiency is discussed as a combination of metrics gathered during simulation runs and compared with the OBU-controlled update logic. These simulation results are subsequently used as input for the system development process.

1. Introduction

Deploying software and data updates is a seemingly simple yet vital process in any software intensive system. Depending on the circumstances, its success can be measured in terms of failed updates, the time needed for the deployment and the costs associated with the rollout of the updates. Taking the example of the German electronic toll system for heavy goods vehicles (HGVs) we investigate the software and update process in the case of a highly distributed system containing more than 700 000 OBUs. The update process needs to reach each OBU in time to ensure the valid software, geo or tariff data is available on each given OBU. Typically updates are distributed in advance and once the date of validity is reached the systems' capacity needs to allow all remaining (and active) OBUs to download the update immediately. The current system architecture implements an update algorithm distributed across the vehicle fleet (i.e. each OBU makes its own decisions) with some global timing parameters to shape the overall update rate.

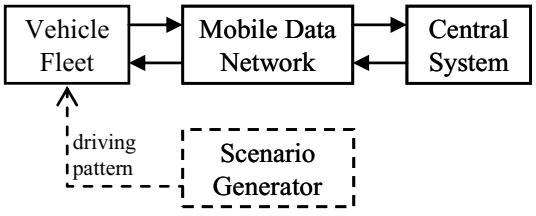

Figure 1: Block diagram of the simulation model and the (external) scenario generator

The difficulty in predicting the properties of a fleet-wide update is the complexity of the whole system. Non-deterministic, non-linear effects as well as emergent behavior [3] possibly manifest only at the level of the whole system: "Emergent behavior is that which cannot be predicted through analysis at any level simpler than that of the system as a whole" [4]. Taking the existing simulation model of the Toll Collect system [1, 2] we use simulations at a scale of 1:1 to determine the behavior and the efficiency of fleet-wide updates for different update control protocols.

The outline of the article is as follows: Section 2 provides a brief overview of the simulation model, the changes to the one described in [2] and it introduces two independently developed models of the driving patterns used in the simulation runs. Section 3 explains the major addition to the simulation model: a closed-loop control of the update process that can be configured in four distinct modes of operations. The comparison of simulation results for the different control protocols is discussed in section 4 followed by a brief outlook and summary in section 5.

2. A simulation model of the Toll Collect system

More than 90% of the toll fees (of a total of 4.36 bn € in 2012, table 4 in [5]) due on federal motorways in Germany are collected automatically by the Toll Collect system. In total, more than 700 000 OBUs are

978-1-4799-2504-9/14 $31.00 © 2014 IEEE
DOI 10.1109/HICSS.2014.634

deployed using up-to-date geo and tariff data to determine the toll charge and to transmit the tolls to the data centre. As a starting point we take an existing discrete event simulation model of the automatic tolling system [1, 2]. This model includes the scenario generator, all subsystems (fig. 1) and processes necessary to determine and to collect the toll charges as well as to deploy updates of the software, geo and tariff data across the vehicle fleet. In effect the simulation encompasses all the relevant HGV driving behavior and the technical processes down to time scales of one second (and some processes with time scales of 50ms, e.g. network authentication handling).

The vehicle fleet is currently set up as a loosely coupled distributed system – each OBU is running the same logic (software code and global configuration parameters) independently for most of the time with infrequent OBU-initiated data transmissions between an OBU and the data centre at random points in time to transmit the toll charges collected or to download updates to the OBU. Especially the mobile data network and the central systems running in the data center exhibit a non-linear behavior, e.g. when operated close to the maximum design specification. The driving patterns are non-deterministic in nature since they encode the driving behavior of the HGVs. To limit the impact of the non-linear and non-deterministic behavior included in the tolling system, several network components are designed to actively limit the maximum load forwarded to the central systems. As a consequence, it is not obvious that a small-scale simulation model will produce the actual system behavior. With several performance optimizations (profiling of atomic blocks, removal of non-affecting retry protocols, dynamic driving pattern calculation [2, 6, 7]) the simulation model is able to run at a scale of 1:1 and simulate a time period of one year within a day on a standard single CPU core using the MSArchitect [9] discrete event simulation core.

A number of different approaches are used to ensure the validity of the simulation model:

- The system architecture is known (white-box approach) and can be transferred to the simulation model (up to including source code from the real-world system in the simulation model).
- The whole simulation model has undergone source code inspection [10] with the relevant subject matter experts.
- The software, geo and tariff data updates occurring in the real-world system over the past year have been reproduced in simulation runs as well as updates of test fleets (which in turn yield insights on how to implement considerably faster rollouts).

For the purpose of this work we extended the simulation runs to a fleet size of 800 000 OBUs (projected future fleet size, compared to 140 000 in [1, 2]) and a simulated time period of 30 weeks (compared to 16 weeks). Each simulation run includes two software updates and three geo and tariff data updates in between, a typical workload occurring over 6 months.

The driving patterns (power cycles and the points in time where tolls are collected) used in the simulation are built on statistical data observed in the actual vehicle fleet and calibrated to the average annual distance driven on German toll roads [11,12]. To get a sense of the uncertainty inherent in the driving patterns we created two different models (by two independent teams) to simulate the driving patterns on a time scale of one second (table 1).

Table 1: Statistical (annualized) averages of the HGV driving patterns generated by two different scenario generators A, B.

	A	B
speed [km/h]	80,2	77,8
toll distance [km]	30 255	31 300
power cycles	**721**	**1 326**
duration [min]	152	62
network losses per power cycle	3,1	2,4
duration of network loss [s]	16	90
power-on time	**21,0%**	**15,6%**
percentage spent on toll roads	20,5%	28,7%
percentage without network	0,6%	4,3%

The driving patterns created by the scenario generators A and B make different assumptions when breaking down the statistical observables to the second-by-second truck behavior. While the annual average toll distance is (calibrated to be) comparable, the number and duration of power cycles differs considerably. As a result, HGVs are powered-on for almost a third longer with scenario generator A (table 1) – time available for deploying updates. Technical limitations and data protection rules make it impossible to observe the power cycle behavior across the real-world fleet. Scenario generator B includes additional data gathered from the Toll Collect test fleet leading to many short power cycles.

Since the publication of [1,2] the simulation model has been enhanced to include network outages occurring at any time – introducing a considerably higher load on the error recovery protocols already present in the simulation model. Obviously, mobile GSM data connections can fail at any time – especially since the HGVs are moving. Again, the exact nature of the network losses (probability, duration) cannot be observed in the real-world fleet. Taking data from the Toll Collect test fleet and a report on the German

mobile networks [13] we conclude that for a given OBU the mobile data network is not accessible for at least up to 5-10% of the total power-on time. As a result, both driving patterns include, on average, several network losses per power cycle (table 1).

Starting with the observed weekly activity patterns (over a 15 week period in 2011, [2]) we extrapolate to 30 weeks (table 2). Using this distribution we determine the active weeks for each OBU, i.e. a given OBU has the same activity pattern for the whole simulation (e.g. an OBU that is rarely active will not switch its activity pattern to very active during the simulation run).

Both scenario generators differ slightly in the weekly activity patterns (especially for HGVs that are rarely seen within the German mobile data networks in the 15 week data set), e.g. scenario generator A has 6% of the vehicle fleet active at least once in 1 to 5 weeks (randomly spread over the 30 week simulation run) compared to 11% for scenario generator B.

Modeling the duration of the power cycles we take additional data from the Toll Collect test fleet in scenario generator B [2] and an older analysis of the real-world fleet in the driving patterns A (table 3). The main difference is the addition of many very short power cycles in model B. This in turn should interrupt updates more frequently and trigger the error recovery protocols more often than with the driving patterns of model A.

To summarize, the existing simulation model has been extended to include network outages and the performance allows simulations at a scale of 1:1. Where possible, the simulation model has undergone source code inspection by subject matter experts and the simulated update behavior was validated against the updates occurring in the real-world system.

The major limitation remaining in the scenario generators is the lack of geographical information, i.e. the tolling process is limited to points in time but not points on a map. For this article the approach taken is sufficient as long as the update process is independent of the OBUs' position (which is the case for HGVs driving within Germany). Location-sensitive algorithms (e.g. special update behavior next to the boundaries of the toll area or caching algorithms [14, 15] or different roaming strategies) will require the addition of location data to the scenario generators.

To mitigate the uncertainty inherent in modeling the driving patterns we implemented two models independently. Taking this foundation we apply the simulation model to investigate different update control protocols.

Table 2: Number of active weeks for a 30 week period used by the scenario generators

	A	B
0	10%	6%
1 - 5	6%	11%
6 - 11	7%	8%
12 - 17	9%	8%
18 - 23	12%	11%
24 - 30	56%	56%

Table 3: Duration of the power cycles used in the driving patterns models

[h]	A	B
0,5	6,0%	64,3%
1,0	8,9%	12,6%
2,0	24,6%	11,6%
3,0	22,5%	6,4%
4,0	16,2%	2,3%
5,0	21,8%	1,1%
> 5,0	0,00%	1,7%

3. Software and data update control protocols

The purpose of the automatic toll system is of course to collect the toll charges due. However, updates are also occurring regularly either due to new features (or optimization) in the OBU software or to changes in the geo and tariff data. Updates need to be deployed fleet-wide within a given time period (typically several weeks), at acceptable cost and without impacting the tolling process. The current mode of operations allows for up to four software updates per year and many more updates to the geo and tariff data.

Today the rollout of updates is an emergent property of the tolling system: While powered-on, each OBU decides independently and randomly (according to global timing parameters) if and when to connect to the central system to check for updates. Changing the global timing parameters allows different rollout rates – that can only be determined through simulations. To ensure the operational validity of the simulation model [16,19] we simulate the update behavior observed between March 2012 and Jan 2013 and compare the fleet-wide percentage of updates deployed at the end of each day. Computing the Pearson product-moment correlation coefficient [17] for the deployment of software, geo and tariff data updates we find the simulation results to be very close to the actual system behavior (as given by a correlation factor close to 1,

table 4, using the driving patterns of scenario generator B).

Table 4: Pearson correlation between simulated and observed updates

	correlation
software	0.99898
geo data	0.99570
tariff data	0.99474

As part of the Toll Collect system development process we use the simulation model to investigate the system behavior of planned changes. For the purpose of this article we investigated a proposed redesign of the update process: Instead of an algorithm running locally (and in large parts independently) on all OBUs, we introduce a closed-loop controller running in the data center. This change to the system architecture has two consequences:

- The update logic contained in the OBU is removed and
- the central closed-loop controller needs the ability to connect to a given OBU to deploy the update.

From an operational point of view another consequence is that instead of the update logic the OBU would need to implement a "self-test" algorithm to check whether the software and data is up-to-date.

Today it is possible to design a distributed system assuming a permanent mobile data connection to the data center – as seen in the widespread adoption of smartphones. In our investigation we use this behavior as one extreme ("Always-On", at least when the HGV is powered-on) and the other extreme is a dedicated signaling channel used to start an update on a given OBU ("On-Demand-On"). Since updates occur infrequently we include an intermediate behavior where the OBU is only online for certain time periods ("Sometimes-On"), e.g. after the power-on and at regular (or random) intervals thereafter.

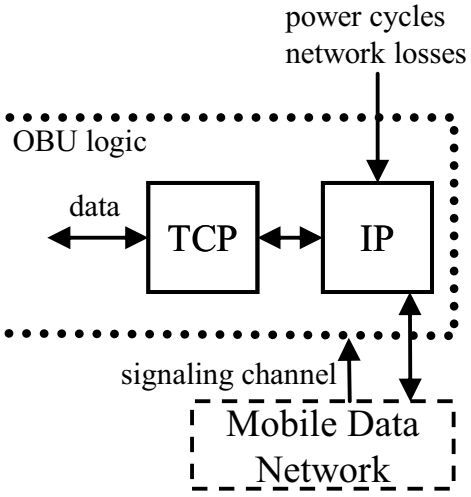

Figure 2: IP and TCP connection machines added to the simulation model (optional signaling channel for On-Demand-On).

For the transmission of updates we use a TCP/IP-based communication (apart from the extra signaling channel in the On-Demand-On scenario, fig. 2). Of course, the closed-loop controller running in the data center needs to be able to connect to the OBU. If we decide to simplify the OBU so that it does not run its own TCP/IP server, incoming connections to the OBU need to be emulated through polling. The following subsections explain the four controllers implemented in the simulation model.

3.1. Always-On and Sometimes-On control protocols

For the Always-On scenario the IP connection machine added to the OBU automatically ensures the availability of a fully configured IP-stack. I.e. as long as the OBU is powered on, it will automatically connect to the GSM network and establish a packet data protocol (PDP) connection. In our case the IP addresses are allocated in a private network by our data center once the OBU is successfully authenticated.

Whenever the network access is lost, the OBU will continuously try to reestablish the PDP context. Depending on the configuration of the mobile data network, the previous PDP context may be reused or a new PDP context needs to be created by the mobile network operator (MNO) putting an additional strain on the MNO and the central services.

The Sometimes-On scenario modifies the IP connection machine so that the OBU releases the PDP context for some time periods. The central controller must use the available time for updates. In our simulation the OBU will remain reachable after each power-on event and wake up periodically afterwards (once every four hours).

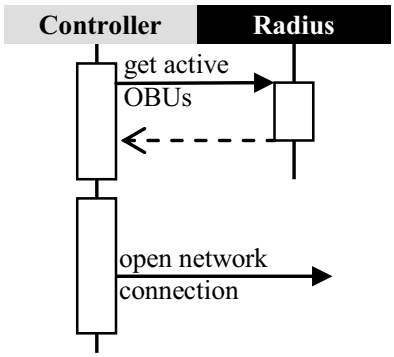

Figure 3: Always-On and Sometimes-On controller synchronization

In both scenarios TCP/IP servers are running in the data center and on the OBU, allowing the establishment of new connections from each side. The closed-loop controller in the data center is therefore able to initiate network connections to any OBU that needs to receive an update.

For the purpose of the simulation we implemented a simple algorithm for the controller (fig. 3): We periodically take the table of currently active OBUs (given as the IP-addresses allocated by the Radius-server in the data center) and loop over the active OBUs as long as the controller needs to deploy updates and still has spare capacity. Of course, the time lag between the OBU connecting to the data center and the controller trying to establish a connection will lead to some OBUs being no longer available (e.g. due to power cycles or loss of the network connection). The controller then needs to wait for a timeout period to conclude that the OBU is no longer available before it can proceed to the next OBU in line. In effect this reduces the update speed since part of the controller capacity is tied up waiting for timeouts. Initially we synchronized the table of active OBUs every 60 seconds (the shortest power cycles included in the driving patterns) but noticed a sizeable amount of unreachable OBUs as consequence. Therefore the synchronization frequency was doubled to once every 30 seconds – incurring a substantial performance overhead. We repeat the simulations with two different controller capacities (up to 1 000 or 5 000 parallel connections) to gauge its effect on the update speed.

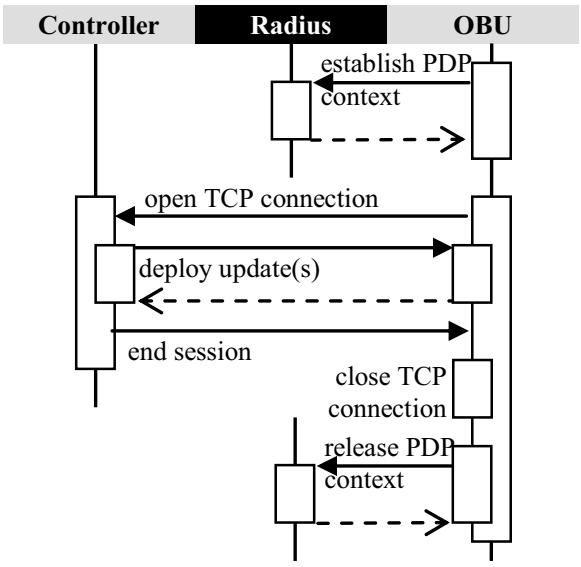

Figure 4: Polling mechanism used for OBUs without TCP/IP server

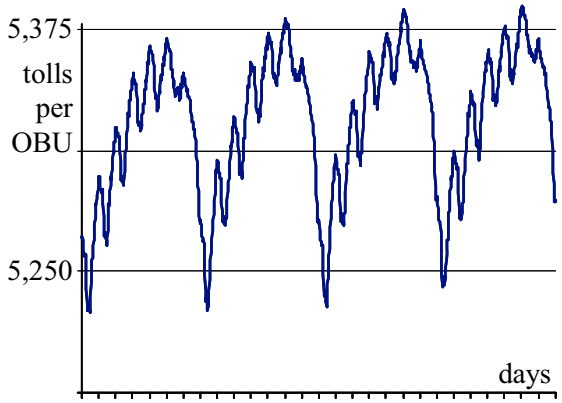

Figure 5: Average of tolls stored on OBUs over four weeks with a software update at the end of the second week (Always-On)

3.2. Sometimes-On using polling operations

The communication protocol needs to be modified if the OBU does not run its own TCP/IP server, i.e. does not allow incoming connections. The inability of the controller to open network connections to the OBU will of course introduce delays to the update process. Since the controller logic resides in the data center the OBU establishes "polling" connections (fig. 4): After establishing a PDP context the OBU opens a TCP connection directly to the controller and waits. The controller can use the TCP connection to deploy one or several updates, maintain the idle connection for some time ("long polling") or send a command to terminate the connection. Once the connection is closed the Sometimes-On logic will open a new connection after a given time period or at the start of the next power cycle. If the polling rate is higher than the publish rate, the polling mechanism will create many unnecessary connections – unfortunately the typical scenario in our case: We aim for fleet-wide updates within a few days (i.e. deploying updates at the maximum rate supported by the controller) occurring only several times per year.

Compared to the Sometimes-On scenario without polling the major difference is that the polling protocol immediately opens a network connection to the controller via TCP/IP. There is no intermediate step where the controller periodically retrieves newly registered OBUs from the Radius server. As a consequence the polling protocol should make better use of the available parallel connections – the time lag between registering the IP address and the controller trying to open a connection is eliminated. However, the controller is still checking once every 30 seconds whether any updates need to be deployed.

3.3. On-Demand-On: A control protocol using a dedicated signaling channel

A dedicated signaling channel should allow much more efficient control protocols since the OBU only establishes a PDP context when the controller needs to connect to the OBU. In our case we choose the network initiated USSD (Unstructured Supplementary Service Data) protocol present in the GSM standard [18]. Similar to the short message service (SMS) USSD allows sending messages to a mobile subscriber (MS). However, in the case of USSD messages are not stored and forwarded later when the MS cannot be reached. In effect, sending USSD messages the controller can wake up OBUs that are powered-on and within the reach of the mobile data network (but have not yet activated the data network). After waking up the OBU establishes a PDP context and opens a TCP connection to the controller.

Since USSD messages that cannot be delivered are discarded by the network, the controller is not faced with old messages being forwarded e.g. at the start of the week (a case that needs to considered if SMS is chosen as signaling channel). Monitoring the controller capacity, the controller will issue as many USSD request as possible to fill the available 1 000 (or 5 000) parallel connections.

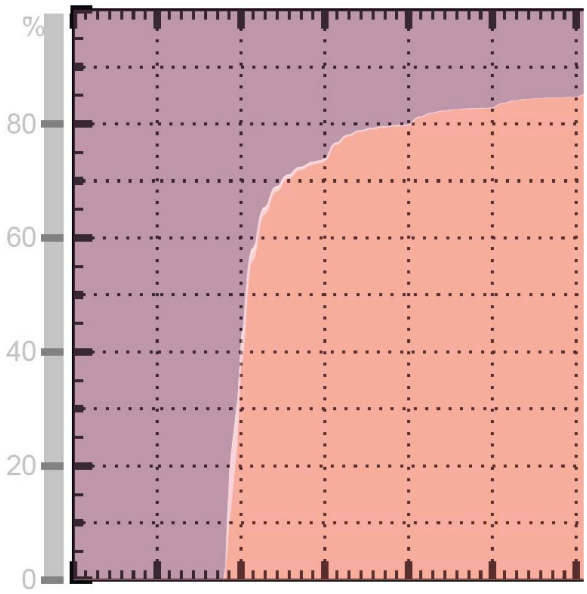

Figure 6: Fleet-wide software update over 6 weeks (Always-On, 1000 parallel connections, driving patterns model B)

4. Determining the efficiency of control protocols

In the Toll Collect example the most important property of the closed-loop controller is its ability to quickly deploy fleet-wide updates without impact on the ongoing tolling process. During each simulation run we monitor the toll charges stored on the OBUs (fig. 5, starting on a Monday and showing four consecutive weeks) to ensure that updates do not affect the tolling process. The daily activity pattern is clearly visible as well as the weekends. As intended the weekly behavior is very similar (in pattern and in absolute terms). The software update starting at the end of the second week does not affect the tolling – it is invisible in fig. 5.

In a typical scenario we deploy a small delta patch of the software with a file size requiring several minutes of download via the mobile data network. Starting the update on a Saturday afternoon (at the end of the second week, fig. 6) it is distributed quickly across the fleet, reaching 50% of the fleet within less than 2 days. In fact, looking at the number of parallel connections used by the controller (fig. 7) we find that the cap at 1 000 connections is only reached for less than 12 hours. I.e. two days after the start of the update the update is limited by the driving patterns used in the simulation – the fact that most OBUs are unreachable for considerable periods of time.

Taking this into account we compare the four different scenarios by looking at the time taken to reach 50% of the OBU fleet (table 5). As expected, the Sometimes-On scenario will give the worst rollout speed (when used without polling). Otherwise the

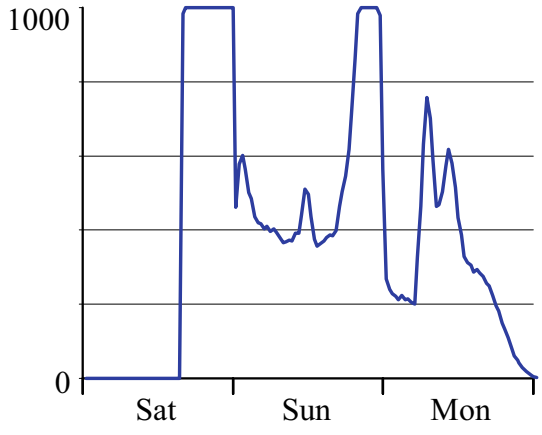

Figure 7: Parallel connections at the controller at the start of a fleet-wide update (Always-On, up to 1 000 parallel connections, driving patterns model B)

different protocols perform almost identically (with slightly longer rollout durations for the driving patterns generated by scenario generator B). Increasing the controller capacity from 1 000 to 5 000 parallel connections levels the field and all four control protocols reach 50% of the fleet within 36 (39) hours for model A (B).

The effect of including a large proportion of short power cycles in the driving patterns model B is visible in many other metrics as well. E.g. the average hourly number of failed connections from the Always-On or Sometimes-On controller to the OBU is up to three times higher when simulated with the driving patterns from scenario generator B rather than A.

A direct comparison between the distributed update algorithm (status quo) and the new control protocols is difficult since the two solutions implement opposite operational philosophies: The new closed-loop controller is intended to deploy updates at maximum speed and capacity whereas the status quo is configured to level updates over considerable periods of time. Taking this into account, the main difference is the duration the OBU is running with an established PDP context. The distributed update algorithm very rarely connects to the wireless data network whereas the new solutions differ widely (table 6): As expected – Always-On is considerably longer online although most often no updates need to be deployed.

Replacing the distributed update algorithm with the controller running in the data center we change the performance behavior of the simulation model: The continuously running algorithm on each OBU is shut down, the performance of simulating the fleet-wide behavior is improved. However, the closed-loop controller in the data center requires additional computing resources – especially since we chose a rather simplistic algorithm with periodic replication of tables and the subsequent check whether any updates need to be deployed. For the sake of fast rollouts the controller checks twice per minute for new updates (where the 30 week simulation run includes only a total of 5 updates).

As a consequence the sum total of CPU commands issued during the simulation run is lowest for the distributed update algorithm and the On-Demand-On scenario (16% more CPU cycles).

Table 5: Rollout duration [h] to reach 50% of the fleet using up to 1 000 connections and two driving patterns A, B

	A	B
Always On	40	42
Sometimes On	61	69
Sometimes-On Polling	40	43
On-Demand-On	42	46

For the Sometimes-On scenario the simulation needs 70% more CPU cycles and for the Always-On scenario 160% more CPU cycles. In the end the new controller logic is responsible for 10% (On-Demand-On) up to 60% (Always-On) of the total simulation runtime.

Table 6: Relative factor of average online time per OBU for the driving patterns A, B (On-Demand-On = 1.0)

	A	B
Always On	921.2	579.4
Sometimes On	11.9	15.4
Sometimes-On Polling	1.3	2.3
On-Demand-On	1.0	1.4

5. Summary and Outlook

We have expanded and validated an existing simulation model of the German electronic toll system for HGVs to allow simulation runs at a scale of 1:1. The simulation performance is sufficient to use the simulations in "what-if" scenarios in the early design stages of the system engineering process including all processes running at time scales down to one second without resorting to poorly understood scaling factors.

Using the simulation model to investigate a proposed change of the overall system architecture, we investigated the fleet-wide update behavior – an emergent property owing to the distributed algorithm implemented in the system. Replacing it with four different implementations of a closed-loop controller running in the data center, we compare the efficiency in terms of the time taken for a fleet-wide update and the average duration the OBUs are online. Especially the online time differs widely between the four scenarios: The use of a dedicated signaling channel ("On-Demand-On") reduces the amount of time spent online by a factor of almost 1 000 (compared to "Always-On"). According to the simulation results it is not necessary for the OBU to be online at all times to achieve similar rollout rates.

Introducing simulations to the system engineering process we were able to quickly improve the specification accuracy. The simulation results give an evaluation of the different proposed implementations and are a new way of specifying the behavior of the dynamic system.

Taking the microscopic simulation model we have identified and mitigated some limitations of the model: Including all technical processes running at time scales down to one second was improved for some networking processes to a level of 50 ms. The validation of the simulation model is currently limited

to the daily update behavior. Additional data with a higher temporal resolution is required to validate the simulation model at shorter time scales where we expect to see the limitations imposed by the statistical data used in the driving patterns.

For future work we propose to use the simulation model and an optimization framework to reconstruct the detailed daily driving patterns from the real world systems' observables instead of trying to determine fleet-wide statistical driving patterns.

10. References

[1] B. Pfitzinger, T. Baumann, and T. Jestädt, "Analysis and evaluation of the German toll system using a holistic executable specification", Hawaii International Conference on System Sciences, pp. 5632-5638, 2012.

[2] B. Pfitzinger, T. Baumann, and T. Jestädt, "Network Resource Usage of the German Toll System: Lessons from a Realistic Simulation Model", Hawaii International Conference on System Sciences, pp. 5115-5122, 2013.

[3] H. Kopetz, "Real-time systems: design principles for distributed embedded applications", Springer, 2011.

[4] G. B. Dyson, "Darwin among the machines: The evolution of global intelligence", Basic Books, 1998.

[5] Bundesministerium der Finanzen, "Monatsbericht des BMF, Februar 2013", ISSN 1618-291X, Berlin, Feb. 2013 [accessed 20-Mar-2013]. Available: http://www.bundesfinanzministerium.de/Content/DE/Monatsberichte/2013/02/Downloads/monatsbericht_2013_02_deutsch.pdf?__blob=publicationFile&v=4.

[6] T. Baumann, B. Pfitzinger, and T. Jestädt, "Simulation driven design of the German toll system – evaluation and enhancement of simulation performance", Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 901-909, 2012.

[7] A. Pacholik, T. Baumann, W. Fengler, and D. Grüner, "Discrete Event Simulation Performance – Benchmarking Simulators", Grand Challenges in Modeling & Simulation, Genua, 2012.

[8] B. Pfitzinger and T. Jestädt, "Exploring HGV fleet behavior: Notes from the German toll system", proceedings of the 9th ITS European Congress, Dublin, 2013 (forthcoming).

[9] MSArchitect, [accessed 17-Mar-2013]. Available: http://www.andato.com/

[10] M. E. Fagan, "Design and Code inspections to reduce errors in program development", IBM Systems Journal 15 (3), pp. 182-211, 1976.

[11] B. Pfitzinger, T. Baumann, and T. Jestädt, „Simulating the German toll system: Choosing 'good enough' driving patterns", proceedings of the mobil.TUM 2013 conference, Munich, 2013 (forthcoming).

[12] Bundesamt für Güterverkehr, "Mautstatistik – Jahrestabellen 2012," [accessed 07-March-2013]. Available: http://www.bag.bund.de/SharedDocs/Downloads/DE/Statistik/Lkw-Maut/Jahrestab_12_11.pdf?__blob=publicationFile

[13] B. Theiss, "Mobilfunk in Deutschland – Der Netztest 2012", connect 12, 2012.

[14] K. Lunde, L. Kieble, and M.-A. Funk, "Prediction strategies in a service level granting prefetching cache for version-controlled gis data," ISAST Transactions on Computers and Intelligent Systems, vol. 2, no. 2, pp. 46-51, 2010.

[15] K. Lunde and L. Kieble, "Simulating communication within a satellitebased automated toll collection system," Proceedings of the 55th International Scientific Colloquium, pp. 318-323, 2010.

[16] R. G. Sargent, "Verification and validation of simulation models", Winter Simulation Conference, pp. 130-143, 2005.

[17] The Apache Software Foundation, "Apache Commons Math, Release 3.1.1", Jan 2013, [accessed 08-March-2013]. Available: http://commons.apache.org/proper/commons-math/download_math.cgi.

[18] ETSI 3rd Generation Partnership Project, „Technical Specification TS 24.090 version 11.0.0", Oct 2012 [accessed 10-March-2013]. Available: http://www.etsi.org/deliver/etsi_ts/124000_124099/124090/11.00.00_60/ts_124090v110000p.pdf.

[19] R. G. Sargent, "Verification and validation of simulation models", Journal of simulation, vol. 7, pp. 12-24, 2013.