24 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

MMBnet 2017

Proceedings of the 9th GI/ITG Workshop

„Leistungs-, Verlässlichkeits- und Zuverlässigkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen“

Udo R. Krieger, Thomas C. Schmidt, Andreas Timm-Giel (eds.)

University of Bamberg Press

24 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Contributions of the Faculty Information Systems and Applied Computer Sciences of the Otto-Friedrich-University Bamberg

Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Contributions of the Faculty Information Systems and Applied Computer Sciences of the Otto-Friedrich-University Bamberg

Band 24

University of Bamberg Press
2017

MMBnet 2017
Proceedings of the 9th GI/ITG Workshop

„Leistungs-, Verlässlichkeits- und Zuverlässigkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen"

Udo R. Krieger, Thomas C. Schmidt, Andreas Timm-Giel (eds.)

University
of Bamberg
Press
2017

Bibliographische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Informationen sind im Internet über http://dnb.d-nb.de/ abrufbar.

Herstellung und Druck: Digital Print Group, Nürnberg
Umschlaggestaltung: University of Bamberg Press, Larissa Günther
Umschlagbild: © Marcel Großmann

ISSN: 1867-7401
ISBN: 978-3-86309-501-7 (Druckausgabe)
eISBN: 978-3-86309-502-4 (Online-Ausgabe)
URN: urn:nbn:de:bvb:473-opus4-497624
DOI: http://dx.doi.org/10.20378/irbo-49762

Contents

Organization

The 9th GI/ITG Workshop MMBnet 2017 „Leistungs-, Verlässlichkeits- und Zuverlässigkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen" was organized by the Institute of Communication Networks at Hamburg University of Technology (TUHH), the Internet Technologies Group at Hamburg Univer-sity of Applied Sciences (HAW Hamburg) and the Faculty of Information Systems and Applied Computer Sciences, Professorship of Computer Science, on behalf of University of Bamberg, Germany.

Organizing Committee

General Chairs

Udo R. Krieger	University of Bamberg, Germany
Thomas C. Schmidt	Hamburg University of Applied Sciences (HAW Hamburg), Germany
Andreas Timm-Giel	Hamburg University of Technology (TUHH), Germany

Technical Program Committee

Torsten Braun	Universität Bern
Peter Buchholz	TU Dortmund
Georg Carle	TU München
Hans Daduna	Universität Hamburg
Hermann De Meer	Universität Passau
Markus Fidler	Leibniz Universität Hannover

Reinhard German	Friedrich-Alexander Universität Erlangen-Nürnberg
Gerhard Haßlinger	Deutsche Telekom AG and TU Darmstadt
Klaus-D. Heidtmann	Universität Hamburg
Tobias Hoßfeld	Universität Duisburg-Essen
Udo R. Krieger	Universität Bamberg
Paul J. Kühn	Universität Stuttgart
Michael Menth	Universität Tübingen
Peter Reichl	Universität Wien
Johannes Riedl	SIEMENS AG
Thomas C. Schmidt	HAW Hamburg
Jens B. Schmitt	TU Kaiserslautern
Markus Siegle	Universität der Bundeswehr München
Otto Spaniol	RWTH Aachen
Dirk Staehle	Hochschule Konstanz – Technik, Wirtschaft und Gestaltung
Helena Szczerbicka	Leibniz Universität Hannover
Andreas Timm-Giel	TU Hamburg-Harburg
Kurt Tutschku	Blekinge Institute of Technology, Karlskrona
Tadeus Uhl	Hochschule Flensburg
Alexander von Bodisco	Hochschule Augsburg
Oliver Waldhorst	Hochschule Karlsruhe - Technik und Wirtschaft
Bernd E.Wolfinger	Universität Hamburg
Martina Zitterbart	KIT Karlsruhe

Additional Reviewer

Gerhard Haßlinger	Deutsche Telekom AG and TU Darmstadt

Preface

Today, the demand for smart, efficient services and the effective transport of their related traffic streams is rapidly increasing on a global scale. It is arising from growing mobility, new industrial services, smart cities, complex private applications, and the dissemination of multimedia information. The latter is caused by underlying massive human interactions and supported by a variety of online social networks. The integration of wireless 5G networks, wired high-speed networks and the evolving Internet-of-Things additionally provides major drivers of the fast growing Internet traffic. The steadily increasing demand to utilize the deployed network infrastructures and server resources most effectively generate some of the greatest challenges for engineering and science in modern information societies.

Mathematical methods of systems and network monitoring, modeling, simulation, and performance, dependability and reliability analysis constitute the foundation of modern quantitative evaluation methods. These techniques represent the basis for an application of the underlying methodologies to a large variety of areas in modern information society. If we consider the mathematical and technical foundation of currently applied modeling, analysis, simulation, and performance evaluation techniques with regard to their future development, a substantial scientific training and a constructive dialog of young scientists and upcoming engineers with experts in the field is needed. It may facilitate a more efficient way to apply these new techniques and reveal the most effective modalities to design software-defined next-generation networks and advanced cloud computing systems of an information society.

To support the sketched personal development goals of young scientists and engineers working in this area, the responsible administrative forum of GI/ITG Technical Committee “Measurement, Modelling and Evaluation of Computing Systems” (MMB) has established the workshop series MMBnet. The MMB com-

munity hopes that related research activities of young members can be stimulated by these means. Covering corresponding research topics of MMB, the 9^{th} GI/ITG Workshop MMBnet 2017 „Leistungs-, Verlässlichkeits- und Zuverlässigkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen" was held at Hamburg University of Technology (TUHH), Germany, on September 14, 2017.

After a careful review process, the Organizing Committee of the workshop has compiled a scientific program that included 4 regular papers and the invited talk "Caching for Content Delivery and Cloud Services: Use Cases, Caching Strategies and Performance Evaluation" by Dr. Gerhard Haßlinger, Deutsche Telekom AG and TU Darmstadt, Germany. We thank the authors for their submitted papers and all speakers, in particular the invited expert, for their lively presentations and the constructive discussions.

As conference chairs we are grateful for the support of all members of the Technical Program Committee and we also thank the external reviewer for his dedicated service and for the timely provision of the reviews.

We express our gratitude to Hamburg University of Technology as conference host of MMBnet 2017 and, in particular, to all members of the local organizers' team regarding their great efforts.

We acknowledge the support of the EasyChair conference system and express our gratitude to its management team.

Last but not least, we appreciate the relentless support by Marcel Großmann, Faculty WIAI at University of Bamberg, and University of Bamberg Press while compiling the proceedings.

Finally, we hope that our reader's future research on monitoring, modeling, analysis, and performance evaluation of next-generation networks and distributed systems will benefit from the proceedings of the 9^{th} GI/ITG Workshop MMBnet 2017.

September 2017

Udo R. Krieger
Thomas C. Schmidt
Andreas Timm-Giel

Invited Talk

Caching for Content Delivery and Cloud Services: Use Cases, Caching Strategies and Performance Evaluation

Gerhard Haßlinger

Technische Universität Darmstadt and
Deutsche Telekom AG
Heinrich-Hertz-Straße 3-7
D-64295 Darmstadt, Germany

Abstract

The topic of Internet content caching regained relevance over the last decade due to the extended use of data center infrastructures in CDNs, clouds and ISP networks to meet capacity and delay demands of multimedia services.
The talk addresses global CDN and cloud platforms from QoS demands to the underlying server and caching architectures as well as statistics on the traffic and user request behavior.
Appropriate caching strategies have to be assessed in a tradeoff between hit rate efficiency and simple cache update processes. Results on the cache hit rate are presented comparing three types of cache (replacement[1]) strategies based on

- LRU: Least Recently Used,
- LFU: Least Frequently Used,
- Optimum caching with knowledge of future requests.

Caching evaluation results are obtained by simulation on the entire range of Zipf distributed user requests, which have been confirmed for many web applications. Analytic results are also relevant for special cases. Optimum caching turns out to provide a non-trivial playground for Markov models. Finally, we show that optimum caching does not only provide a theoretical upper bound on the hit rate, but can be combined for improving usual strategies whenever a limited look-ahead is possible.

[1] Web caches can also select their input in contrast to paging systems and local caches.

Curriculum Vitae

Gerhard Haßlinger has more than 10 years of experience as a researcher and lecturer in computer science at Darmstadt University of Technology, and as an engineer at Deutsche Telekom, where he works on architectures for service and Internet access provisioning via fixed/mobile broadband platforms. His research interests include content distribution, traffic engineering, reliability, and quality of service aspects of computer and communication networks, information theory and coding.

Workshop Papers

Real-Time Communication

Saving Bandwidth by Limiting the Buffer Size in HTTP Adaptive Streaming

Sebastian Surminski, Christian Moldovan, Tobias Hoßfeld

University of Duisburg-Essen
http://www.uni-due.de

Abstract. Video streaming is one of the most bandwidth-intensive applications on the Internet. In HTTP adaptive video streaming the video quality is selected according to the available bandwidth. To compensate bandwidth fluctuations, players use a buffer in order to ensure a smooth video output. On one hand, if the buffer runs empty, the video playback stops, which users experience as negative. On the other hand, if the user aborts video playback, the video in the buffer was unnecessarily transmitted, hence this bandwidth was wasted. In this paper, we present a study in which we investigate the behavior of two video players and different buffer configurations in real-world bandwidth scenarios. Thereby, we focus on the dimensioning of the buffer size and the trade-off between wasted bandwidth and the playback quality.

Key words: Buffer, Adaptive Video Streaming, Wasted Bandwidth

1 Introduction

Video streaming is a popular service on the Internet. It accounts for about 70% of all downstream traffic on fixed access networks in North America during peak evening hours [1]. As a consequence, it is worth investigation and identification of saving potentials by limiting unnecessary transmissions. Nowadays, mainly HTTP adaptive streaming (HAS) is used. This means, that the video is played while being downloaded, and the video quality is selected according to the available bandwidth. As this process is sensitive regarding bandwidth degradations and especially outages, the player maintains a buffer of video that is yet to be played, to cope with bandwidth fluctuation. This happens mainly while being on the move. If the buffer runs empty, because the video is played faster than downloaded, the

playback stops. From user studies, it is known that such stalling events have a negative impact on the Quality of Experience (QoE) [2].

On the downside, video playback is often aborted by the user before the video has been completely played. This means that the remaining content of the buffer is never played, hence transmitted without need, which means this bandwidth is wasted. A study on YouTube in 2011 showed that about 60% of all requested videos were watched for less than 20% of their duration. In numbers, about 40% of all viewed videos on YouTube were aborted within the first 30 s, 20% of the videos were played less than 10 s [3].

Considering these facts, it is interesting to investigate the influence of the buffer size on the playback behavior of HAS video players, under realistic bandwidth conditions in mobile scenarios. To do so, we conducted a measurement study in which we evaluated the playback behavior of two different HAS video players in real-world scenarios. Subject to this study are two different video players, namely the YouTube player[1] and the Shaka player[2]. The Shaka player allows us to configure the buffer configuration, this way we can compare different buffer configurations. The bandwidth conditions, that the players face, are mobile scenarios, which were recorded in everyday commuting situations, using different means of transport.

This methodology allows investigating the behavior of video players, especially the adaptation logic. Additionally, we have developed a method to evaluate the buffer level of the video player. By this, we can determine the amount of video that is wasted when playback of the video is aborted.

The structure of this paper is as follows. First, we introduce the issue of wasted bandwidth and its implications, as well as previous work on this topic in Section 2. After that, we sketch the way HAS works and the influence of the buffer size. In Section 3, we describe the methodology of the experiments we conducted. In Section 4, the results are presented. The paper ends with the conclusions in Section 5.

[1] `https://www.youtube.com`
[2] `https://github.com/google/shaka-player`

2 Background and Related Work

First, we identify the causes for wasted bandwidth. Then we introduce basic concepts of HAS. Different aspects of the buffer size in HAS were investigated in previous work, which we will present in the final part of this section.

2.1 Wasted Bandwidth

For video streaming services, wasted bandwidth at large scale induces significant costs. Not only server capacities, but also link capacity is unnecessarily used. The implied cost is not only relevant for the video provider, but also for the user, for example when the user is on a data plan. There, often a fixed, maximum data contingent is determined. Additional traffic costs extra charge. So the user is interested in saving bandwidth, in the best case without reducing the video quality [4]. Additionally, transmitting data consumes a lot of energy on mobile devices. Thus, transmitting less data may increase battery life time [5].

Furthermore, wireless networks have a limited overall capacity per cell. So it is interesting to limit the traffic transmitted to allow to serve more users per cell, and improve the overall performance of all users.

2.2 Adaptive Video Streaming

Adaptive video streaming allows to adapt the video quality to the available bandwidth using different quality levels. To do so, the video is encoded in different qualities, the so-called representation levels, which are then split into small segments, typically with a length of several seconds. The video player selects autonomously which segments to play. Optimally, the video player selects the maximum quality without exceeding the available bandwidth. On one hand, if the player is too conservative, the video quality is not as good as it could be. On the other hand, if the bandwidth is exceeded, the video is played faster than it is possible to fetch new video data. However, the player has to select the right quality level depending on the current available bandwidth.

2.3 Buffer Size

To compensate for bandwidth fluctuations, the player maintains a playback buffer. In this work, we measure the buffer level by the length of the video it contains,

independently of the actual file size. If the buffer runs empty, the playback stops. This is called a stalling event. Users experience stalling events as very negative, so they have to be prevented [6]. The playback is resumed if the buffer is sufficiently filled. In the following, this limit is called the rebuffering goal. The maximum fill level of the buffer is called the buffer size.

When speaking about the buffer size, we simplify it by referring to the length of the video in the buffer, not the file size of the video. This is the same way it is done by video players like the Shaka player, which we use in this study, and also in research investigating this topic, like [16]. From the perspective of wasted bandwidth, it would make more sense to determine the buffer level depending the file size. Using the average video data rate, which can be found in Section3.5, an estimation of the wasted bandwidth can be done from the buffer level.

2.4 Related Work

Research on HAS often focuses on the adaptation strategies, which select the representation layer, e.g. the segments that will be played. Especially in mobile scenarios, the available bandwidth is constantly changing. As this development is difficult to foresee, the adaptation logic is limited, although there are good approaches, like [7]. To solve this problem, location-aware approaches have been introduced. This allows to predict the bandwidth in the future and to adapt the video quality accordingly. On one hand, this approach is appealing because it allows better adaptation of the video quality, and in practice, mobile devices already come with technology to determine their location. On the other hand, this introduces technical overhead and serious privacy concerns [8], [9].

Nevertheless, the issue of wasted bandwidth has already been addressed by research from different points of view. Li et al. addressed the aspect of energy consumption and introduced a cache management method that reduces the power consumption of the radio modules in smart phones [10], [11]. Wu et al. presented an adaptive method to reduce the amount of wasted bandwidth by making the buffer size of HAS players adaptive [12]. This approach allows to weigh the costs for transmitting data and its energy consumption and react accordingly.

In this paper, we conduct experiments with HAS video players. To do so, we use the players' API to monitor the behavior and the internal status, including the

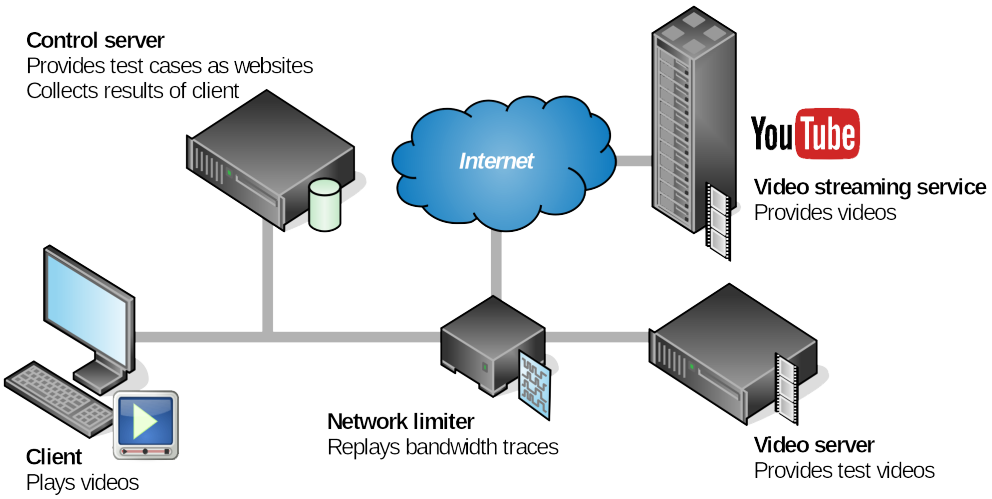


Fig. 1. Overview of the setup of our experiment. The client is directly connected to the control server, while the video is transmitted via the network limiter.

buffer level. Another approach for monitoring video players is made in [13], where the video playback is monitored using an Android application. Both methods need cooperation with the users or the video server. A completely passive monitoring technique is presented in [14] with the tool YOUQMON. Instead of monitoring the video player directly, the data stream is captured. With this information the supposed behavior of the player is simulated, which allows to predict when stalling events occur.

3 Methodology

We developed a setup that allows us to test HTML video players under predefined bandwidth conditions, while monitoring their behavior. In the experiments, we used two different video players for HAS. Our complete setup is shown in Figure 1. In the following, we describe the elements of our setup.

3.1 Video Players

For the experiments, we used two different video players. The YouTube player is used by the most popular video streaming web site YouTube. So it represents the state of the art for HAS, that is used in practice. We compare it to the Shaka player, an open source video player, that allows us to configure the buffer configuration. This way, we can change the buffer configuration and investigate its effects on the playback behavior.

Both video players offer a JavaScript API to control and give status information. Our approach is to use these interfaces to monitor the playback behavior to get statistics about the video playback. As these numbers are directly taken from the video player, they are very accurate. While the YouTube player has to be treated as a black box, the open source player can be configured, so it is possible to change its buffer configuration, in particular, the buffer size and the rebuffering goal. The buffer size is the maximum length of the video, that can be stored in the buffer. The rebuffering goal defines the minimum buffer level, which needs to be reached, before the player starts playing the video. This is relevant, as a lower buffer level can reduce the stalling time, but increase the stalling frequency, which both have an influence on the QoE.

Using the players' API, we collected the following information, which then was used to determine the quality of the playback of the video:

- Initial waiting time
- Number and duration of stalling events
- Resolution of the video and adaptation events
- Buffer level

Additionally, we also documented the bandwidth of the video stream and the bandwidth estimation of the player.

3.2 Client

The client can be an arbitrary device running a recent web browser, that is able to run the video player. In our experiments this is a *Firefox 50* browser on a machine running *Ubuntu 16.04*. It could also be changed to any other device, for example a smart phone, to investigate whether the players behave differently on other device types.

3.3 Control Server

The experiment is controlled by the 'Control server', which provides the web page for the client with the video player, and collects the results. In case of the Shaka player, the video is hosted using a local video server. With the YouTube player, we had to use YouTube as video source. Because of this, we had two differently encoded versions of the same video. This is further described in Section 3.5.

3.4 Network Limiter

The bandwidth is controlled by bandwidth traces, which reflect real-world scenarios. These were provided by Riiser et al. using a notebook and a 3G modem for measuring download speed, and a GPS module for localization [15]. The team investigated bandwidth rates on different mobile travel situations. During the measurements, files were downloaded via HTTP constantly. As this is the same technique used by adaptive video streaming the same behavior would be experienced by a video streaming client. The bandwidth traces reflect situations as they are typical for commuting, using different ways of transportation (bus, car, ferry, metro, tram, and train). We focus on mobile scenarios, because in these the bandwidth is varying, while in stationary situations, there is often more bandwidth available, which also has a lower variation over time. Numbers about the bandwidth scenarios can be found in Table 1. These bandwidth traces are reproduced using *tc*[3] which configures the packet scheduler of the Linux kernel of a dedicated Linux machine.

Table 1. Bandwidth characteristics of the bandwidth traces.

Scenario	Mean	Std. Deviation	$10^{\text{th}}/90^{\text{th}}$ Percentile	Min/Max
Bus	244.7 kB s^{-1}	164.1 kB s^{-1}	34.6 kB s^{-1}/ 499.6 kB s^{-1}	25.3 B s^{-1}/ 773.3 kB s^{-1}
Car	67.3 kB s^{-1}	90.8 kB s^{-1}	3.26 B s^{-1}/ 208.5 kB s^{-1}	0 B s^{-1}/ 545.3 kB s^{-1}
Ferry	161.0 kB s^{-1}	146.2 kB s^{-1}	342 B s^{-1}/ 365.1 kB s^{-1}	0 B s^{-1}/ 848.9 kB s^{-1}
Metro	72.1 kB s^{-1}	74.5 kB s^{-1}	342 B s^{-1}/ 183.0 kB s^{-1}	0 B s^{-1}/ 593.4 kB s^{-1}
Train	134.2 kB s^{-1}	119.4 kB s^{-1}	188 B s^{-1}/ 292.4 kB s^{-1}	14.9 B s^{-1}/ 1006 kB s^{-1}
Tram	100.4 kB s^{-1}	62.4 kB s^{-1}	23.2 kB s^{-1}/ 169.3 kB s^{-1}	53.5 B s^{-1}/ 824 kB s^{-1}

3.5 Video Server

The video players use different video sources. While the video source of the Shaka player can be configured arbitrarily, the YouTube player can only play videos from

[3] `http://manpages.ubuntu.com/manpages/xenial/en/man8/tc.8.html`

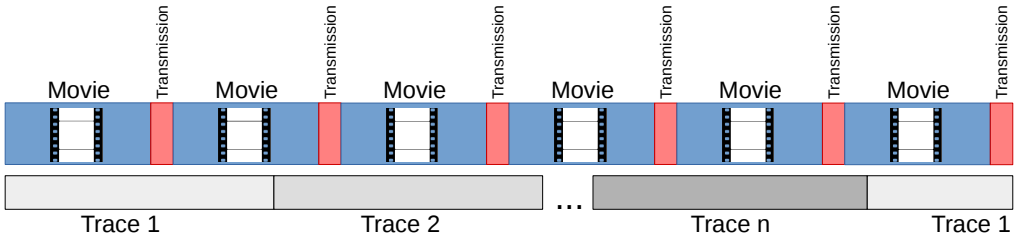


Fig. 2. Repetition of bandwidth traces and videos. When the movie finished playing, the playback statistics are sent to the control server. After that, the movie is played again.

YouTube. As a result, we had to use different test videos. The video for the Shaka player was provided by a local web server. The YouTube player fetched a video directly from YouTube. Because this relies on an Internet connection, during the experiments the Internet connectivity was continuously monitored to detect impairments.

Both video players used the same test movie with a length of 9 min 54 s, but in a different encoding and different representation layers. While the video for the Shaka player comes with three quality (Average data rate 79.99 kB s^{-1}, 178.60 kB s^{-1}, 482.69 kB s^{-1}) levels, the YouTube video has seven quality levels (Average data rate 13.41 kB s^{-1}, 30.17 kB s^{-1}, 40.70 kB s^{-1}, 80.08 kB s^{-1}, 150.60 kB s^{-1}, and 268.72 kB s^{-1}). This means the YouTube player has a lower minimum and maximum bandwidth and more possibilities to adapt the video quality to the current available bandwidth.

3.6 Mode of Operation

The mode of operation is as follows: The bandwidth traces of one scenario are infinitely repeated by the bandwidth limiter. At the same time, the video playback is repeated continuously, so that it starts at different positions of the bandwidth traces. This process is also shown in Figure 2. Using the players' API, every second the current playback status, the buffer level and video quality are recorded, which are then sent to the control server after a video playback has finished. Additionally, this is also done whenever a stalling or adaptation event occurs. This way, we obtain a precise history of the players' behavior.

4 Results

In this Section, we present the results of the experiments. Each experiment can be identified by two different parameters: the video player (either YouTube player or Shaka player) and the bandwidth scenario (Bus, Car, Ferry, Metro, Train, Tram or Unlimited). The bandwidth scenario determines the precise available bandwidth at every time. In case of the Shaka player, also the buffer configuration can be changed. Then, the rebuffering goal and the buffer size can be configured, both parameters are explained in Section 2.3. The video player plays the test video. All experiments use the same test video, but the two different players use a different version of it.

4.1 Shaka Player

With the Shaka player, we conducted experiments with different buffer configurations. The rebuffering goal sets the lower threshold when the playback starts, and the buffer size determines the maximum amount of video in the buffer. As all experiments are run in real-time, we limited the experiments to four different configurations: the default values with a buffer size of ten and a rebuffering goal of two seconds, a rebuffering goal of ten seconds, together with an increased buffer size of 90 and 180 seconds, and finally a high rebuffering goal of 20 seconds and 180 seconds buffer size.

Played Resolution and Adaptation Events In Figure 3, the number of adaptation events per video is plotted. An adaptation event means that the player changes the playback quality, thus selects another representation layer.

The number of adaptation events clearly differs between the different bandwidth traces. The bandwidth scenarios with the highest numbers of adaptation events are 'Bus', 'Ferry', and 'Train' with a median number of adaptation events between five and six, respectively three and four, and two in case of 'Train'. Those bandwidth scenarios are the three with the highest average bandwidth, and play a significant share of the video in medium quality, while in all other scenarios nearly all the time the lowest quality is played. As it can be expected, with unlimited bandwidth, always one single adaptation event occurs, when the player switches from the initial quality to the highest representation layer. It is notable, that the buffer configuration has no influence on the number of adaptation events.

Figure 4 shows the distribution of the resolutions, in which the video was played. In case of unlimited bandwidth, most of the time the video was played at maximum quality. When the bandwidth was limited, the highest quality was never played, most time the lowest quality was selected. Only in the scenarios 'Bus', 'Ferry', and 'Train' the medium quality was played for a significant time. It can be assumed, that only in these scenarios the bandwidth was high enough, see Table 1.

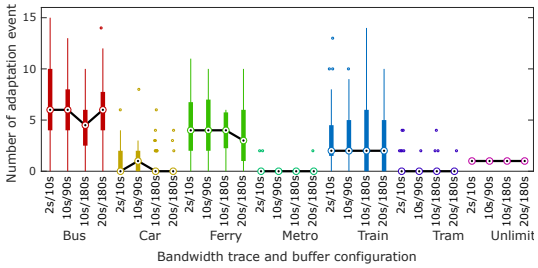


Fig. 3. Shaka player: adaptation events.

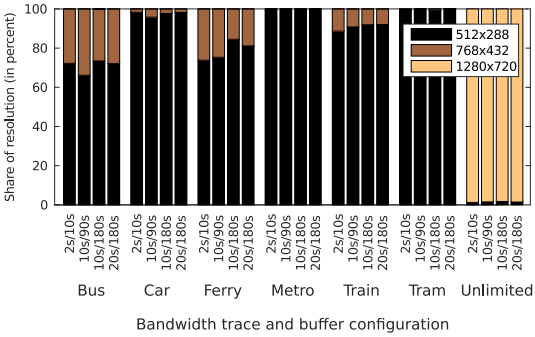


Fig. 4. Shaka player: played resolution.

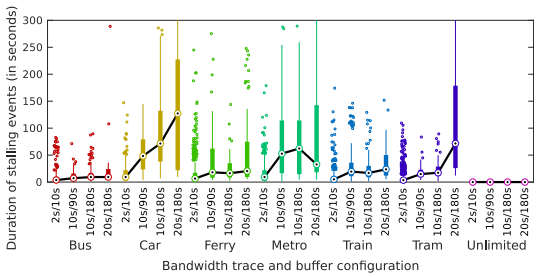


Fig. 5. Shaka player: stalling durations.

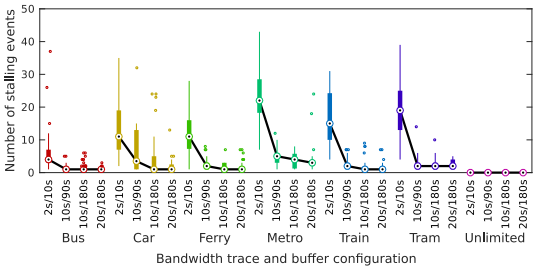


Fig. 6. Shaka player: stalling events.

Now we take a look at the stalling events. Figure 6 shows their number, while Figure 5 shows their duration. It can clearly be seen, that in all cases, where the bandwidth was limited, the lower the rebuffering goal was set, the more stalling events occurred. Especially in the configuration with the smallest buffer, in median there were a large number of stalling events. Compared to the lowest configuration, there is a huge improvement using the buffer size of ten seconds and the 90 s rebuffering goal. Whereas, there is no significant reduction of the number of stalling events between a rebuffering goal of ten and twenty seconds and a buffer size of 180 s. As can be expected, in case of unlimited bandwidth, no stalling

happened.

When looking at the duration of the stalling events, it can be found that the median length of the stalling events was significantly longer with a higher rebuffering goal. The exception in the 'Metro' scenario and the 20 second rebuffering goal can be explained by the methodology, where the video starts at arbitrary positions of the bandwidth traces. In this case, the variance is much higher than the cases with a smaller buffer and a lower rebuffering goal.

4.2 YouTube Player

In contrary to the Shaka player, the buffer configuration of the YouTube player cannot be configured. So the experiments only differ in the bandwidth scenario that was used.

Played Resolution and Adaptation Events In Figure 7 the number of adaptation events, that occurred during one video, is showed. We can see that the median number of adaptation events is less or equal four. In the scenario 'Metro', it was exactly four, in case of the 'Bus' the least adaptation events of all scenarios with limited bandwidth happen, the median number is one. Interestingly, in case of unlimited bandwidth, no adaptation events happen. This means, that from the beginning on the highest quality was selected, in contrary to the Shaka player, where in this scenario exactly one adaptation event occurs.

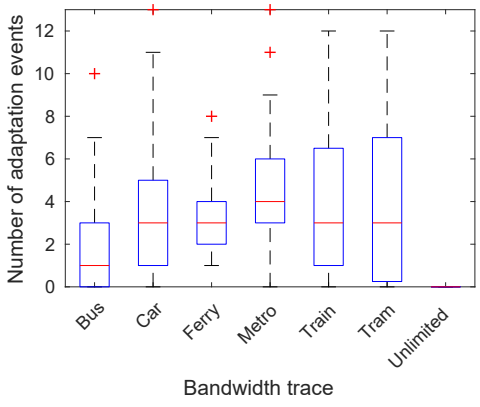


Fig. 7. YouTube: number of adaptation events.

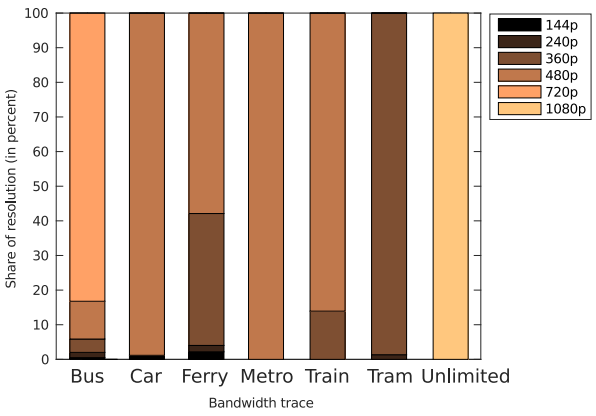


Fig. 8. YouTube: played resolution.

Figure 8 shows the share of the resolution that has been played, which ranges from 144p to 1080p. In case of unlimited bandwidth, the highest quality was played all the time. In the 'Bus' scenario, 80% of the time the 720p version was played. In all other scenarios a lower quality was played. The lowest quality was played in the 'Tram' scenario, where the player selected the 360p representation layer most of the time.

Stalling In Figure 10, the number of stalling events per video is showed for every bandwidth scenario, whereas Figure 9 shows their duration. It can be noted, that the median number of stalling events in case of 'Bus' and 'Ferry' is zero. In the scenarios 'Train' and 'Tram' it is one. A massive number of stalling events occurred in the 'Metro' and the 'Car' scenario, which also have a large variance in the number of these events.

The length of stalling events is mixed. In all scenarios apart of 'Train', the median length is below ten seconds. But the stalling events can also be very long, especially in the scenarios 'Car, 'Metro', and 'Train', where there are stalling times up to 150 s.

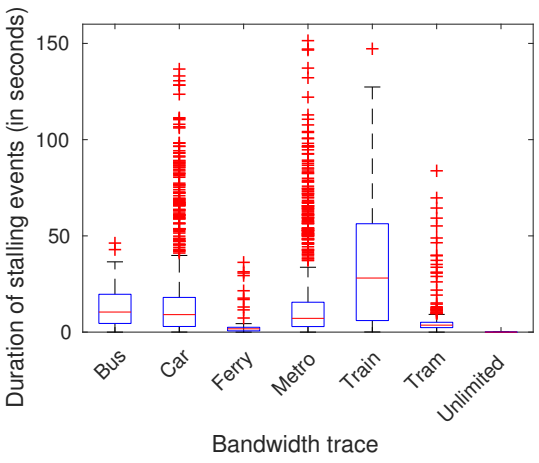


Fig. 9. YouTube: stalling duration.

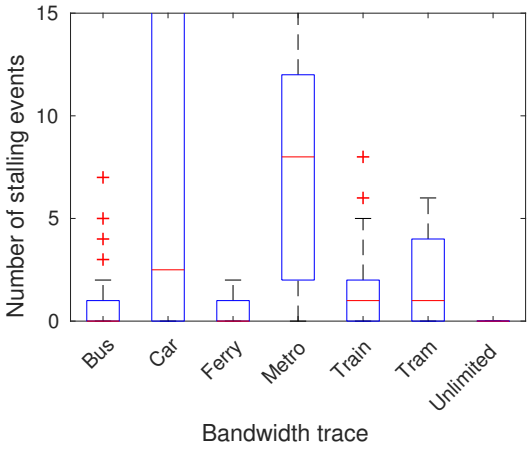


Fig. 10. YouTube: number of stalling events.

4.3 Conclusions Regarding Wasted Traffic

During all experiments, the buffer level was monitored. In order to assess the potentially wasted bandwidth, we evaluated the buffer status during the playback

of the video. This is done for the YouTube player and the Shaka player. The Shaka player is evaluated with four different buffer configurations. In the following we focus on two configurations one with the buffer size of 90 s and one with 180 s, each with a ten second rebuffering goal. During the playback of the video, each second the current buffer level is written to a log file. Only the readings while the video is playing are selected. Now, for every second of the video the average buffer level is calculated and plotted. This is repeated for every bandwidth scenario.

When the user aborts the playback of the video, the video in the buffer is not needed, hence the bandwidth used for it is wasted. In these diagrams, the average buffer fill level for every scenario can be read directly.

Figure 11 shows this for the Shaka player with a buffer size of 90 s, while in Figure 12 the Shaka player comes with a buffer size of 180 s. In the scenario with the 90 s buffer size, in all scenarios with limited bandwidth, there is a buildup-phase in which the average buffer level rises, a steady phase, in which the buffer level remains at about the same level and a final degradation phase, in which the buffer is emptied, until at the end of the video when the buffer is empty. In the scenario with the buffer size of 180 s, which is showed in Figure 12 the steady phase is less distinguished, the average buffer level slowly increases until the degradation phase. In both scenarios, in case of unlimited bandwidth, the buffer is filled completely all the time.

Figure 13 shows the case of the YouTube player. It takes about 200 s until the average buffer level reaches a stable level for all scenarios. The average buffer level never rises above 120 s, during the stable phase it remains between 80 s and 120 s. Interestingly, in case of unlimited bandwidth, the buffer level remains at about 40 s.

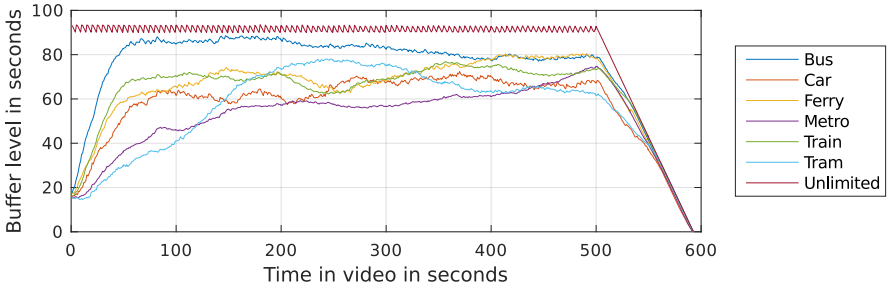


Fig. 11. Shaka player with a buffer size of 90 s: average buffer level over all test runs during the video, thus the potentially wasted bandwidth.

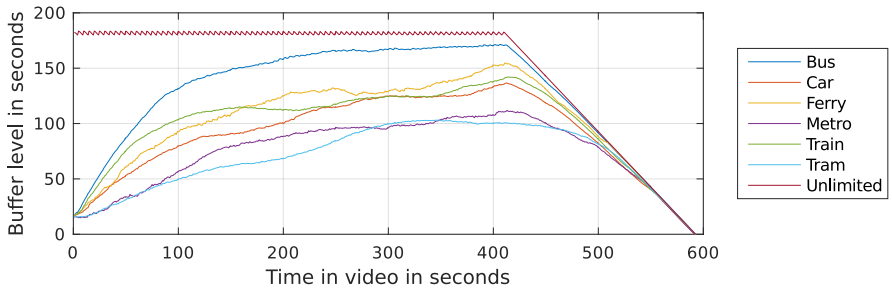


Fig. 12. Shaka player with a buffer size of 180 s: average buffer level over all test runs during the video, thus the potentially wasted bandwidth.

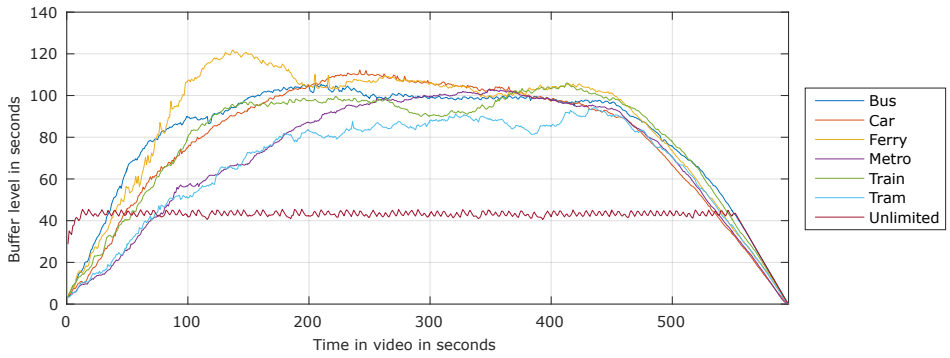


Fig. 13. YouTube player: average buffer level over all test runs during the video, thus the potentially wasted bandwidth.

5 Conclusions

In this paper, we investigated the influence of the buffer size on the playback behavior of two different HAS players in real commuting scenarios. We compared different buffer configurations of the Shaka player and the standard behavior of the YouTube player.

We could see that a larger buffer, above a certain size, does not increase the played video quality. That means, the maximum buffer size does not influence playback so it could theoretically be as high as anyhow possible. However, in terms of bandwidth consumption, the possible wasted traffic has to be considered. This study showed, that considering the large share of the overall traffic, that video streaming accounts for [1], on provider scale, there is a massive saving potential by limiting the buffer size accordingly. This applies especially in the scenario where

the available bandwidth is much higher than the average video bandwidth. In these cases, the buffer gets filled almost immediately, and as soon as the user aborts the video, all this bandwidth is wasted. As stated before, 20% of all YouTube videos were aborted by users within the first ten seconds. With the Shaka player and a buffer size of 180 s, this means that in those cases at least 18 times more video was transmitted than actually played. In the same scenario, the YouTube player still downloads more than 30 s of video unnecessarily.

One possibility to solve this issue is to leave it to the user to select the buffer size. In [16] it was proposed to allow the user to manually select the players' behavior. But as users typically stick to the default settings [3], this will probably only have success in limited cases, like if the user wants to watch a movie without interruptions, and knows that the available bandwidth is not sufficient. To do so, knowledge about the bandwidth and planning is required, so that most users will not use this possibility.

This leads to the conclusion that the optimal buffer size is limited. Above a certain buffer size, the overall quality does not further increase. For the sake of economic bandwidth usage, it should be an option to strictly limit the buffer size in case the available bandwidth is much higher than the video data rate. This can be done by making the buffer size adaptive, like done in [12]. Another option is the YouTube approach, where the buffer size depends on the video quality [17]. In case of high available bandwidth, the player selects the best video quality, and the buffer size is automatically lowered. It can be stated that an implementation is easy, as rate-adaptive video players like the YouTube player or the Shaka player already come with a module for bandwidth estimation.

References

1. Sandvine: Global Internet Phenomena Report: Africa, Middle East, and North America (2015) https://www.sandvine.com/downloads/general/global-internet-phenomena/2015/global-internet-phenomena-report-latin-america-and-north-america.pdf. Last accessed on 14th January 2017.
2. Hossfeld, T, Egger, S. Schatz, R.: Initial Delay vs. Interruptions: Between the Devil and the Deep Blue Sea, 2012 Fourth International Workshop on Quality of Multimedia Experience, Yarra Valley, VIC, pp. 1-6 (2012)
3. Finamore, A., Mellia, M., Munafò, M. M., Torres, R., Rao, S. G.: YouTube Everywhere: Impact of Device and Infrastructure Synergies on User Experience. In: Proceedings

of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, pp. 345-360 (2011)
4. Sen, S., Joe-Wong, C., Ha, S., Chiang, M.: Pricing data: A look at past proposals, current plans, and future trends. CoRR, abs/1201.4197 (2012)
5. Balasubramanian, N., Balasubramanian, A., Venkataramani, A. (2009, November). Energy consumption in mobile phones: a measurement study and implications for network applications. In: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, pp. 280-293 (2009)
6. Seufert, M., Egger, S., Slanina, M., Zinner, T., Hobfeld, T., Tran-Gia, P. (2015). A survey on quality of experience of HTTP adaptive streaming. In: IEEE Communications Surveys & Tutorials, 17(1), pp. 469-492 (2015)
7. Mangla, T., Theera-Ampornpunt, N., Ammar, M., Zegura, E., Bagchi, S.: Video through a crystal ball: effect of bandwidth prediction quality on adaptive streaming in mobile environments. In: Proceedings of the 8th International Workshop on Mobile Video, p. 1 (2016)
8. Riiser, H., Endestad, T., Vigmostad, P., Griwodz, C., Halvorsen, P.: Video streaming using a location-based bandwidth-lookup service for bitrate planning. In: ACM Transactions on Multimedia Computing, Communications, and Applications, 8(3), 24 (2012)
9. Hao, J., Zimmermann, R., Ma, H.: GTube: geo-predictive video streaming over HTTP in mobile environments. In: Proceedings of the 5th ACM Multimedia Systems Conference, pp. 259-270 (2012)
10. Li, X., Dong, M., Ma, Z., Fernandes, F. C.: Greentube: power optimization for mobile videostreaming via dynamic cache management. In: Proceedings of the 20th ACM international conference on Multimedia, pp. 279-288 (2012)
11. Tian, G., Liu, Y.: On adaptive HTTP streaming to mobile devices. In: Packet Video Workshop (PV), 2013 20th International, pp. 1-8 (2013)
12. Wu, D., Huang, J., He, J., Chen, M., Zhang, G.: Toward cost-effective mobile video streaming via smart cache with adaptive thresholding. In: IEEE Transactions on Broadcasting, 61(4), pp. 639-650 (2015)
13. Wamser, F., Seufert, M., Casas, P., Irmer, R., Tran-Gia, P., Schatz, R.: YoMoApp: A tool for analyzing QoE of YouTube HTTP adaptive streaming in mobile networks. In: 2015 European Conference on Networks and Communications (EuCNC), pp. 239-243 (2015)
14. Casas, P., Seufert, M., Schatz, R.: YOUQMON: A system for on-line monitoring of YouTube QoE in operational 3G networks. In: ACM SIGMETRICS Performance Evaluation Review, 41(2), pp. 44-46 (2013)
15. Riiser, H., Vigmostad, P., Griwodz, C., Halvorsen, P.: Commute path bandwidth traces from 3G networks: analysis and applications. In: Proceedings of the 4th ACM Multimedia Systems Conference, pp. 114-118 (2013)
16. Hoßfeld, T., Moldovan, C., Schwartz, C.: To each according to his needs: Dimensioning video buffer for specific user profiles and behavior. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 1249-1254 (2015)

17. Sieber, C., Blenk, A., Hinteregger, M., Kellerer, W.: The cost of aggressive HTTP adaptive streaming: Quantifying YouTube's redundant traffic. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 1261-1267 (2015)

On the Probability of Meeting Deadlines in IP-based Real-time Communication

Alexander Beifuß

University of Hamburg,
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany

Abstract. Today we observe many different real-time services on the Internet, though, this network cannot provide any guarantees regarding the quality of data transmissions. For this reason, various distinct approaches (e.g., codecs, buffers, algorithms, and protocols) have been developed with the goal to improve the transmission of real-time data. Even if each solution is unique on its own, all of them aim for the same objective: to make sure that the data arrives at the receiver in time. In this paper, we investigate this aspect and present a straightforward and uniform approach to estimate the end-to-end delay of packet trains based on the convolution of the probability distributions of an initial delay and of the inter-arrival times at the receiver. We provide a detailed mathematical formulation of our model and share considerations regarding the applicability of our model. We implemented a proof-of-concept and conducted an empirical study with real measurements to evaluate our model.

Key words: real-time services, inter-arrival times, end-to-end delay, packet trains, probability distributions, convolution

1 Introduction

Today many different real-time services use the Internet: video streaming, video conferencing, and voice over IP (VoIP) are well-known examples, but the spectrum is indeed much broader (e.g., in the fields of mobile computing [24], gaming [8], telemedicine [10], sensor networks [11][2], and robotics [4][25]).

According to Cisco, alone “the sum of all forms of IP video, which includes Internet video, IP VoD, video files exchanged through file sharing, video-streamed gaming, and video conferencing, will continue to be in the range of 80 to 90 percent of total IP traffic” [7] until 2020. This underlines the considerable importance of the subject of this work.

By definition, real-time services have to respect delay bounds, otherwise they are not able to provide the required quality to the end user [12]. Although, there are techniques to provide priority to real-time flows (e.g. IntServ, DiffServ, or MPLS), the Internet cannot generally provide guarantees when data is transmitted because of its store-and-forward architecture and its limited resources which are shared among the users. Hence, the Internet generally offers a best effort transmission service only. Many different techniques have been developed that address this issue and the challenging task to improve the transmission of real-time data. For example, information dispersal schemes aggregate resources of multiple networks [29], adaptive encoding [28] adjusts the amount of data according to the available resources, playout buffers [16] relax the deadline constraints, and prioritization strategies [3] favor real-time traffic over other best effort traffic.

Even if these techniques are very diverse, all of them aim for the same objective: to increase the probability that the real-time data arrives timely at the receiver. Therefore, gaining knowledge about the probability at which network packets will arrive in time is the key to assess and improve existing approaches and to inspire for novel solutions.

Many researchers have considered this topic. Rubin [23] presented a queuing theory model to determine the distribution of message delays over a multi-channel communication path. His model considers messages with arbitrary distributed message lengths and divided into fixed length packets arriving according to a Poisson process. Wolfinger [27] modeled the end-to-end delay of file transfers by means of analytical tandem queuing networks, where the total end-to-end delay for a file transfer was decomposed into the average delay of the first message and the sum of average inter-arrival times of the messages at the receiver. Evequoz and Tropper [9] developed a heuristic to compute the mean end-to-end delay of multi-packet messages in lightly loaded packet switching networks with flow control. Papagiannaki et al. [21] found that the tail of the single-hop queuing delay through a router fits a Weibull distribution. Hernández and Phillips [15] proposed

a Weibull Mixture Model to characterize the end-to-end Internet delay for single packets. Burchard and Liebeherr [6] as well as Fidler [13] use network calculus to derive probabilistic delay bounds for tandem servers. Bisnik and Abouzeid [5] modeled the random access MAC of IEEE 802.11 and derived a closed expression for the end-to-end delay in multi-hop wireless networks.

In this paper, we investigate the end-to-end delay of packet trains from a black-box perspective and present a unified empirical model to estimate the probability distribution of end-to-end delays of packet trains. In addition to the mathematical formulation of our model we evaluate the quality of our model in an empirical case-study with real-world measurements.

The remaining part of this paper is structured as follows. First, we present our communication model and derive our model for estimating end-to-end delays of packet trains in Section 2, which is evaluated in Section 3. Considerations regarding the on-line applicability of our model are shared in Section 4. Finally, we conclude our work and provide an outlook for future work in Section 5.

2 Approach

In this section we present a rather general communication model of real-time systems and provide our assumptions and some basic definitions. Based on that, we derive a probabilistic model of the end-to-end delay of packet trains.

Communication System Model We consider basic sender-receiver systems with a setup as illustrated in Fig. 1. Such systems typically consist of a sender S, a receiver R, and a communication network CN.

First, we specify two target interfaces: IF_S at the sender and IF_R at the receiver to decompose the system into three components: the sender environment E_S, the receiver environment E_R, and the transport system TS which consists of CN and the communication system parts within S and R. In this work, TS is considered to be a black box.

E_S generates send events s_i according to a certain process. The first bit of the packets that correspond to s_i will arrive at IF_S at the time $t(s_i)$. TS transmits the data from S to R, and IF_R will deliver the data to E_R, so that a receive event r_i is observed (in consequence of s_i) when the last bit is received at the time $t(r_i)$. Let us note $t(r_i) > t(s_i)$.

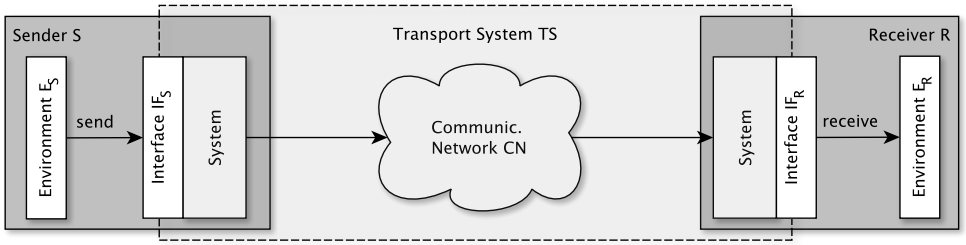


Fig. 1: Illustration of the communication system model

By definition, a real-time service demands a delay bound δ and transmissions with a delay $> \delta$ are worthless. Thus, for real-time systems it is of general importance to estimate whether a packet will arrive in time or not.

End-to-end Delay We define *end-to-end delay* $\Delta(s_i, r_i)$ as the length of the time interval between a send event s_i and the consequent receive event r_i observed at IF_S and IF_R, respectively (cf. Fig. 2a).

$$\Delta(s_i, r_i) := t(r_i) - t(s_i), t(r_i) > t(s_i) \tag{1}$$

Inter-arrival Time We define *inter-arrival time* (IAT) $\tau(e_{i-1}, e_i)$ as the length of the time interval between two consecutive events e_{i-1} and e_i of the same arrival process, observed at the interface IF (cf. Fig. 2b). In addition to the expression $\tau(e_{i-1}, e_i)$ we also use the term "IAT at position i" and the identifier $\tau(i)$ if the interface is clear. Otherwise, $\tau_S(\ldots)$ and $\tau_R(\ldots)$ denote the inter-arrival times at the sender S and the receiver R, respectively.

$$\tau(i) := \tau(e_{i-1}, e_i) := t(e_i) - t(e_{i-1}), t(e_i) \geq t(e_{i-1}) \tag{2}$$

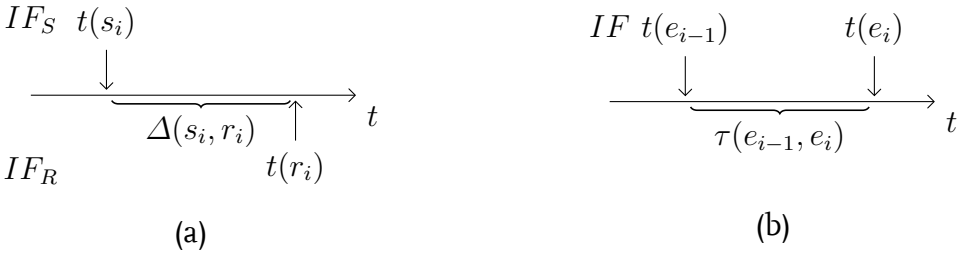


Fig. 2: Visualization of (a) end-to-end delay, and (b) inter-arrival time

Packet Trains Without loss of generality, we consider the sender process to produce so-called *trains* of $n \geq 1, n \in \mathbb{N}^+$ send requests at once, where $\mathbb{N}^+$ denotes the set of natural numbers without 0 ($\mathbb{N}^+ := \{1, 2, 3, \ldots\}$). We assume that all packets of the train are delivered to IF_S back-to-back at the same instant, such that

$$t(s_j) = t(s_k), \forall j, k \textit{ with } 1 \leq j \leq k \leq n \tag{3}$$

A real-world example for a packet train generating process is live video streaming. A camera captures pictures at a specific frame rate (e.g., 24 Hz). The picture data is likely to exceed the maximum allowed length of IP packets, thus, single pictures are typically segmented and carried by $n > 1$ IP packets.

In this work, we focus on packet trains. As of now, $\{s_i : 1 \leq i \leq n\}$ denotes the ith send event of the packet train. Along with that, $\{r_i : 1 \leq i \leq n\}$ denotes the ith delivery event of a packet train. If not stated otherwise, s and r now refer to the send and receive events of packet trains, respectively. Accordingly, a send event of a train of the size of n packets occurs when the first bit of the first packet is sent to IF_S, and a receive event occurs when the last bit of the nth packet train data is delivered by IF_R. Therefore, we define $t(s) := t(s_1)$ and $t(r) := t(r_n)$.

We postulate, that the packets of a packet train that propagates through TS will suffer a constant delay (i.e., the propagation delay), so that, we can conclude $\Delta(s, r_1) > 0$. Moreover, packets of a certain packet train will disperse during transmission because of variable delays, e.g., variable queuing delays caused by cross traffic or due to possibly varying data rates, e.g., in the case of dynamic resource allocation (e.g., WiFi RTS/CTS or LTE SC-FDMA). As we suppose that TS does not reorder packets, we can conclude $\tau_R(r_i, r_{i+1}) > 0$. This assumption is generally not completely fulfilled, but, at least, reordering is rather unlikely in today's Internet and in practice we can simply detect reordering and exclude corresponding packets before applying our model.

According to (1) we define packet train end-to-end delay as $\hat{\Delta}_n := \Delta(s, r_n)$. Fig. 3 illustrates that $\hat{\Delta}_n$ can be decomposed into the sum of an initial delay (for the transmission of the first packet), followed by a sum of $n - 1$ IATs (for the delivery process of the remaining packets).

$$\hat{\Delta}_n = \Delta(s, r_1) + \sum_{i=2}^{n} \tau(i) \tag{4}$$

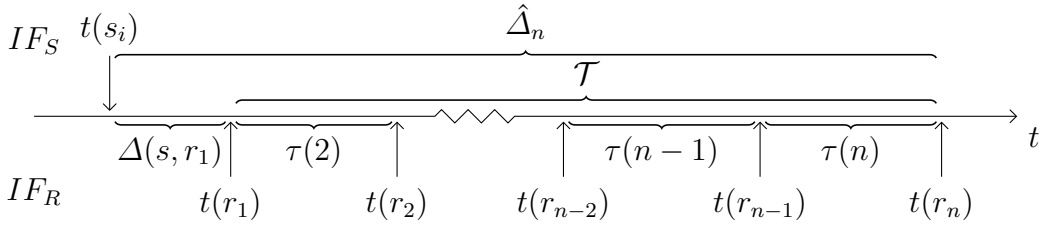


Fig. 3: Visual explanation of inter-arrival time

Generally, the probability distribution of the sum $Z = X + Y$ of independent continuous random variables with densities $f(x)$ and $g(y)$ equals the convolution of the individual probability distributions of the random variables [17].

$$h(z) = (f + g)(z) = \int_{-\infty}^{\infty} f(x)g(z - x)dx \tag{5}$$

We assume the ith term of $\hat{\Delta}_n$ to be an independent random variable X_i that follows a particular probability distribution $\mathcal{D}_i$. If all the individual $\mathcal{D}_i$ are known, then, we can calculate the probability distributions of the end-to-end delay $\hat{\mathcal{D}}_n$ of $\hat{\Delta}_n$ by means of convolution:

$$\hat{\mathcal{D}}_n \sim \sum_{i=1}^{n} \mathcal{D}_n : \tag{6}$$

Let X denote a random variable that follows the distribution $\hat{\mathcal{D}}_n$ and let δ denote a variable which represents the maximum accepted delay. Then, our model enables to calculate the probability $P[X \leq \delta]$ that X is below or equal to δ for packet train sizes $\leq n$.

It should be stressed here that the assumption of independent $\mathcal{D}_i$'s is not generally met in real-world settings. For example, in case of prevalent packet reordering or frequent retransmissions, like possibly with TCP, delays can correlate. Thus, this independence assumption has to be validated by measurements before this model can be applied.

3 Evaluation

We evaluate our model by means of real-world experiments. In our testbed we measure frequencies of time intervals of end-to-end packet delays, end-to-end delays for packet trains, and inter-arrival times at the receiver. To transform the time discrete distributions into corresponding time continuous distributions we fit different well-known distributions based on the measured frequencies to approxi-

mate the true distributions. Finally, we apply our model to estimate the distribution of the delay for packets at a specific position and compare the results with the delay traces of the measured data.

Testbed Setup Fig. 4 illustrates the loop-like setup of our testbed. A standard PC (MacBook Pro 2016) hosts the two environments of our system model: a load generator process as well as a load sink process. Please note, that this setup does not require any synchronization of physically distributed clocks. Only the PC's clock is utilized, thus, one-way end-to-end delay measurements are as precise as possible. This setup should achieve comparable accuracy with less effort than the one way delay measurement setup presented by Hernandez and Magaña [14].

The PC is equipped with two network adapters, each providing connectivity to the Internet. On the one hand, a network interface card (NIC) connects the PC via a wired Ethernet link to our department's LAN. The LAN is connected to the Internet via the German Research Network (DFN). The PC has a public IP address and can be reached from the Internet on a special port. On the other hand, USB tethering enables the PC to use the smartphone as a gateway to the LTE network of O2 (Telefonica). The smartphone we use in our testbed is a Huawei P10 plus with Android (7.0). To avoid power-saving effects which might impact on transmissions, the smartphone is continuously plugged to the charger.

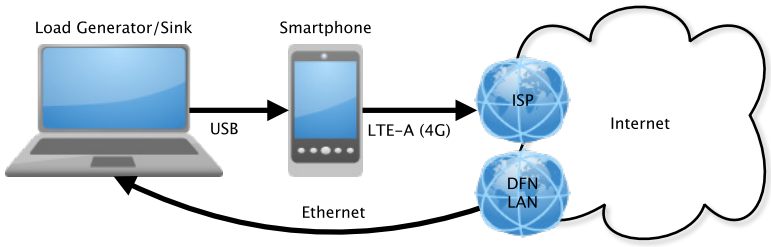


Fig. 4: Testbed setup

Measurements The load generator generates packet trains of the size of n packets. All packets of the train have the same Ethernet frame length l (in Byte). The packets are addressed to the public IP address of the PC's NIC, but we bound the sending socket to forward all packets to the smarthpone via USB tethering. The smartphone does not know the destination's IP-address, thus, the LTE-A uplink is

used to send the packets through the Internet, back to the NIC of our PC. In other words, the explained approach creates a setup that sends packets from the PC in a loop back to the PC which means it enables to conduct one-way end-to-end delay measurements without the use of a feedback channel.

In our experiments, E_S generated trains of n=50 packets for packet lengths of $l \in \{64, 128, 256, 512, 768, 1024, 1280\}$ Byte. For each l we generated 1.000 packet trains, which sums up to a total of 350.000 packets carrying ca. 200 MB. The IAT of packet trains was set to 100 ms; large enough to send all packets before the next packet train arrives, and small enough to avoid that the smartphone enters deeper power saving states. Finally, for each packet train we traced the end-to-end delays $\Delta(s, r_i)$ *for* $1 \leq i \leq n$ as well as the IAT $\tau(i)$ *for* $2 < i \leq n$.

Calibration Now, we describe our calibration process of our model and share some findings. Fig. 5 illustrates the mean IAT we observed for different packet positions i and different packet lengths l. On the one hand, this figure motivates that, at least in our scenario, the distribution of IAT depends on the packet length as we observe an increasing mean IAT on growing packet length. This is not surprising, because the IAT comprises the transmission delay which depends on the packet size. On the other hand, we found a strong evidence that the distribution of IAT can depend of the packet position i. For small packet lengths (i.e., l =64 B and l =128 B) we observe in average a much higher $\tau(i)$ *for* $i \in \{1, 2, 3\}$ than for the remaining IATs. We investigated on this effect and found that the distribution of the inter-arrival time can be multi-modal as shown by the histogram in Fig. 6.

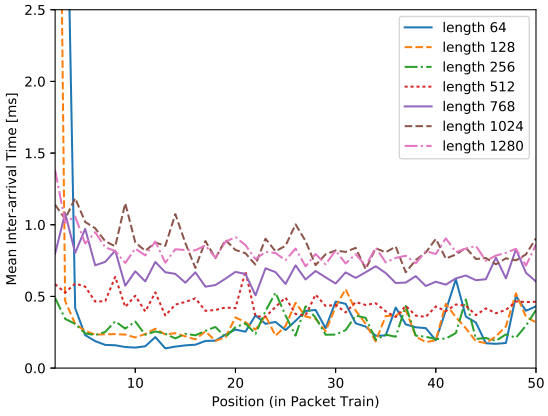


Fig. 5: Mean IAT as a function of i and l

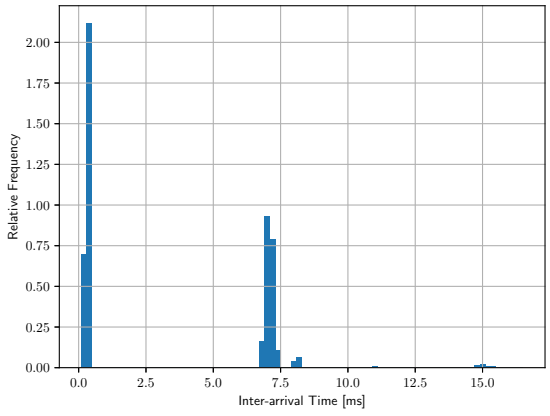


Fig. 6: Multimodal distribution of IAT (l=64 B, i=2)

In addition, we applied Pearson correlation calculations to analyze whether consecutive IATs exhibit a linear relationship. For this purpose, we calculated the correlation coefficients (r-values r) and the corresponding p-values p for consecutive IATs for all packet lengths. To assess whether the r-values are reliable we calculate $s = |r * (1 - p)|$. Large values of $|r|$ indicate a strong linear relationship. As large p-values indicate that the corresponding r-value is obtained rather by chance than through a real effect, large values of s can suggest a strong linear relationship which is unlikely by chance. The results of the correlation analyses are presented in Fig. 7 (for smaller packet lengths) and Fig. 8 (for larger packet sizes). We observe small values of s for all IAT positions at large packet lengths. Compared to that, the s values for smaller packet lengths partially vary, but at a closer look at the scatter plots (not shown here) we rarely found clear evidence for a linear relationship between consecutive IATs. Hence, we assume IATs to be mostly independent from each other.

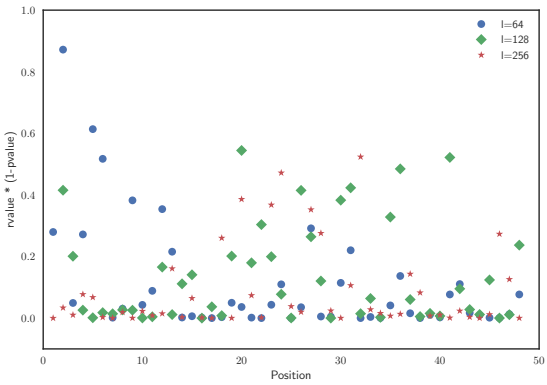


Fig. 7: $|r * (1 - p)|$ for small packets

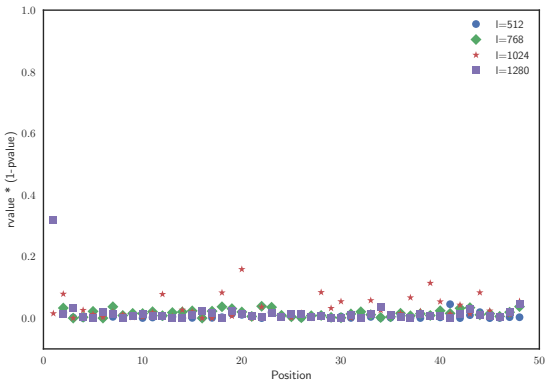
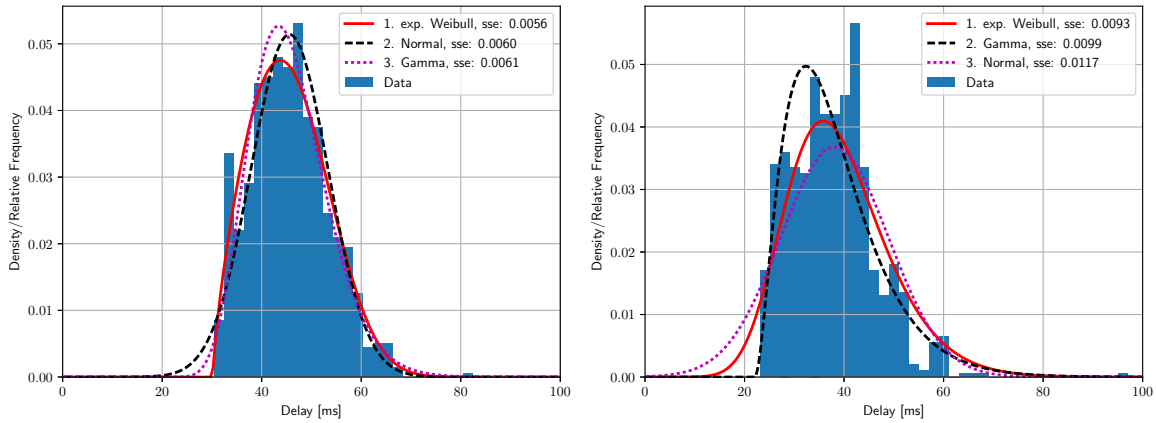


Fig. 8: $|r * (1 - p)|$ for large packets

We implemented an evaluation tool that automatically fits more than 80 distributions (from Python's scipy.stats module) to the traced data for the initial delay ($\Delta(s, r_1)$) for all packet lengths separately. For the fitting process, we sorted the traced data into histograms with a binwidth of 0.5 ms. Our tool also ranked the fitted distributions by the sum of square errors (SSE) which describes the difference between the fitting result and the histogram; we assume that the smaller the difference, the better is the fitted distribution (model). Afterwards, we applied statistical tests to assess the goodness-of-fit of our model. For this purpose, we used Kolmogorow-Smirnow(KS), Cramér–von Mises (CVM), and Anderson–Darling (AD) tests to compare each model with 100 samples of initial delay. For

all packet lengths, we observe that all three tests reach a p-value of more than 0.5. Hence, with the standard significance level of $\alpha = 0.05$ we would not reject the null hypotheses, which is that the samples were drawn from the provided distribution. However, the author would like to note that p-values must be treated with caution as stated in [26].

We found that our automated ranking process might give best scores to PDFs that do not model the rapidly increasing initial delay that we observe in the data. Such models should be avoided, as they would suggest that there is a probability that packets (and even packet trains) could arrive at the destination with a delay smaller than the actual minimal possible delay. Thus, we manually filtered out such bad fittings; examples of good and bad rated fittings are presented in Figs. 9a and 9b, respectively. Finally, we found that, in our case, $\Delta(s, r_1)$ is best modeled either by shifted Gamma or shifted exponentiated Weibull. Hence, we propose to consider different distributions depending on the packet size as presented in Table 1 which shows our fitting results for the $\Delta(s, r_1)$ sub-model (Δ). Among others, Gamma[1][21][18], Weibull[15][18], Pareto[30][14], as well as Lognormal[14] are distributions reported by other researches in diverse scenarios.



(a) Good rated fitting of exp. Weibull; exp. Weibull captures slope best

(b) Badly rated fitting of exp. Weibull; Gamma captures the slope better

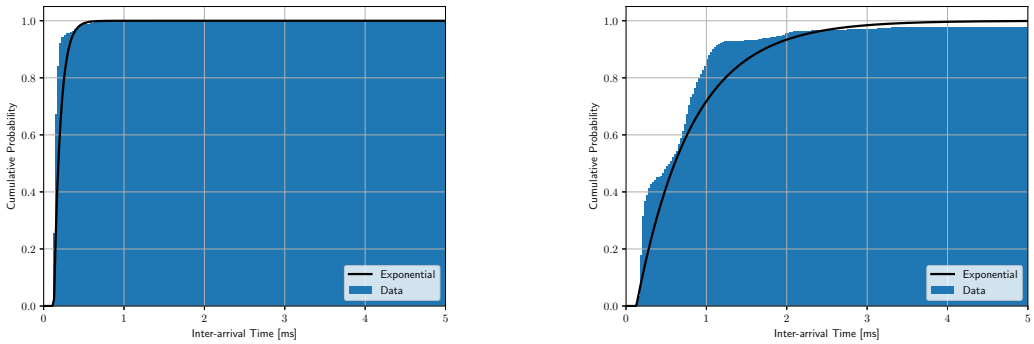
Fig. 9: Fitting result for $\Delta(s, r_1)$; (a) l=256 B, (b) l=64 B

$f(x; a, \lambda) = \frac{(\frac{1}{\lambda})^a x^{a-1} e^{\frac{-x}{\lambda}}}{\Gamma(a)}$ and $f(x; a, \lambda, b) = \frac{ab}{\lambda}\left[\frac{x}{\lambda}\right]^{a-1}\left[1 - e^{-(\frac{x}{\lambda})^k}\right]^{b-1} e^{-(\frac{x}{\lambda})^a}$ are the PDFs of the Gamma and the exp. Weibull distributions, respectively. a and b are shape parameters while λ is the scale parameter. The location parameter c shifts PDFs by a value of c such that $f_c(x; \ldots) = f(x - c; \ldots)$.

pkt. length [B]	Distribution	a	b	c	λ	SSE	KS p-value	CVM p-value	AD p-value
64	Gamma	2.759	-	22.163	5.772	0.0099	0.073791	0.156879	0.186607
128	Gamma	13.067	-	8.865	2.269	0.0070	0.097350	0.179962	0.194824
256	exp. Weibull	0.658	2.659	30.160	20.449	0.0056	0.136524	0.224433	0.225684
512	exp. Weibull	0.375	3.358	31.289	21.454	0.0058	0.072864	0.094055	0.088410
768	Gamma	10.019	-	20.787	2.410	0.0080	0.073133	0.097588	0.131122
1024	Gamma	16.740	-	12.205	1.937	0.0087	0.836100	0.682588	0.753509
1280	Gamma	11.313	-	18.846	2.24	0.0070	0.942033	0.897107	0.785359

Table 1: Fitting results for $\Delta(s, r_1)$ (initial delay submodel)

The approach for fitting the IAT sub-model ($\mathcal{T}$) was analogous to the fitting of the initial delay sub-model ($\Delta(s, r_1)$). We started with automatic fitting and ranking, before we manually compared the fitted distributions with the traced data (for some randomly chosen combinations of the packet length and the packet position). This process was very time-consuming and we found that automatic fitting and ranking is currently neither efficient nor reliable. Thus, we stepwise rejected to consider distribution types with partially weak fitting results. In the end, we simply decided to only fit according to exponential distribution. We do not claim that IATs are truly exponentially distributed, but from our point of view this distribution is a rather good choice, because it has only one shape parameter, thus, the fitting is less expensive and quite robust. Furthermore, the idea of exponentially distributed IATs matches to the understanding of queuing delays in the case of Poisson cross traffic. Due to lack of space, we cannot present all fitting results. Instead, we provide Figs. 10a and 10b to give an impression of the average performance of the Exponential distribution.



(a) $l = 256$ B i=14 (b) $l = 1280$ B i=8

Fig. 10: CDF of the fitted the exponential distribution for l, IAT at position i

Presentation and Discussion of the Results In this section we present and discuss results of our evaluation case-study. The task of the model is to estimate the distribution of the packet train end-to-end delay for a packet train size of n =40. To perform the convolution we applied a Monte Carlo simulation approach. For this purpose we first created vectors of 100k samples by drawing according to our calibrated submodels (the fitted distributions). Then, we added these up vectors component-wise to produce the packet train end-to-end delay model ($\hat{\Delta}$).

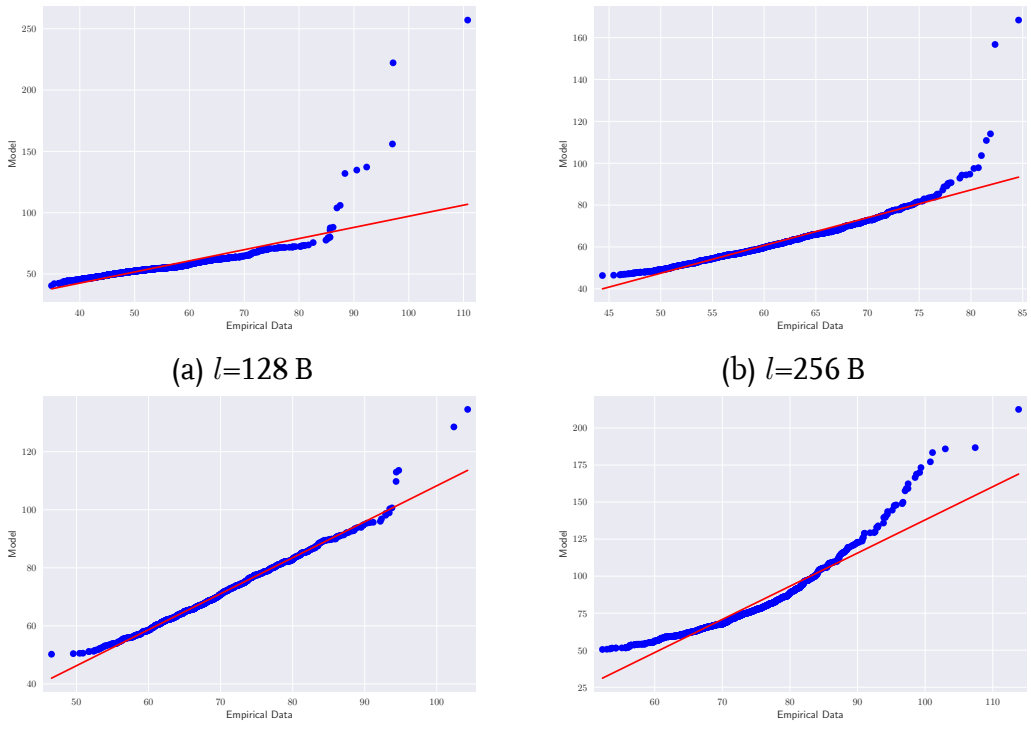
An overview of the performance of our model is illustrated by Figs. 12a to 12g. We observe, that our model estimates quite well the distribution of the measured packet train end-to-end delay; the shapes for our model and for the data do not overlap perfectly, but mainly, the model is able to render the shape and location without much deviation. Most prominent exception is for l =1024 B, where we observe major differences for the estimation and the measured data at the tail of the distributions. We also evaluate the quality of the estimations by means of percentiles. We consider that the tail of the end-to-end delay distribution is more relevant for real-time applications than the head. Therefore, in Table 2 we compare the 95th percentile (p_{95}) and the 99th percentile (p_{99}) of our model's estimation with that of the measured data. The percentile can be considered as the inverse of the CDF.

pkt. length [B]	Data p_{95} [ms]	Model p_{95} [ms]	E_{abs} [ms]	E_{rel}	Data p_{99} [ms]	Model p_{99} [ms]	E_{abs} [ms]	E_{rel}
64	69.3834	74.483	5.099	0.073	85.786	87.049	1.261	0.059
128	70.531	74.930	4.398	0.062	78.842	85.858	7.016	0.054
256	71.131	70.245	0.882	0.012	80.001	76.165	3.836	0.011
512	79.400	74.676	4.724	0.059	92.887	79.956	12.931	0.049
768	89.015	84.674	4.341	0.049	96.854	92.457	4.396	0.044
1024	118.662	89.322	29.339	0.247	159.201	96.991	62.210	0.184
1280	100.753	89.972	10.781	0.107	119.817	97.596	22.220	0.090

Table 2: Comparison of 95th and 99th percentiles estimated by the model and obtained from measured data by $E_{abs} = |x_0 - x|$ and $E_{rel} = E_{abs}/x$, with x being the estimated value and x_0 being the measured value.

From the percentiles we observe that our current model is struggling with precisely capturing the tail of the distribution, as indicated by the error of P_{95} and P_{99}; the relative errors are as high as ca. 25 % and we also observe slight deviations at the heads. The selection of Q-Q plots in Figs. 11a to 11d support this observation

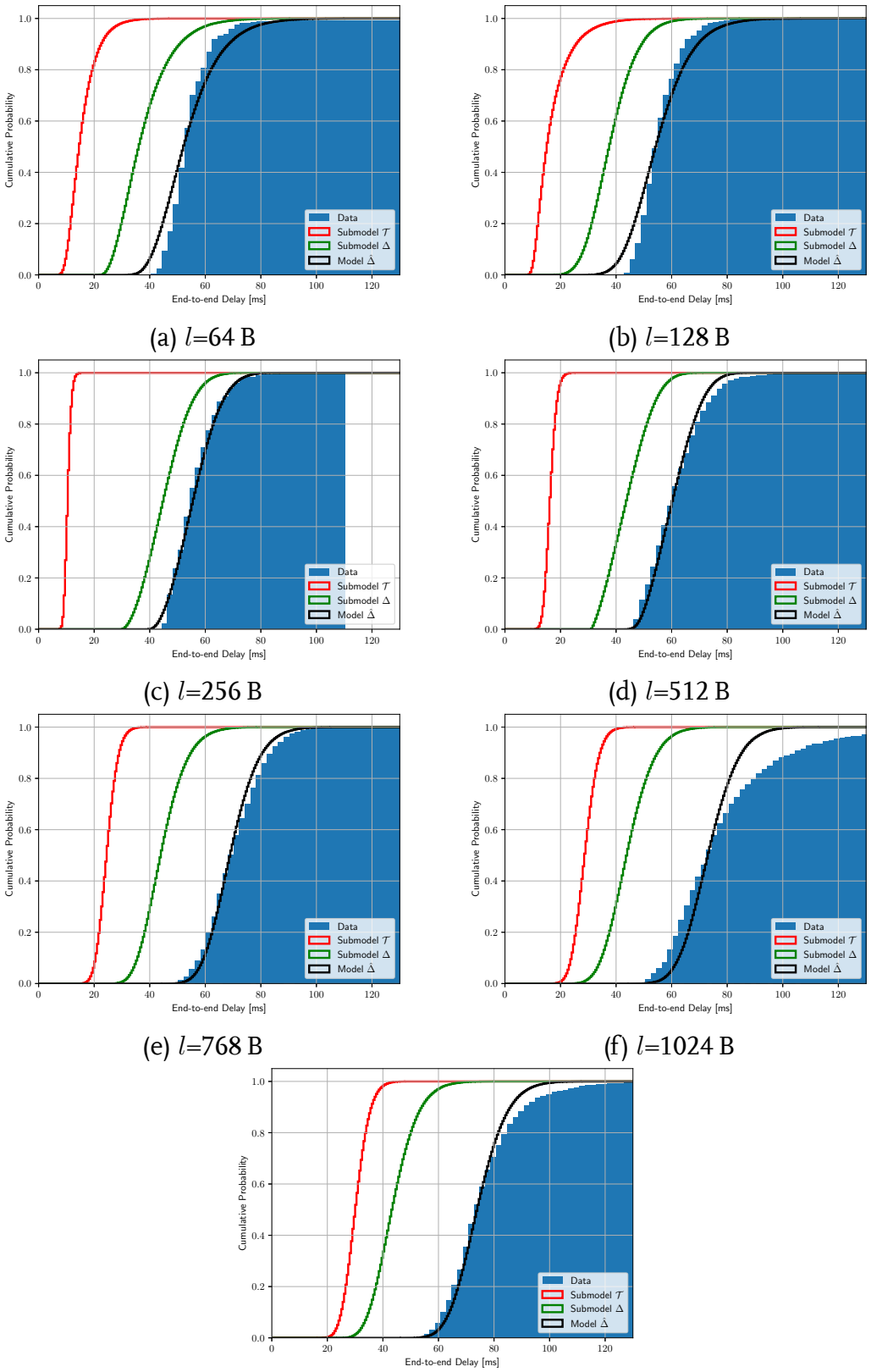
(due to lack of space we cannot show all plots). Moreover, Fig. 10b also reveals that the Exponential distribution is not always able to model the heavy tail of IATs correctly, which can be traced to the multi-modal nature of IATs, as depicted in Fig. 6. These findings support our initial assumption, that there is no common distribution that can be used to generally model the initial delay or the IATs for all packet sizes. Apart from that, our model achieves acceptable results for small to mid-sized packets; the characteristics of the measured end-to-end delay are captured by our model quite well. For $l \leq 768$ P_{95} the absolute error (E_{abs}) is in the range of ca. 0.9 ms to 5 ms which relates to a relative error E_{rel} in the range of 1.2% to 7.3%. For $l \leq 768$ P_{99} the absolute error is in the range of ca. 1.2 ms to 5 ms which relates to a relative error in the range of 1.1% to 5.9%. Another promising observation is, that the relative error of the 99th percentiles is lower than the relative error of the 95th percentiles.



(a) l=128 B (b) l=256 B

(c) l=768 B (d) l=1024 B

Fig. 11: Q-Q plot comparing the distribution of the estimated packet train end-to-end delays of the model with the empirical data for a packet train size of 40 packets and different packet lengths l



(a) l=64 B (b) l=128 B

(c) l=256 B (d) l=512 B

(e) l=768 B (f) l=1024 B

(g) l=1280 B

Fig. 12: Cumulative distribution functions of the results of our model (incl. submodels) and of the measured data for a packet train size of 40 packets and different packet lengths l

4 Applicability Considerations

The proposed model enables to estimate the delay of packet trains of a fixed length, but our prototype implementation performs quite slowly due to the combinatoric complexity of the calibration process; it can take hours to find good fittings for the individual distributions (e.g., regarding number of parameters or the complexity of the implementation of the distribution), especially in case of fitting and ranking with high spatial resolution. A detailed performance analysis is out of the scope of this work, but, even overoptimistic approximations suggest that our fitting methodology can only be applied for off-line evaluation.

For example, let us assume there is no initial knowledge about the true distributions of the IAT in packet trains of size 40, and there are 10 possible distribution types that have to be considered for fitting. Then, for a single packet train flow with a specific packet l already 390 fitting processes are required to calibrate the complete $\mathcal{T}$ sub-model. At an optimistic estimated execution time of 0.1 s per fitting, the total calibration occupies a CPU core for more than 39 s. However, real-time systems should respond on path changes nearly immediately which requires that estimations are available at high frequency. Hence, we will propose options to improve the applicability of the model to cope with this requirement. We consider the two crucial steps of the model w.r.t. its on-line application: (I) the process of fitting and (II) the process of convolution.

Regarding (I): An effective approach to decrease the time which is required for calibration, is to reduce the number of considered distributions to a minimum. This is exactly what we did when we fitted the distributions for the $\mathcal{T}$ sub-model. We started with no knowledge about the adequate distributions. Then, we found that there are some few predominant distributions and we manually filter out insufficient distributions (this could happen in an off-line study). In the end, we considered only one distribution to calibrate the $\mathcal{T}$ sub-model.

Another important option is to check whether the τ_i are independent and identically distributed (iid). To test whether the individual τ_i can be modeled as independent and identically exponentially distributed (iied), we can exploit that the convolution of n Exponential distributions with parameter λ results in a Gamma distribution with parameters n and λ. If we can find a fitting for the Gamma distribution for a constant n that acceptably models $\mathcal{T}$, then the τ_i might be iied. This

test is fast and it could be used in on-line calibration.

$$\sum_{i=1}^{n} Exp(\lambda) \sim \gamma(n, \lambda), \lambda > 0, n \in \mathbb{N}^{+} \tag{7}$$

Generally, if we find an iid property, then the calibration of the $\mathcal{T}$ sub-model is independent from the packet train size n, and we can expect a speedup of factor ca. $n - 2$ if we consider a constant amount of samples for fitting. However, the more important reason to exploit the iid property (if applicable) is to enable the model to predict $\hat{\Delta}_i$ for $i > n$.

Regarding (II): Monte Carlo simulation and numerical integration are essential tools to perform convolution, but their computational runtime is rather poor compared to heuristics, and analytical solutions for special cases. Thus, if available, we recommend to exploit special solutions to speed up convolutions, e.g., the central limit theorem [22] or solutions for Log-Normal sums [20]. The stroke of luck is to find that both, Δ and τ_i, are normally distributed, because the following analytical solution is applicable for all practical purposes:

$$\sum_{i=1}^{n} Normal(\mu_i, \sigma_i^2) \sim Normal\left(\sum_{i=1}^{n} \mu_i, \sum_{i=1}^{n} \sigma_i^2\right), \sigma^2 > 0, -\infty < \mu_i < \infty \tag{8}$$

Another promising numerical approach with low complexity is the convolution of the time discrete representation with a finite number of b steps; via the supported number of steps it is possible to control the computational steps to perform the convolution. The discrete time convolution can also be accelerated by using the Fast Fourier Transformation to convert the data from time to frequency domain where convolution is the multiplication of frequencies which means the complexity to calculate the convolution is proportional to b .

Exploiting IID To evaluate the application of the iid property in our scenario, we conduct a small benchmark to compare the execution time of the fitting of $\mathcal{T}$ (for $n = 40$) and the quality of our model with iid assumed and without. As stated above, we consider a constant amount of samples for fitting. Thus, the data provided to fit τ_{iid} was a randomly sampled fraction ($1/39$) of the data for IAT positions between 2 and 40.

Table 3 shows the execution times required for fitting $\mathcal{T}$ and the relative errors of P_{95} for the cases where we assumed iid and where we did not. We observe the promising results the model with iid assumption achieved. As expected, the mean

execution time decreases by a large factor of $\frac{8.58}{0.21} \approx 40.83$. We also observe, that the quality of the estimations of this model is similar to that without iid.

l [B]	exec. time (iid) [s]	exec. time [s]	E_{rel} of P_{95} (iid)	E_{rel} of P_{95}
64	0.23	8.11	0.048	0.073
128	0.21	8.71	0.031	0.062
256	0.21	9.09	0.044	0.012
512	0.21	9.68	0.074	0.059
768	0.21	8.64	0.059	0.049
1024	0.19	8.27	0.233	0.247
1280	0.21	7.53	0.108	0.107
mean	0.21	8.58	0.085	0.087

Table 3: Execution time vs. quality on calibrating $\mathcal{T}$ by exploiting iid property

Now, we want to give a quick outlook on the quality of predictions made by our special model that assumes iid property. For this purpose, we still use the calibration of $\mathcal{T}$ for $n = 40$, but this time we predict $\hat{\Delta}_{50}$ and compare the estimations with the measured data. The results are shown in Table 4. We omitted the Q-Q plots due to lack of space, but the plots are similar to those presented in Figs. 11a to 11d. Here too, the specialized model achieves acceptable results. E_{rel} is below 11% for P_{95} and P_{99} for all packet lengths except 1024 B.

pkt. length [B]	Data p_{95} [ms]	Model p_{95} [ms]	E_{abs} [ms]	E_{rel}	Data p_{99} [ms]	Model p_{99} [ms]	E_{abs} [ms]	E_{rel}
64	71.944	73.321	1.377	0.019	94.354	85.247	9.107	0.015
128	73.397	70.247	3.149	0.043	82.778	77.998	4.779	0.038
256	73.457	74.142	0.684	0.009	83.224	80.171	3.053	0.008
512	85.303	78.205	7.098	0.083	101.680	85.571	16.109	0.070
768	97.077	90.918	6.159	0.063	111.610	98.807	12.803	0.055
1024	134.082	91.560	42.522	0.317	170.704	99.081	71.623	0.249
1280	111.979	100.324	11.655	0.104	135.238	108.181	27.058	0.086

Table 4: Comparison of 95th and 99th percentiles estimated by the special model and obtained from measured data for packet train size of $n = 50$

5 Conclusion and Future Work

In this work, we presented a uniform model to estimate the distribution of end-to-end delays of packet trains. A testbed setup has been presented which is designed to measure delays with very high accuracy. In a case-study we used this testbed to conduct extensive measurements which we exploited to calibrate and to assess our model. The evaluation shows that a calibrated uniform model is able to mostly estimate relevant percentiles with acceptable quality. Additionally, we

found evidences that the distribution of IATs can be of multi-modal nature and that iid property (w.r.t. packet position) is generally not fulfilled. We also shared some ideas on how specialized models can better meet the requirements of real-time systems w.r.t. on-line end-to-end delay estimation and prediction. Finally, we implemented some of the proposed options and showed within a small case-study that exploiting the iid property has major impact on the runtime performance, enables to predict end-to-end delays for larger packet trains than used for the calibration, and may, incidentally, lead to better outcomes.

Future work might focus on improving the automatic calibration process of the $\mathcal{T}$ sub-model. In particular, we endorse the development of more reliable ranking metrics, more robust fitting methods, and a methodology to detect and fit multi-modal distributions. A very promising approach for this task which we will take into account in our future work is described by Markovitch and Krieger [19]. In their work they present a highly flexible technique to model PDFs by a the combination of a non-parametric "body" model and a parametric "tail" model and they also provide a calibration approach to estimate the density of the "body" via structural risk minimization as well as an approach to estimate. Their work especially features the modeling of multi-modal PDF "bodies" while it also enables to accurately model high quantiles.

We also plan detailed studies on the pros and cons of different density estimation techniques and methods to perform convolution, as well as the trade-off between amount of samples, runtime performance, and quality to gain meaningful insights that help to implement an efficient derivate of our model in order to improve specific real-time applications to cope with end-to-end delay.

Acknowledgement The author would like to thank Prof. Dr. Bernd E. Wolfinger for his support, valuable comments, enriching discussion and last but not least for careful proof-reading. A further word of thanks goes to the reviewers of this paper who provided very useful and sound advises, possible ways for future work as well as references to important related work that was previously unknown to the author.

References

1. Najah Abu Ali, Essa Ekram, Ali Eljasmy, and Khaled Shuaib. Measured delay distribution in a wireless mesh network test-bed. In *IEEE/ACS International Conference*

on Computer Systems and Applications (AICCSA), pages 236–240. IEEE, 2008.

2. John Baillieul and Panos J Antsaklis. Control and communication challenges in networked real-time systems. *Proceedings of the IEEE*, 95(1):9–28, 2007.
3. Alexander Beifuß, Torsten M Runge, Daniel Raumer, Paul Emmerich, Bernd E Wolfinger, and Georg Carle. Building a low latency linux software router. In *International Teletraffic Congress (ITC)*, volume 1, pages 35–43. IEEE, 2016.
4. Igor R Belousov, Ryad Chellali, and Gordon J Clapworthy. Virtual reality tools for internet robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1878–1883. IEEE, 2001.
5. Nabhendra Bisnik and Alhussein A Abouzeid. Queuing network models for delay analysis of multihop wireless ad hoc networks. *Ad Hoc Networks*, 7(1):79–97, 2009.
6. Almut Burchard, Jörg Liebeherr, and Stephen D Patek. A min-plus calculus for end-to-end statistical service guarantees. *IEEE Transactions on Information Theory*, 52(9):4105–4114, 2006.
7. Visual Networking Index Cisco. The zettabyte era—trends and analysis, June 2016. `http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/white-paper-listing.html`.
8. Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, 2006.
9. Claude Evequoz and Carl Tropper. End-to-end delay of multiple packet messages in window flow controlled packet switching networks. In *Proceedings on IEEE Computer and Communication (INFOCOM)*, pages 47–54. IEEE, 1990.
10. Andrew Farmer, Oliver Gibson, Paul Hayton, Kathryn Bryden, Christina Dudley, Andrew Neil, and Lionel Tarassenko. A real-time, mobile phone-based telemedicine system to support young adults with type 1 diabetes. *Journal of Innovation in Health Informatics*, 13(3):171–177, 2005.
11. Emad Felemban, C-G Lee, Eylem Ekici, Ryan Boder, and Serdar Vural. Probabilistic qos guarantee in reliability and timeliness domains in wireless sensor networks. In *Proceedings on IEEE Computer and Communication (INFOCOM)*, volume 4, pages 2646–2657. IEEE, 2005.
12. Domenico Ferrari. Rfc1193: Client requirements for real-time communication services. 1990.
13. Markus Fidler and Amr Rizk. A guide to the stochastic network calculus. *IEEE Communications Surveys & Tutorials*, 17(1):92–105, 2015.
14. Ana Hernandez and Eduardo Magaña. One-way delay measurement and characterization. In *Third International Conference on Networking and Services (ICNS)*, pages 114–114. IEEE, 2007.
15. José-Alberto Hernández and Iain W Phillips. Weibull mixture model to characterise end-to-end internet delay at coarse time-scales. *IEE Proceedings-Communications*, 153(2):295–304, 2006.
16. Mark Kalman, Steinbach Eckehard, and Bernd Girod. Adaptive playout for real-time media streaming. In *International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages I–I. IEEE, 2002.

17. Ulrich Krengel. Wahrscheinlichkeitstheorie. In *Ein Jahrhundert Mathematik 1890–1990*, pages 457–489. Springer, 1990.
18. Hong Li and Lorne Mason. Estimation and simulation of network delay traces for voip in service overlay network. In *International Symposium on Signals, Systems and Electronics (ISSSE)*, pages 423–425. IEEE, 2007.
19. Natalia M Markovitch and Udo R Krieger. The estimation of heavy-tailed probability density functions, their mixtures and quantiles. *Computer Networks*, 40(3):459–474, 2002.
20. Neelesh B Mehta, Jingxian Wu, Andreas F Molisch, and Jin Zhang. Approximating a sum of random variables with a lognormal. *IEEE Transactions on Wireless Communications*, 6(7), 2007.
21. Konstantina Papagiannaki, Sue Moon, Chuck Fraleigh, Patrick Thiran, Fouad Tobagi, and Christophe Diot. Analysis of measured single-hop delay from an operational backbone network. In *Proceedings on IEEE Computer and Communication (INFOCOM)*, volume 2, pages 535–544. IEEE, 2002.
22. Valentin Petrov. *Sums of independent random variables*, volume 82. Springer Science & Business Media, 2012.
23. Izhak Rubin. Message path delays in packet-switching communication networks. volume 32, pages 186–192. IEEE, 1975.
24. Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.
25. Susumu Tachi. Real-time remote robotics-toward networked telexistence. *IEEE Computer graphics and applications*, 18(6):6–9, 1998.
26. Ronald L Wasserstein and Nicole A Lazar. The asa's statement on p-values: context, process, and purpose, 2016.
27. Bernd E Wolfinger. Delay approximations for file transfers in computer networks. In *Proceedings on IEEE Computer and Communication (INFOCOM)*, pages 609–615, 1986.
28. Dapeng Wu, Yiwei Thoms Hou, and Ya-Qin Zhang. Transporting real-time video over the internet: Challenges and approaches. *Proceedings of the IEEE*, 88(12):1855–1877, 2000.
29. Jiyan Wu, Chau Yuen, Bo Cheng, Yanlei Shang, and Junliang Chen. Goodput-aware load distribution for real-time traffic over multipath networks. *IEEE Transactions on Parallel and Distributed Systems*, 26(8):2286–2299, 2015.
30. Wei Zhang and Jingsha He. Modeling end-to-end delay using pareto distribution. In *Second International Conference on Internet Monitoring and Protection (ICIMP)*, pages 21–21. IEEE, 2007.

New Network and Application Architectures

Combining Satellite and Terrestrial Communication Systems: Introduction of the Transparent Multichannel IPv6 (TMC-IPv6) Project

Jörg Deutschmann, Kai-Steffen Hielscher, Reinhard German

Computer Networks and Communication Systems
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
{joerg.deutschmann, kai-steffen.hielscher, reinhard.german}@fau.de
www7.cs.fau.de

Abstract. The combination of multiple communication systems, also known as bonding, is a recent approach to improve the overall network performance. Especially in rural areas with limited access to broadband communication networks the utilization of a terrestrial interface with low data rate/low latency and a satellite interface with high data rate/high latency simultaneously can be highly beneficial. One solution already deployed is Multipath TCP (MPTCP), which is designed to work at the transport layer and is limited to TCP. For many cases, like UDP-based VoIP or IoT protocols, this is insufficient. We therefore introduce the Transparent Multichannel IPv6 (TMC-IPv6) project, which aims to combine the advantages of heterogeneous communication systems at the network layer. For now, we describe the problem statement and involved communication entities as well as first ideas on possible solutions.

Key words: Multichannel; Multipath; Bonding; Hybrid Access; Satellite Communication; Terrestrial and Satellite Networks; Hybrid Networks

1 Introduction

Satellite communication is a way to provide broadband internet access all over the world. However, with geostationary satellites the propagation delay leads to very high delays in the magnitude of several hundred milliseconds. In order to improve

the interactivity and responsiveness of communication systems, utilizing a second communication link can be highly beneficial.

The Transparent Multichannel IPv6 (TMC-IPv6) Project aims to combine the advantages of multiple heterogeneous communication links. An illustrative example is the combination of a DSL connection with low data rate/low latency and a satellite connection with high data rate but high latency, which results in a user's internet access with high data rate and low latency providing a better Quality of Experience (QoE). An abstract view of the communication model is shown in Fig. 1, whereby the left side may be a single private home user up to a small network.

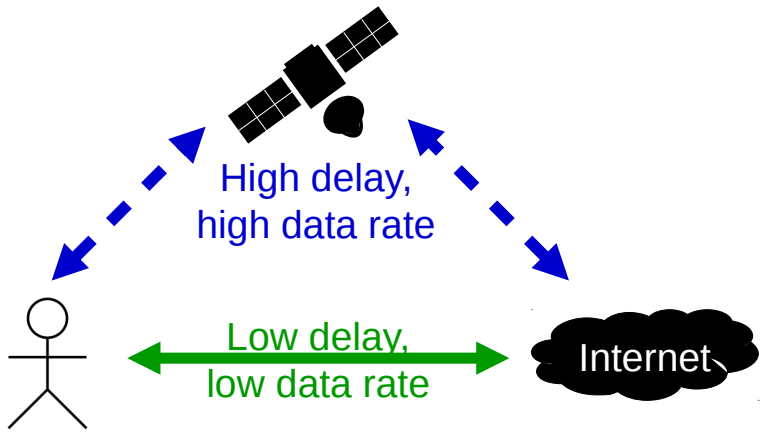


Fig. 1. TMC-IPv6 abstract communication model

In packet switched networks, an IP packet may be transported over arbitrary paths. The problems arise at the transport layer: When packets of a single TCP flow are distributed among links with different delays, out-of-order segments will be interpreted as congestion, trigger retransmissions and thus result in a poor performance [1].

As part of the introduction, we shortly introduce satellite communication networks, terminology and traffic granularity. Section 2 reviews various multipath concepts at different layers. This is followed by a detailed description of the TMC-IPv6 project in Section 3. Section 4 summarizes this paper.

1.1 Satellite Communication

Protocol optimization for satellite communication is a wide area of research. A satellite link has a large bandwidth-delay product (BDP). There are several TCP variants [2] optimized for networks with large BDP (also known as "long fat networks"). Moreover, commercial satellite communication systems deploy performance enhancement proxies (PEP) to mitigate poor protocol performance caused by high delays. PEPs usually work on the transport layer by splitting TCP connections and/or TCP ACK manipulations, see e.g. [3]. However, PEPs can not work on network layer encrypted packets (e.g. IPSec).

In the field of satellite communications a hot topic are "Megaconstellations", i.e., a large number of small and cheap satellites (up to several thousands) in low earth orbit (LEO) to provide worldwide internet access. With LEO communication satellites the propagation delay is significantly lower compared to geostationary earth orbit (GEO) satellites. Thus a LEO satellite link is more comparable to the characteristics of a terrestrial communication link. Nevertheless, the combination of LEO and GEO communication satellites is as interesting as the combination of terrestrial systems and GEO satellites.

1.2 Terminology

The terms multipath, bonding and hybrid access are most common regarding the *simultaneous* use of multiple communication links.

The first one is well-known from Multipath TCP (see Section 2.1), however the term multipath is also used in a more general context (e.g. multipath routing).

Bonding is the most favorable term in our opinion as it specifically means the combination of multiple network interfaces, especially wide area network (WAN) links.

Hybrid access is a term for example used by Telekom's proprietary MagentaHybrid solution. One should not confuse it with hybrid networks, which usually describes networks made up of different technologies or topologies. For literature describing hybrid networks, including mechanisms for handover between terrestrial and satellite networks, see [4].

In this document, we do not consider multihoming and mobility as these concepts do usually not describe the simultaneous use of wide area access networks. Nevertheless, some of these concepts might provide mechanisms relevant for the

TMC-IPv6 project. For example, Mobile IPv4/6 accesses multiple communication channels but in a sequential way.

1.3 Protocols, Flows, Packets

The desired granularity of traffic distribution among different links will have an impact on the feasability and complexity of a potential solution. A simple example of splitting traffic based on the protocol would be sending time-critical data (e.g. VoIP data in UDP packets) via terrestrial DSL access and non-realtime traffic (e.g. any TCP-based file transfer protocol) via satellite. Distributing multiple TCP flows among different interfaces raises the question which flow should go over which link. Widely used load-sharing techniques also distribute TCP flows among different interfaces but assume identical link characteristics. Ultimately, splitting a single TCP flow over links with different characteristics will cause problems due to TCP's flow control.

2 Related Work

There are different multipath approaches at different layers. In the following, we present a survey of related approaches relevant for the TMC-IPv6 project, starting from the transport layer down to the data link layer. As the TMC-IPv6 project aims at a solution at the network layer, we will review related work at this layer most thoroughly.

We would like to point out that there is a interesting work [5] which exploits access network heterogeneity in the context of named data networking (also known as content centric networking). We have not examined such next-generation networking concepts yet.

2.1 Layer 4 Multipath TCP

A recent and already deployed multipath strategy is Multipath TCP (MPTCP) [6], which has also been evaluated for different path characteristics [7]. MPTCP works on the transport layer and creates separate TCP subflows among different links between hosts. Intermediate networks do not have to be aware of MPTCP, for them a MPTCP subflow is like any other TCP/IP connection. See Fig. 2 for the MPTCP design and protocol stack. As one subflow usually contains only some parts of the

overall TCP session, this may impact the correct function of middleboxes as well as middleboxes may affect the correct function of MPTCP, as described in [8]. This might be relevant for satellite backbone networks, where middleboxes like PEPs are commonly used. There is the effect that PEPs for accelerating web traffic will not work properly, or even worse, a badly implemented PEP might make MPTCP unusable [9].

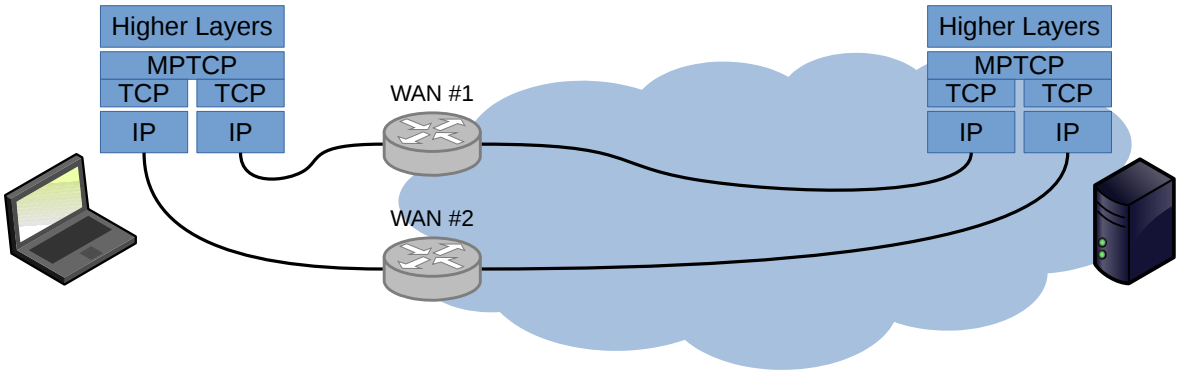


Fig. 2. MPTCP design and protocol stack

In its original form, MPTCP is implemented only on the end points of a connection. This is sufficient for common use cases, e.g. when a mobile phone has simultaneous access to wireless LAN and a cellular network like LTE. Besides aggregated datarates and higher connectivity, MPTCP also provides better mobility in such scenarios. Approaches to provide MPTCP as proxy or middlebox in order to support non-MPTCP capable end-hosts are described in [10,11,12,13,14,15]. Of special interest is [16], which also considers satellite communication.

The major drawback of MPTCP is the limitation to TCP traffic. In the TMC-IPv6 project, any protocol from Layer 3 upwards shall be supported. This includes for example UDP, which is very important for time-critical protocols and applications like Voice over IP (VoIP), Real-time Transport Protocol (RTP) or Internet of Things (IoT) protocols like Constrainted Application Protocol (CoAP).

2.2 Layer 3 Bonding Strategies

Load-balancing traffic within a network can be achieved with Equal Cost Multipath routing (ECMP). By creating hashes of packet headers, a per-flow traffic distribution is realized. We also note source routing, which is disgraced due to security

issues [17].

A different strategy at layer 3 utilizing multiple WAN access links is referred to as “Bonding” or “Hybrid access”. We list relevant approaches in chronological order. All of them are based on an architecture shown in Fig. 3, with a gateway at the customer side and a gateway in the backbone/provider network. Between the two gateways multiple connection links are utilized. Packets sent from the customer to the internet (“uplink” or “return link”) are distributed at the customer gateway among multiple communication links, recombined/reordered at the backbone gateway and then forwarded to the destination. In the reverse direction (“downlink” or “forward link”) this happens conversely.

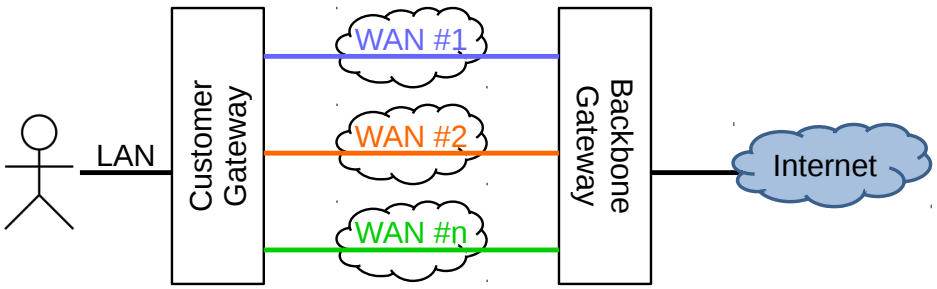


Fig. 3. Architecture for Layer 3 bonding strategies presented in Section 2.2

The reordering solves the problem of out-of-order TCP segments but it is also a major drawback as buffers are required and additional delay is introduced. When terrestrial communication links (e.g. DSL and LTE, or WiFi and LTE) with similar characteristics are combined the drawbacks are less severe. With vastly varying link delays, especially high delay satellite links, deferring a packet several hundreds of milliseconds seems not acceptable for most use cases. We emphasize that the following approaches were designed without having satellite communication in their focus.

Proprietary solution: Viprinet. Viprinet is a longstanding commercially available solution for WAN bonding [18]. The gateway at the customer side is called “Multichannel VPN Router” and the device in the backbone network is called “Multichannel VPN Hub”. The connections between both gateways are virtual private network

(VPN) tunnels, and the gateways itself are custom-tailored hardware. Moreover, TCP performance optimizations are integrated into the gateways, similar to that of a PEP. However, it is unclear how data is split among the different WAN links and we are not aware of any scientifically proven performance evaluation of the Viprinet system. Some rather inaccurate performance measurements are available online [19].

Proprietary solution: MagentaHybrid. Telekom's MagentaHybrid is a bonding solution combining DSL and LTE access networks, some technical details of this solution are presented in [20]. The customer gateway is called "Home Gateway", and the backbone/provider gateway is called "Hybrid Access Aggregation Point". In [21] some performance evaluations based on high-level simulations, without considering specific protocol implementations, are presented. Time-sensitive traffic, e.g. VoIP, is always sent via DSL, else MagentaHybrid follows a "cheapest link first" approach where LTE is only used if the capacity of the DSL link is exhausted. Furthermore, we note that a press release states that a transition to MPTCP is planned [22]. Currently Telekom is also involved in the IETF working group described in the following.

Research and Standardization Activities. In [23], a Layer 3 multipath strategy is presented, which the authors call "multipath bonding". The authors describe their approach as a generalization of MPTCP proxies to Layer 3. Their focus is the aggregation of DSL and cellular networks, there is a "Customer Bonding Gateway" and a "Provider Bonding Gateway". They introduce a protocol which takes care of the reordering and present evaluations from a testbed. At the same time, a technical report from the Broadband Forum [24] describing hybrid access techniques was published.

Finally, there are several IETF activities related to hybrid access [25,26,27], bandwidth aggregation [28,29] and tunnel bonding [30], where persons of the previously mentioned work are involved. An important outcome of these documents is the formation of the bandwidth aggregation for internet access (BANANA) working group[1]. In Fig. 4 the Generic Routing Encapsulation (GRE) tunnel bonding solution as desribed in [30] is visualized.

[1] `datatracker.ietf.org/group/banana`

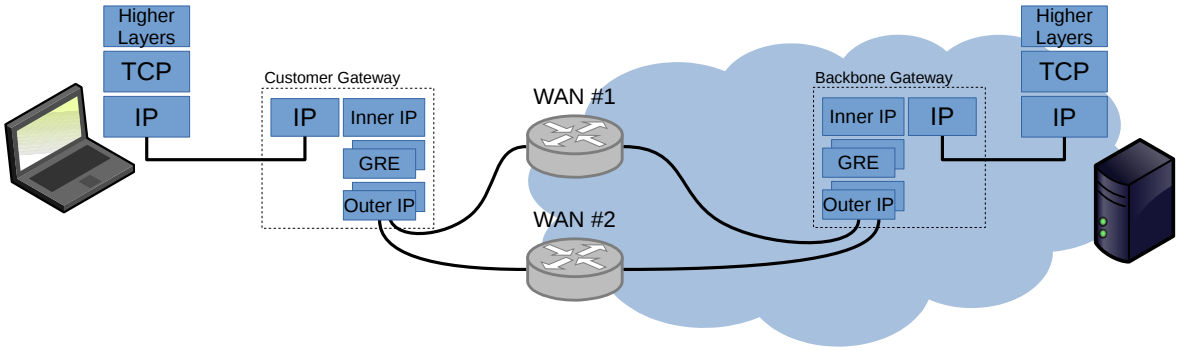


Fig. 4. GRE Tunnel Bonding as described in RFC8157 [30]

2.3 Layer 2: MLPPP and LACP

One multipath solution designed for the data link layer is the Multilink PPP protocol (MLPPP) [31], it is also one of the oldest standardized solutions we found. It was mostly used to bundle multiple telephone lines. Another solution is the Link Aggregation Control Protocol (LACP) which is typically used to bundle multiple ethernet links. Both protocols usually assume similar link characteristics. As the WAN internet accesses in the TMC-IPv6 project are assumed to be IP-based, Layer 2 solutions are currently out of scope.

3 The Transparent Multichannel IPv6 Project

The TMC-IPv6 project aims to combine the advantages of multiple communication links. Although a final solution should not be restricted to any specific kinds of wide area access networks, satellite links with their high delay and high BDP are of special interest. Customer premises equipment shall not be changed, and neither the satellite backbone network nor the terrestrial backbone network can be altered (backbone networks may contain authentication, quality of service, or middleboxes like PEPs, etc.). Fig. 5 illustrates these requirements.

3.1 Project Goals

- Bonding of multiple heterogeneous wide area access networks.
- Network layer based solution. This allows transparent communication for any higher layer protocols. It is also motivated by the fact that hosts and/or networks at the customer site are already connected via Layer 3 VPNs (e.g. IPSec) und thus higher layer headers are not visible.

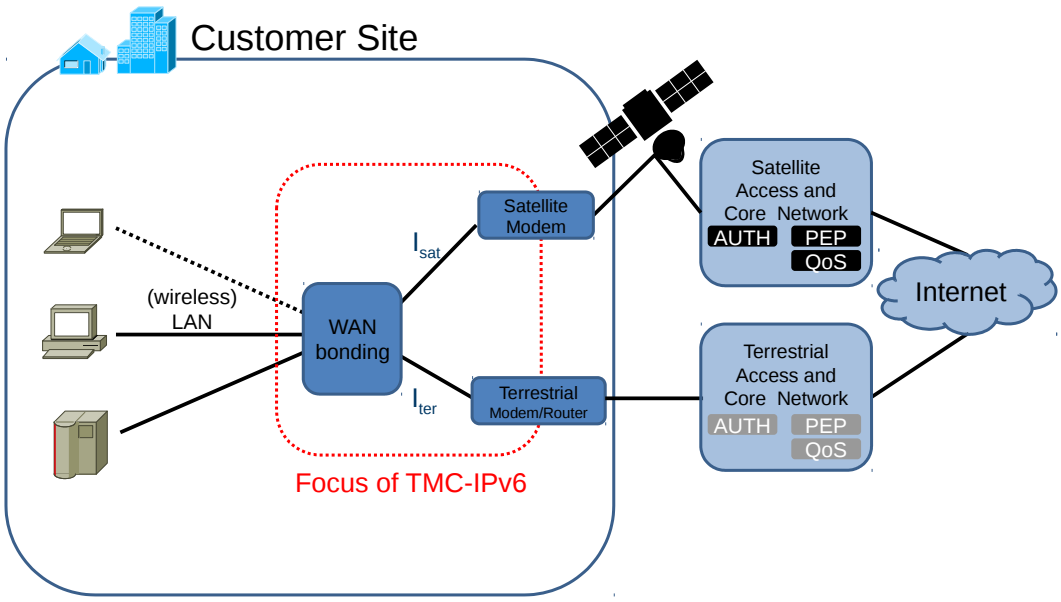


Fig. 5. TMC-IPv6 project (I_{sat}: Satellite access link, I_{ter}: Terrestrial access link)

- Increased reliability. In case a WAN link becomes unavailable, the remaining WAN links shall take over without further user interaction. The multipath approaches presented before inherently provide this feature by always maintaining paths distinct from each other. Yet, protocols especially designed for failure handling, like Site Multihoming by IPv6 Intermediation (SHIM6), might be relevant in this context.
- Quality of Service. All approaches shall be evaluated with Quality of Service (QoS) metrics.
- Simulation, Emulation and Testbed deployments. By use of simulation models, the overall system shall be evaluated from the physical layer up to selected applications. Additionally, a reference implementation is required for publishing a final solution as IETF Request for Comments (RFC). Advances in network simulation software like ns-3 Direct Code Execution, which allows the integration of existing protocol implementations into a network simulator, might become handy in this context.

3.2 Individual Analysis of Involved Communication Links

Before combining different communication links, we do some analysis on a typical DSL link (1 Mbps, round trip time: 60 ms) as well as a typical satellite link (10 Mbps, round trip time: 600 ms). Both links are symmetric. The transmission dura-

tion of a single object can be estimated according to simplified TCP performance calculations shown in equation 1 (taken from [32]), which considers TCP slow start with an initial window size of one, three-way handshake, round trip time, data rate and maximum segment size. Not considered are packet errors, congestion, processing delays, header sizes and buffer limitations.
Given object size O, data rate R, and segment size $S = 1500$ byte the transmission duration of a single object is

$$\text{Transmission duration} = 2 \cdot RTT + \frac{O}{R} + P\left[RTT + \frac{S}{R}\right] - (2^P - 1)\frac{S}{R} \tag{1}$$

where P depends on two cases and is calculated as follows:

$$K = \left\lceil \log_2\left(\frac{O}{S} + 1\right)\right\rceil \qquad \text{(number of windows that cover the object)}$$

$$Q = \left\lfloor \log_2\left(1 + \frac{RTT}{S/R}\right)\right\rfloor + 1 \qquad \text{(number of windows with idle times assuming an infinitely large object)}$$

$$P = \min\{Q, K - 1\}$$

Figure 6 shows that for small object sizes the DSL link ($R = 1$ Mbps, $RTT = 60$ ms) is faster because of the smaller round trip time (RTT). For larger object sizes, the transmission duration is dominated by the data rate, therefore the satellite link ($R = 10$ Mbps, $RTT = 600$ ms) is superior. The absolute values and the break-even point at 0.8 MByte should be considered with caution due to the simplifications described before, in real implementations the values might vary. Yet it visualizes the statement that small objects (e.g. connection setup) should use the DSL link and large objects (e.g. file transfer) should use the satellite connection. More accurate analysis considering different TCP parameters and/or variants are to follow, and more realistic internet traffic patterns would also be helpful.

3.3 Traffic Models

Besides bulk data transfer and constant bite rate traffic, it will be important to consider realistic web traffic models. A survey of traffic generators for internet traffic and which one to use will be carried out during the project.

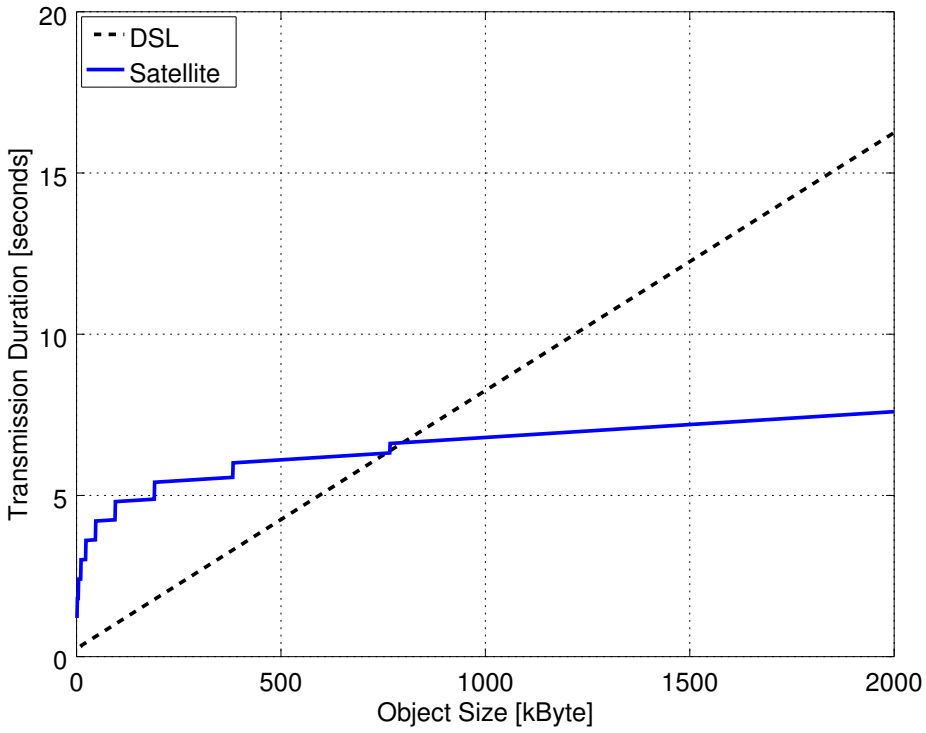


Fig. 6. Comparison of a typical satellite link (10 Mbps, RTT: 600 ms) and a slow DSL link (1 Mbps, RTT: 60 ms)

3.4 First Ideas

In the following, some preliminary ideas regarding network layer bonding of multiple WAN interfaces are presented:

- Performance of other transport layer protocols. As mentioned before, arbitrarily distributing packets among different WAN interfaces will cause TCP out-of-order segments and lead to poor TCP performance. However, recent transport layer protocols like Stream Control Transmission Protocol (SCTP) or Quick UDP Internet Connections (QUIC) are designed with multihoming in mind and might therefore be less sensitive to packet reordering. There is ongoing research for Concurrent Multipath Transfer for SCTP (CMT-SCTP), and the QUIC working group at the IETF has multipath in its charter, too.
- Distribute traffic depending on protocols: Send UDP via terrestrial link and use MPTCP for bonded links. This is a combination of the approaches shown in Section 2. However, it requires knowledge of the header following the IP header, which might not be available as stated in the project goals. Nevertheless such a setup is interesting as a performance benchmark.

– Distribute packets depending on size and/or statistical measures. Without having knowledge of individual flows, traffic classification based on heavy-tailed distributions and the self-similarity of internet traffic may help to decide which packet should go over which link.

4 Conclusion

We presented the TMC-IPv6 project and some of the challenges that are upcoming. In the related work section we saw that multiple wide area access networks can be utilized simultaneously at different layers. Layer 3 solutions involve middleboxes between the bonded WAN links: In order to avoid TCP performance degradation, packets sent among links with different delays are reordered to their original sequence when leaving the bonding. Such a reordering seems unfeasible for satellite links. Generally, satellite links are often out of scope when multipath bonding is discussed.

Acknowledgement This work is funded by the German Ministry for Economic Affairs and Energy (BMWi) on the basis of a decision by the German Bundestag.

References

1. E. Blanton and M. Allman, "On Making TCP More Robust to Packet Reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 32, pp. 20–30, Jan. 2002.
2. A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-Host Congestion Control for TCP," *IEEE Communications Surveys Tutorials*, vol. 12, pp. 304–342, Third 2010.
3. J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, June 2001.
4. P. Berthou, C. Baudoin, T. Gayraud, and M. Gineste, *Satellite and Terrestrial Hybrid Networks*. John Wiley & Sons, Inc., 2015.
5. K. M. Schneider and U. R. Krieger, "Beyond network selection: Exploiting access network heterogeneity with named data networking," in *Proceedings of the 2Nd ACM Conference on Information-Centric Networking*, ACM-ICN '15, 2015.
6. A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, January 2013.
7. B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo, "Impact of Path Characteristics and Scheduling Policies on MPTCP Performance," in *Proceedings of the 2014 28th International Conference on Advanced Information Networking and Applications Workshops*, WAINA '14, pp. 743–748, IEEE Computer Society, 2014.
8. E. Lopez, "Multipath TCP Middlebox Behavior," Internet-Draft draft-lopez-mptcp-middlebox-00, IETF Secretariat, November 2014.
9. O. Bonaventure, "Multipath TCP through a strange middlebox." `blog.multipath-tcp.org/blog/html/2015/01/30/multipath_tcp_through_a_strange_middlebox.html`, January 2015. Accessed: 2017-07-05.
10. G. Hampel and T. Klein, "MPTCP Proxies and Anchors," Internet-Draft draft-hampel-mptcp-proxies-anchors-00, IETF Secretariat, February 2012.
11. G. Detal, C. Paasch, and O. Bonaventure, "Multipath in the Middle(Box)," in *Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMiddlebox '13, pp. 1–6, ACM, 2013.
12. D. Lingli, D. Liu, T. Sun, M. Boucadair, and G. Cauchie, "Use-cases and Requirements for MPTCP Proxy in ISP Networks," Internet-Draft draft-deng-mptcp-proxy-01, IETF Secretariat, October 2014.
13. X. Wei, C. Xiong, and Ed, "MPTCP proxy mechanisms," Internet-Draft draft-wei-mptcp-proxy-mechanism-02, IETF Secretariat, June 2015.
14. B. Peirens, G. Detal, S. Barre, and O. Bonaventure, "Link bonding with transparent Multipath TCP," Internet-Draft draft-peirens-mptcp-transparent-00, IETF Secretariat, July 2016.
15. M. Boucadair, C. Jacquenet, O. Bonaventure, D. Behaghel, S. Secci, W. Henderickx, R. Skog, S. Vinapamula, S. Seo, W. Cloetens, U. Meyer, L. Contreras, and B. Peirens, "Extensions for Network-Assisted MPTCP Deployment Models," Internet-Draft draft-boucadair-mptcp-plain-mode-10, IETF Secretariat, March 2017.
16. L. Ottavj, E. Duros, J. Webert, F. Gamberini, C. Niephaus, J. Perez-Trufero, and S. Watts, *Intelligent Gateways Enabling Broadband Access via Integrated Terrestrial and*

Satellite Systems, pp. 92–102. Springer International Publishing, 2015.
17. J. Edge, "IPv6 source routing: history repeats itself." `lwn.net/Articles/232781`, 2007. Accessed: 2017-07-05.
18. "Anordnung zum Übermitteln eines Datenstroms über gebündelte Netzwerkzugangsleitungen, sowie Sende- und Empfangshilfsvorrichtung dafür," 2013. DE Patent 202,008,018,396.
19. "Testbericht: Viprinet Multichannel-Router VPN 300 Hybrid Internet für Unternehmen im Praxistest." `www.lte-anbieter.info/lte-hardware/tests/15/viprinet-vpn300.php`. Accessed: 2017-07-05.
20. N. Leymann, "Hybrid Access Deployment @ DT." `https://www.ietf.org/proceedings/97/slides/slides-97-banana-hybrid-access-deployment-at-deutsche-telekom-01.pdf`, November 2016. BANANA BoF, IETF97.
21. O. D. Ramos-Cantor, M. Lossow, H. Droste, G. Kadel, and M. Pesavento, "A Network Simulation Tool for User Traffic Modeling and Quality of Experience Analysis in a Hybrid Access Architecture," in *WTC 2014; World Telecommunications Congress 2014*, pp. 1–6, June 2014.
22. Telekom Innovation Laboratories, "Evolution of Hybrid Access - T-Labs is working on a stardardized solution." `http://www.laboratories.telekom.com/public/english/newsroom/news/pages/hybrid-evolution.aspx`, December 2014. Accessed: 2017-05-01.
23. M. Bednarek, G. B. Kobas, M. Kühlewind, and B. Trammell, "Multipath Bonding at Layer 3," in *Proceedings of the 2016 Applied Networking Research Workshop*, ANRW '16, pp. 7–12, ACM, 2016.
24. Broadband Forum, "Hybrid Access Broadband Network Architecture," Tech. Rep. TR-348, July 2016.
25. N. Leymann, C. Heidemann, M. Wasserman, L. Xue, and M. Zhang, "Hybrid Access Network Architecture," Internet-Draft draft-lhwxz-hybrid-access-network-architecture-02, IETF Secretariat, January 2015.
26. P. Muley, W. Henderickx, G. Liang, H. Liu, L. Cardullo, J. Newton, S. Seo, S. Draznin, and B. Patil, "Network based Bonding solution for Hybrid Access," Internet-Draft draft-muley-network-based-bonding-hybrid-access-01, IETF Secretariat, February 2017.
27. M. Menth, A. Stockmayer, and M. Schmidt, "Lisp hybrid access," Internet-Draft draft-menth-lisp-ha-00, IETF Secretariat, July 2015.
28. M. Cullen, N. Leymann, C. Heidemann, M. Boucadair, D. Hui, M. Zhang, and B. Sarikaya, "Problem Statement: Bandwidth Aggregation for Internet Access," Internet-Draft draft-zhang-banana-problem-statement-03, IETF Secretariat, October 2016.
29. M. Cullen and M. Zhang, "Considerations for Bandwidth Aggregation," Internet-Draft draft-mrc-banana-considerations-01, IETF Secretariat, July 2016.
30. N. Leymann, C. Heidemann, M. Zhang, B. Sarikaya, and M. Cullen, "Huawei's GRE Tunnel Bonding Protocol," RFC 8157, May 2017.

31. K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti, "The PPP Multilink Protocol (MP)," RFC 1990, August 1996.
32. J. F. Kurose and K. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 2002.

Evaluation of Actor Based Code Offloading from Android Smartphones in Real-Time

Marcel Großmann and Andreas Keiper

Computer Networks Group, University of Bamberg
An der Weberei 5, 96047 Bamberg, Germany
marcel.grossmann@uni-bamberg.de

Abstract. The widespread idea of cloud computing affects developing approaches of mobile applications. With ubiquitous computing capabilities and mobile devices, one question arises for mobile application developers: How could battery drainage be reduced? To combine the cloud computing services and mobile applications, an offloading of processes may produce a benefit to save battery life. For example, background processes are mainly uninteresting for mobile users and can be offloaded to the cloud, as well as image processing or searching. Only a feedback mechanism is necessary to obtain results of a process in the cloud. With already existing technologies, an approach based on actors seems to be able to handle code offloading. By avoiding the pitfalls of this concept, system designers, who identify battery and processing power consuming applications, can offload critical code to an actor in real-time. The signalling concept is analysed with existing web-service technologies and described by a messaging paradigm, which fulfills the actor's requirements. A trial architectural design demonstrates the functionality of real-time code offloading on mobile devices.

Key words: Android; Architectural Design; Actor Framework; Code Offloading; Real-Time

1 Introduction

Since a lot of computing power nowadays is provided by cloud services, code offloading may reduce battery drainage of mobile devices. Modern technologies offer a lot of platforms that are capable to ease the process of programming code

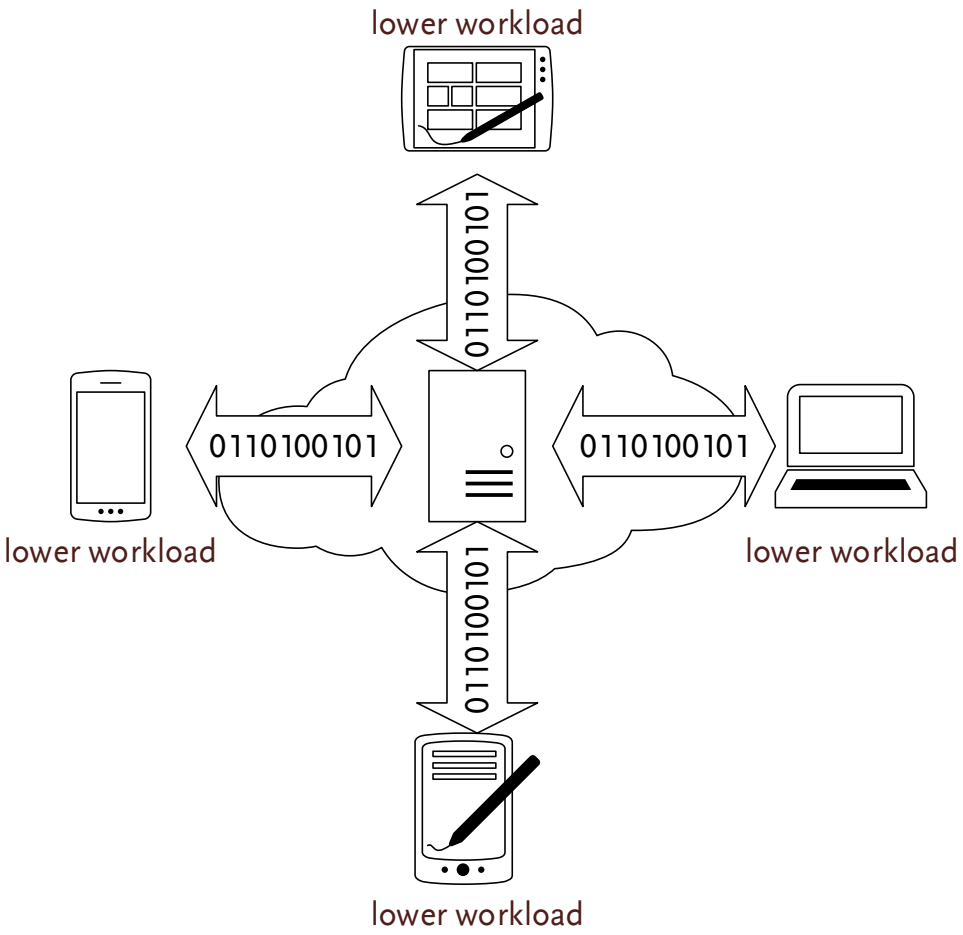


Fig. 1. The idea behind dynamic code offloading into the cloud. The server in the middle has highspeed network access, while the mobile devices around can offload user defined code onto it. This results in a lower workload of the devices.

offloading paradigms. The following approach describes, how to dynamically outsource code to be computed inside a cloud or fog architecture as depicted in Figure 1. Especially, a selection of communication protocols is analysed, if they are capable to handle the exchange of dynamic code offloading in a platform independent way. On cloud side, an actor based methodology is used, in order to provide a scalable server infrastructure. This architectural principle eliminates the problem of accessing commonly used memory; it is implemented by the open source actor framework *akka.io* [13]. The actor approach was presented by Hewitt *et al.* [6], which was adapted and implemented several times. The model is based on four principals:

- share nothing
- message transmission
- message queues
- simple scaling

Standard Java libraries for serialization are used for code transmission. Mobile classes are dynamically loaded into a Java runtime environment via a classloader approach. Therefore, the transmission technologies are evaluated to determine an approach to access the server application running on a cloud infrastructure.

1.1 Related Work

Roelof Kemp *et al.* introduced the framework *Cuckoo* [8]. With Cuckoo it is possible to offload compute intensive services by defining interfaces for them before the application is built, but not on realtime. Besides, a weakness is the concurrent usage of a single remote resource by multiple users, as the server is not solving concurrency issues by now. In *CloneCloud* [2] it is not possible to migrate native state, while this approach tries to offload classes by serializing them in realtime with their state information for execution. In our approach the execution of tasks is restricted to actor based technology that offers the ability to construct independent jobs, which can not interfere. Moreover, every job is controlled through a standardized message based methodology.

Approaches like *MAUI* [3] offload mobile code to the infrastructure, based on the whole process. In the cloud MAUI uses virtual machines, where an emulated smartphone system is executed. Thus, this results in a lack of scalability and wastes CPU resources through the virtualization of an entire mobile platform on a cloud

server. Besides, Thinkair [9] removes this burdensome emulation and runs with an Android x86 port on server side. Nevertheless, both approaches provide their scalability in the cloud by generating a new VM for each offloading device, which is a very heavyweight and less cost effective solution.

Flores *et al.* [4] propose that offloading as a service is not going to be established in those architectural proposals, because computational capabilities of the latest smartphones are comparable to some servers running in the cloud. Thus, emulation and heavyweight creation of VMs lacks the ability to scale offloading services in a cost-effective fashion.

In this paper, a finer offloading granularity is achieved by using the serialization of classes in realtime to migrate their byte representation to a cloud service. The cloud service itself is based on an actor system, where a new actor is generated for each offloading device. Moreover the actor system is integrated in a Glassfish server, which runs in a Docker container.

2 Technologies

In this section, the principles of the actor framework are described in more detail. This is followed by the question, how to perform dynamic classloading in a Java environment to be able to offload code in realtime. Existing technologies for transmission are analysed in subsection 2.4 and evaluated accordingly. Finally, the cloud infrastructure, especially the server and middleware technologies are evaluated in subsection 2.5.

2.1 The Actor Framework

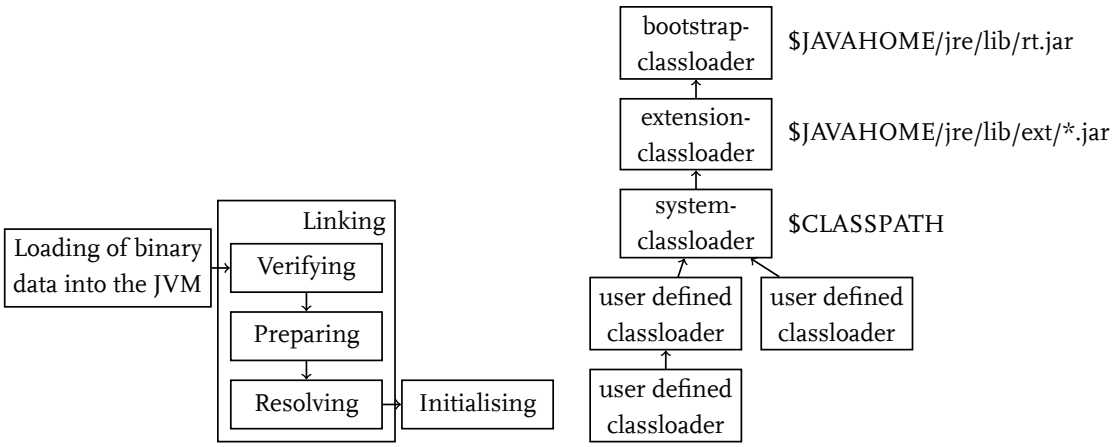
The open source actor based framework *akka* is written in Scala and targets the parallel and fault-tolerant execution of scalable applications [13]. Thus, the actor model offers scalability and concurrency, while fault tolerance is achieved by a *let it crash* model within the error handling. For the later, it is possible to define strategies on how to handle errors of actors. Akka is a distributed system, which may be extended over several *Java Virtual Machines* (JVM) and possesses an approach to realise atomic messaging as transactions. Furthermore, there are possibilities to configure the internal thread management, to manipulate the message flows and comprehensive functions to manage the actor system.

To program actors in the simplest way, a subclass of `AbstractActor` must be implemented, where the method `createReceive` is overwritten to handle received messages. With the help of the configuration class `Props`, actors can be generated from the actor class files in an actor system. Scala's integrated execution framework performs the actual execution of the actors, thus, the actor execution is solely separated from user commands.

2.2 Dynamic Classloading

While akka executes the code, one question arises, on how to transmit the code from a mobile device to the server. Investigations in akka's source code revealed that instances of `Props` are serializable, but also that a loaded actor class file is needed in the JVM for actor generation. Hence, separate dynamic reloading of class files with a self-defined classloader is required. Due to the transmission process, all of the byte code of a class and the code of individual instantiated objects must be transmitted. Therefore, the serialization is done in two steps. On the one hand, class files must be converted into byte arrays, such that a class loader can process them on the server side; this is done with the help of Java's streaming *application programming interface* (API). For serialisation and de-serialisation of instantiated objects of the memory Java offers a serialization API, which takes care of referential dependencies. To use this mechanism, the interface `Serializable` must be implemented by the class that should be serialized. It is the user's responsibility to define, if an object state is serializable. For identification purposes the `serialVersionID` is the same on both JVMs for a successful de-serialization. Nevertheless, dynamic class loading is even more complicated and must be analysed in more detail.

Classloader offer the possibility to find, load and link the binary representation of a class, such that they are usable for initialisation in the virtual machine. All classloaders inherit the abstract class `ClassLoader`. Concerning the JVM 8 specification, the point in time for classloading is differentiated either during the program start or user-defined at runtime. While the program starts up, the JVM uses its own *Bootstrap*-classloader for loading and linking of the non-array classes that are known on compile time. Arrays in Java are an internal class of the virtual machine and do not need any form of an explicit representation. Regarding the spec-



(a) . . . procedure [10]. (b) . . . hierarchy [14].

Fig. 2. Java's classloading . . .

ification this procedure including the method call is processed for a single class `Test` with the method `main` as follows:

– *Loading* of the class `Test`

The JVM tries to execute the program via the **main** method. It recognizes that there is no binary representation of the class `Test` in the memory and thus uses the classloader to find a representation in its classpath. If it succeeds, it loads this representation in the cache of the classloader, else it throws an exception. Is the classloader part of a classloader hierarchy, it delegates the loading procedure to the next higher classloader in the hierarchy. This procedure continues until no higher classloader is found anymore and the current one performs the loading. On the highest program level, this would be the system classloader. Inside the JVM two more hierarchy levels exists, the *bootstrap* and the *extension* classloader. The former is integrated into the JVM and loads the core classes from `$JAVAHOME/jre/lib/rt.jar`, while the later is responsible for extensions in `$JAVAHOME/jre/lib/ext`. The system classloader serves for loading of the program classes of the classpath of a program.

– *Linking* of `Test`

The method `main` requires an object of the class `Test`, which must be initialized first by linking the class. Linking consists of *verifying*, *preparing* and *resolving*. Verifying ensures that the code is well-formed regarding the semantics of Java and the JVM. Preparing includes the allocation of static memory and all data structures, which are internally used by the JVM, e.g., method tables. Resolving

controls the references of `Test` to other classes and interfaces and checks their correctness. Two methods are used for linking: *static* or *dynamic*. With static linking all references of a class are resolved recursively. In contrast, by dynamic linking it is done successively within the usage of a class. Java implements this dynamic approach, which allows a transmission and execution of remote code [10].

– *Initialising*
  Initialising consists of the generation of all static elements of `Test`. Even more, this means that all super classes of `Test` are also initialised. Therefore, a recursive loading and linking of the super classes must take place to build up the inheritance hierarchy in the memory. Initialization is a prerequisite, which must be fulfilled, to create object instances of a class.
– *Calling*
  After initialization of `Test` and all super classes, the method `main` can be called. Specific for Java is that `main` is declared `public`, `static` and `void`.

To influence the classloading procedure, a new classloader must be implemented, which inherits the abstract class `ClassLoader` or any other classloader implementations and serves to load classes from external sources. This is a main contribution Java has ahead of machine-oriented languages, like C or C++.

2.3 Manual Classloading

Classloaders deliver class objects, which can be instantiated. With the extension of a delegation model and resulting hierarchies, requests are delegated upwards in the hierarchy. One classloader usually delegates the loading procedure to the next upper one before it tries to load the class by itself. As a consequence, classes loaded on a higher hierarchy level can not refer to classes, which were loaded on a lower level, though it works vice versa. Threads use their own context classloader, such that all class requests inside a thread are primarily handled by this classloader. To understand the structure of the classloader itself, all important methods are described in the following:

– `Class loadClass(String name)`
  Until Java 1.2 this methods must be overwritten for an adjusted classloading procedure. With the help of the classes name with a given package structure it tries to find the class and returns a not instantiatable `Class` object.

- `protected Class defineClass(String name, byte[] b, int off, int len)`
  Method that generates a `Class` object out of a byte array by reading, controlling and converting byte code into an executable data structure.
- `protected Class findClass(String name)`
  Method used inside of `loadClass` to find the class by a given name. Adjustments of the classloader should be done by overwriting this method.
- `protected Class findLoadedClass(String name)`
  Searches the cache of the actual classloader, if the corresponding class was already loaded.
- `protected Class findSystemClass(String name)`
  Tries to load a class with the help of the system classloader.
- `protected void resolveClass (Class c)`
  Links the `Class` object, such that it is instantiatable.
- `static ClassLoader getSystemClassLoader()`
  Returns the system classloader.
- `Classloader getParent()`
  Returns the classloader of the next higher hierarchy level.

Many classloader methods are declared `protected`, such that they can be used only by implementing a subclass of `ClassLoader`. The primary method is `loadClass`, whereas all others can be used internally to design the loading procedure. Additionally, only the method `findClass` should be overwritten, which is used to define a loading procedure or a self defined source of class files, which is internally used by `loadClass`. For defining a loading procedure of class files from byte arrays, they are read in by `defineClass` and linked by `resolveClass`. Subsequently, they are available from the standard implementation of `loadClass` [7]. Moreover, the self-defined classloader should be involved in the hierarchy, such that the executing thread uses it as standard context classloader to instantiate new objects.

2.4 Server Access

Systematically, the server access is separated into synchronous and asynchronous request-response communications. This is extended by the transmission of user code and the generation of actors. In general three access structures could be used.

RESTful Webservice

Representational State Transfer (REST) is an architecture, which offers the following principles [12]. Resources are addressable via *Uniform Resource Identifiers* (URI). Resources are manipulated in a representation oriented way, which means that a representation consists of a sequence of bytes and metadata, to describe those bytes. Metadata represented with the *Hyptertext Transfer Protocol* (HTTP) can be a request type, a parameter, or a MIME type. Each message is self explanatory, such that the message content is sufficient to tell the server what to do. There is no need of an implicit session, since there is no information that is not contained in the message. Communications are stateless and independent from former requests. *Hypermedia as the engine of application state* (HATEOAS) defines that interactions are data driven and document oriented. Links are used to connect data and clients recognize from the response, how to process within the context.

The communication interfaces are self-defined HTTP requests, which are comparable to a *Remote Procedure Call* (RPC) calling methods with parameters on server side and expect a return value as HTTP response. The HTTP basis is a system independent and well supported technology. It offers a simple mechanism for sending and receiving requests through to mighty servlet technology for server development. With the *Java API for RESTful webservices* (JAX-RS) a framework is available that offers declarative annotations for designing server functions. However, designing all requests is more elaborative. Without a standardised interface, the documentation should be done in a proper way to compensate this fact. Alternatively, a *source development kit* (SDK) could be provided for the clients.

WSDL/SOAP Webservice

This type of webservice is composed of two technologies, the *webservice description language* (WSDL) as interface definition language and the *simple object access protocol* (SOAP) to encapsulate data for transmission. Both are based on the *extensible markup language* (XML) and gained a lot of interest with the hype of *service-oriented architectures* (SOA). For an implementation, Java offers several robust webservice stacks, which enable a simpler development. The standard also offers extensive functionalities for authentication, encryption, and secured data transmissions. A clear benefit of this approach is its standardised interface and communication technology with a clear definition by WSDL. Moreover, *Java API for XML webservices* (JAX-WS) is a framework for Java, which utilizes several webservice stacks.

However, extensive technology knowledge is required on developer side and the project depends on webservice stacks, which come with the issue that extending the basic functionality may lead to incompatibilities. Finally, inside of a WSDL there is no metadata included to classify binary data, such that additional documentation is necessary.

Remote Method Invocation (RMI)

This is Java's variant of the *remote procedure call* (RPC) client-server model on an object oriented basis. Method calls are performed with a proxy object on client side that represents the actual object on server side. On server side the communication object is a skeleton and regulates the access on the actual object. Two subvariants are available: with unicast each remote connection is delegated to the same object and in contrast, activatable, where a new object is created for each connection establishment.

This technology is integrated in Java APIs. A bidirectional connection is established between client and server, which results in a complicated connection setup. This approach is restricted to use the Java language. Regarding mobile devices, the handling of potential connection losses in mobile networks is uncertain, since RMI requires a constant connection.

Evaluation results

All three technologies are usable as server interfaces. On a glassfish basis it is possible to implement them as parallel access points for the actor system component. The last approach RMI, however, expects an excessive effort while developing and establishing; also the problem of connection losses in an instabile mobile context must be solved. The RESTful webservice based on JAX-RS offers all functionality, however the development of the client component is more complex, since the HTTP-requests have to be designed; in addition an interface definition is not automatically available. That is why a WSDL/SOAP webservice is the first choice for the access systematic. On server side, the functionality is configurable with annotations in the source code and the corresponding WSDL file as interface definition is generated automatically after deploying the program on a glassfish server. For the client side, it is possible to generate classes with tools of the webservice stack from a WSDL file.

2.5 Server and Middleware Technologies as Execution Environment

The execution environment should support Java and a free usage of threads. The first condition simplifies the usage with Android, while the free thread requirement is function critical for an actor system.

Cloud Computing Platforms Cloud computing platforms offer different kinds of services to third parties to use their system resources. This begins with predefined software for web applications over runtime environments to execute own programs, to complex network and storage solutions dependent on the offer of the cloud platform. For the server software the platform as a service is most promising, as it offers the possibility to run programs on cloud platforms. Therefore, the application has to be rewritten using platform dependent SDKs. In the following both platforms Windows Azure [11] and Google's AppEngine [5] with Appscale [1] are analysed, if their service supports the actor framework and the access technology. The main pitfall impacted by the architecture of the cloud platforms and the access on the SDK functionality restricts the possibility of thread generation and management. This causes malfunctions in combination with the actor system.

Google's AppEngine

This system supports Java as development language. However, extensive trials with a test implementation revealed massive restrictions when threads are generated. The Java *standard edition* (SE) functionality is replaced by a Google implementation, which enables the cloud platform to manage the threads. In the program code, the class `ThreadManager` for the generation of new threads according to the factory design pattern has to be used. A modification of the akka framework, especially the class `MonitorableThreadFactory` was identified as component for the internal thread generation and rewritten, such that no Java SE functionality is used anymore, but Google's implementation. Hence, the thread generation is properly working, new restrictions occurred; the akka framework calls the method `setDaemon` on the thread objects internally, which causes Google's security manager to throw an exception.

Windows Azure

The documentation revealed, similar to Google's AppEngine, a restricted thread usage.

Java EE with Glassfish To develop scalable distributed systems, Java *enterprise edition* (EE) comes with an API and runtime environment for a simplified development. According to the *enterprise Java bean* (EJB) API, the management of threads is done by the server component. Server internal management of instances of program components may cause an overlapping between runtime environment and user threads, when a self-defined thread is generated. User threads can be multiply created by a server internal instantiation of program components, what uses resources and disturbs a clean access of themselves. To solve this problem an architecture internal singleton EJB can be used. This variant of a session EJB is generated only once per running application by the Java EE runtime environment, thus this omits a multiple generation of user threads through internal mechanisms. However, an access synchronization on the threads is necessary. The actor framework uses Scala's own executor framework and its own thread management. In a test implementation the actor system is executed successfully in a singleton EJB. Moreover, Java's webservice APIs can be used for an efficient access on a consistent thread management. There is a fracture between actor and server threads, but both thread managements are logically separated and cause at most a not optimal scheduling of both thread groups inside the JVM. Additional trial implementations showed pitfalls by using this platform. Internally, glassfish uses a strict classloader hierarchy. It prevents that different Java EE programs gain mutual code access on each other. Furthermore, glassfish uses the executor framework of Java, which prevents to assign self-defined classloader to executing threads. In the trial environment it is not possible to add self-defined classloaders to the existing hierarchy to load classes from byte arrays; the glassfish internal loading procedure is designed to circumvent self-defined classloaders. To use self defined classes at runtime, it is necessary to copy them directly into the classpath folders of the corresponding classloaders or to deploy them together with the program on the server.

Stand-Alone-Server Implementation In this approach, no middleware would be used. The complete server component would be written manually with the help of Java's internal frameworks; this is a burdensome implementation, but the only one that results in full functionality. The thread management is completely designable, including the actor framework with configuration files. Without the restrictions of a middleware, a construction of a self-defined classloader hierarchy is

possible with this architecture, including a self-defined classloader in the execution environment of the actor framework.

Decision on Used System Finally, two types of systems are capable. The first system, which is also the current trial implementation is based on glassfish. Though, the loading procedure of remote classes is not implementable, this program enables to offload program routines from a mobile device. Therefore, the corresponding class files are deployed with the server implementation or copied into the classpath of the glassfish server. The second system would be a stand-alone-server program without middleware technology, enabling full functionality including the remote class loading procedure. The final decision must be this stand-alone-server program, but the functionality besides the remote class loading is implemented on a glassfish basis and therefore, trial programs can already be tested.

3 Architectural Design

This section lays the foundation for the first architectural design of the server in subsection 3.1. Moreover the client generation is described in subsection 3.2.

3.1 The Server Component

The server component implemented on a glassfish basis is separated into two subcomponents. The EJB component, which contains the actor system and all internal methods for accessing it. And the user access with a web component, which consists of an annotated class using the JAX-WS standard. To simplify deployment, the software is designed in a way to work with a standard setup of a glassfish server. Without a database for the trial implementation, all data is only available during the runtime of an application in the memory. However, it is important to copy the jar libraries of akka into the domain folder of the glassfish server.

The project is divided into three subprojects:

- *dynOff-EJB* contains the EJB component of the actor environment. The access on it is provided by the web component.
- *dynOff-Web* consists of a webservice for the remote access.
- *dynOff-EAR* is a project to encapsulate the previous two projects and to generate a deployable *enterprise application archive* (EAR) file.

The Class Diagram of the Server Figure 3 shows the separation into two container components, the left one for EJB and the right one for the webservices. The diagram only depicts classes and methods with self defined program logic. Libraries of Java and akka are not contained for providing a better overview.

Description of the Server Functionality Usage of the server functionality and including the akka libraries into glassfish's classpath, is the last part necessary for understanding the system. The libraries, including the Scala ones, are available on the akka.io website and must be separately downloaded to use the server component. Alternatively, the project is built with Maven and generates a Docker image, which can be deployed on a Docker enabled server. The implementation relies on akka in version 2.5.3, though no problems should occur with future versions, as long as there are no changes on the basic functionality. All library files for akka must be copied into the classpath of the glassfish server. Recommended by glassfish's documentation is to include user libraries into the directory *domain-dir*/lib. It is also possible to include the libraries into the EAR container, but it will increase the size of the container and slow down deployment. Executable code is bound to actors and controlled by messages.

Figure 4 depicts the general system usage in more detail:

1. Generation of actors with the webservice function `generateActorFromProps` or `generatePreAvailableActor`.
2. Transmission of a message to a generated actor via `sendMessage` or `dispatchAsyncJob`.
3. Reception of a message object either synchronously or asynchronously via `getAsyncJobResult`.

Actor Generation All actors must inherit the class `AbstractActor` of akka. All classes used inside the actors, including the actor and message classes, must be integrated in the classpath of the glassfish server either on application or on server level. For the generation of actors with the method `generateActorFromProps` a serialised instance of the configuration class `Props` is needed in a byte array representation and in return a string with identification features is generated. This string is used by the actor for identification. The return string of the method `getPreAvailableActor` has the same functionality. An example implementation is found in the test actor.

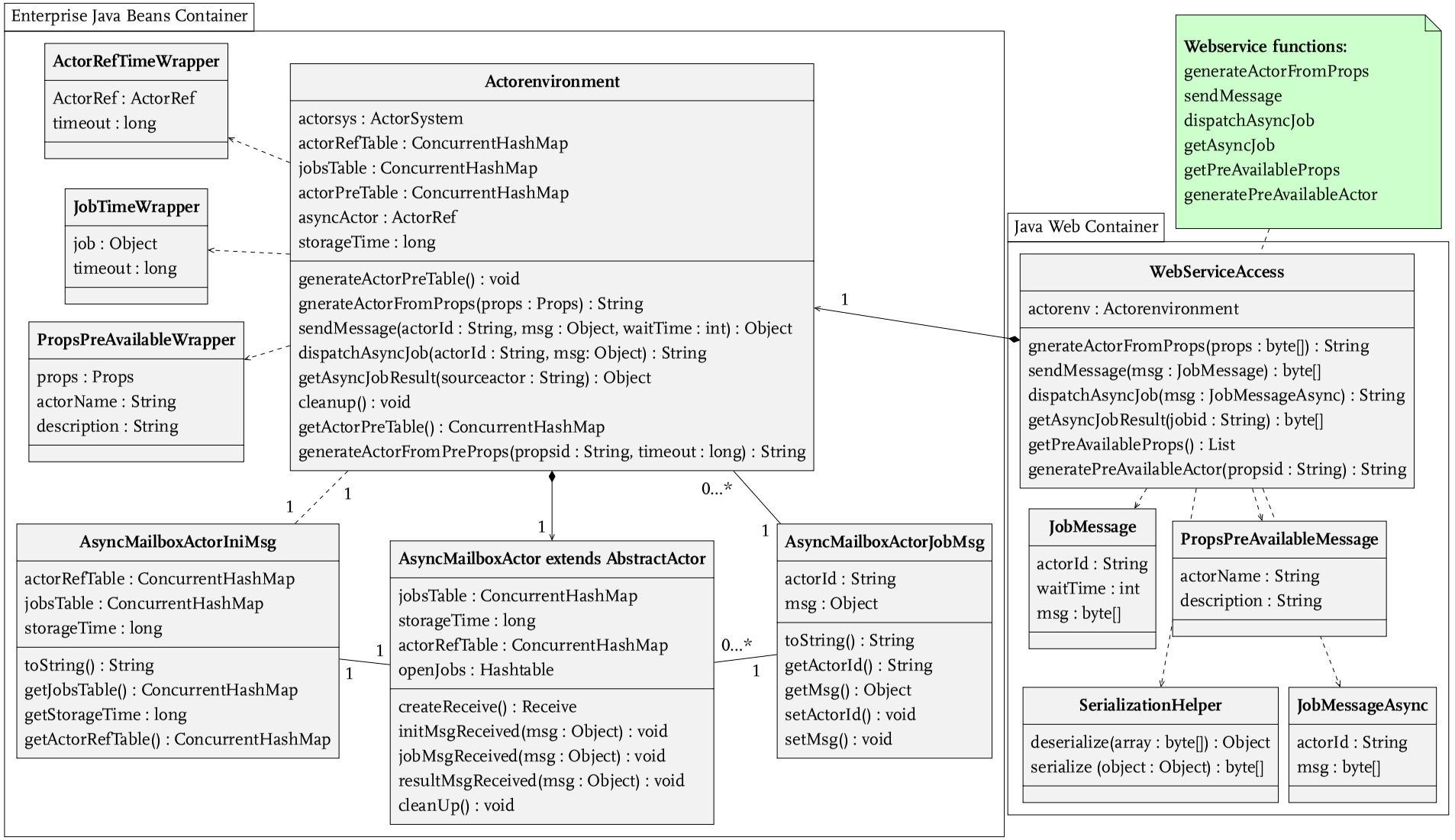


Fig. 3. An UML class diagram of the glassfish implementation. It shows the modular structure of the EAR archive, which contains two containers: the *EJB* container, which is responsible for exchanging messages with the actor environment, and the *web* container that publishes the services via web service functions described in WSDL.

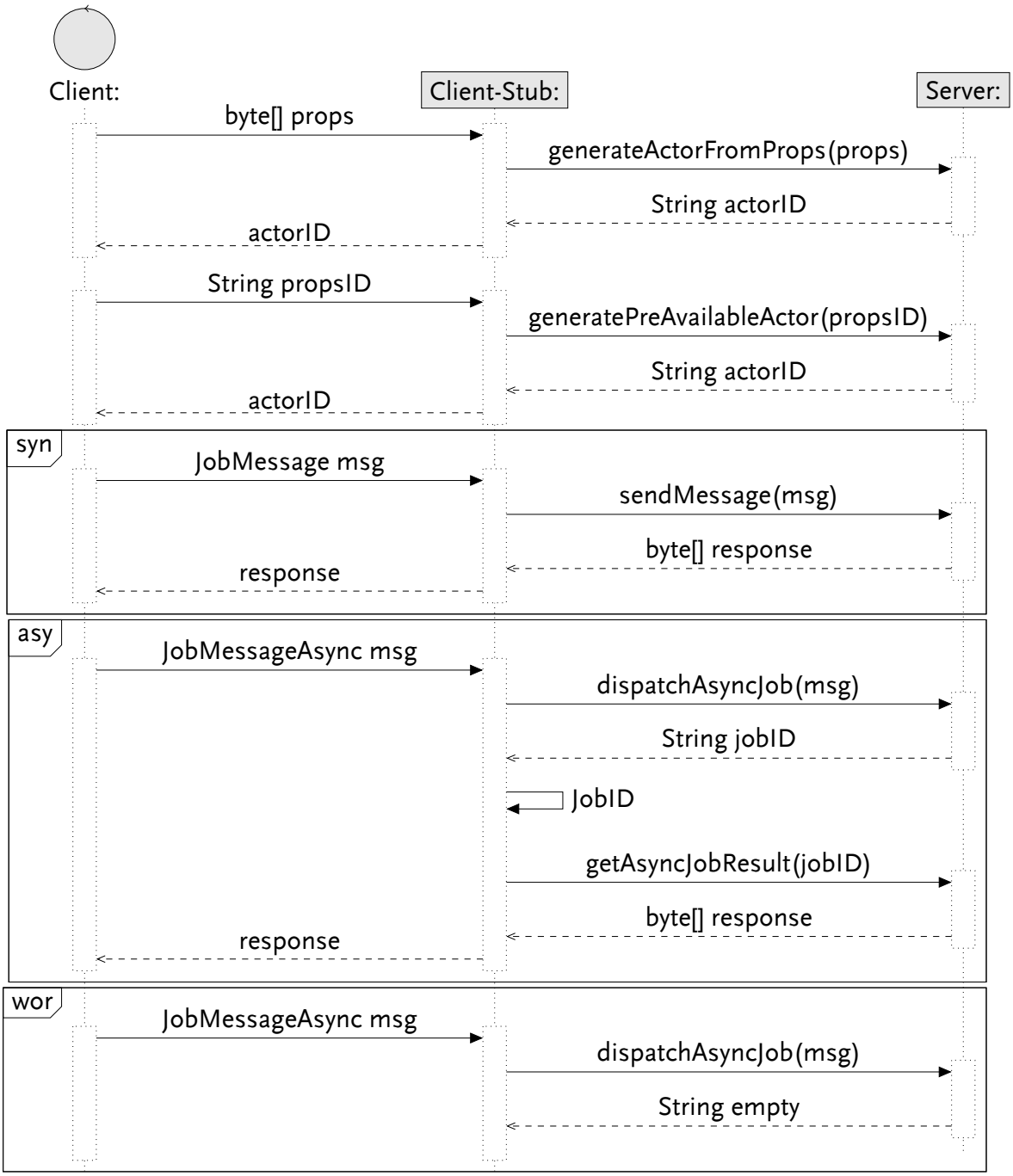


Fig. 4. Sequence diagram of communication processes between a client and a server. The job message *msg* can be transmitted with three different paradigms: either *syn*, which describes a synchronous pattern, *asy*, which shows the asynchronous communication, or *wor*, which shows an offloading without a reponse.

Message Transmission Message transmission is done by either `sendMessage` or `dispatchAsyncJob`. The first uses synchronous communication and takes a `JobMessage` object as encapsulating object for the SOAP transmission. This object contains an actor identification string, a serialised message object and a maximum response time. The second method provides an encapsulating object without a timeout and returns a job identification string, which enables a delayed message call via `getAsyncJobResult`. For the case that no response is expected, it is advisable to send an empty object as answer from your own actor.

Message Reception With a synchronous communication by `sendMessage`, a serialised response object is returned directly. If the wait time is expired a `ServerFault` exception is thrown. Using asynchronous communication the response message must be called with the job identification string with the method `getAsyncJobResult`. If a response message is not available, a `ServerFault` exception is thrown, too. For a successful deserialisation and casting into the message class, it is necessary that the corresponding class files are available on the client and on the server.

3.2 The Client Access

The implementation on client side depends on the webservice technology with its access based on WSDL/SOAP. Since the API of JAX-WS was already used for the preparation of the webservice functionality and is included into the Java SE standard libraries, the following description is based on it. After deploying the project on the glassfish server, a WSDL file is available via an URL. With the help of this file and the tool *wsimport*, classes and interfaces can be generated, which can be instantiated into a service object on runtime. On this object the service functions can be called like methods. The generated artefacts can be divided into the following categories:

– *Message artefacts*
  Classes for message transport annotated regarding the *Java architecture for XML binding* (JAXB). For each request and response as well as own classes for complex encapsulated content and exceptions. All classes are necessary for the functions.

– *Helper artefacts*
  An `ObjectFactory` class to generate classes regarding the factory pattern and a `package-info` class with annotations for setting the namespaces in the transmitted XML code. These classes are not relevant for the functions.
– *Stub artefacts*
  Consisting of an interface with JAX-WS annotated methods for using the webservice and a factory class for generating stub instances, which can be used on the former mentioned interface. Both classes are function critical.

3.3 An Actor Example

With the trial program code an example actor is delivered, which implements a simple echo call. It consists of two classes, a `TestActor`, which is the actor class and a `TestMessage` as message class, which expects an actor and transmits it. Instances of `TestMessage` contain a String that the actor sends back as response.

4 Conclusion & Future Work

The concepts of remote usage of the akka framework for dynamic code upload from a mobile device were described and a sample architectural implementation is made publicly available at the github repository whatever4711/dynOff[1]. This platform is hopefully used for dynamic code offloading experiments soon. A great advantage is the usage of the akka.io framework, which simplifies outsourcing user code, since Java libraries and resources are used, which offer a widespread area of application. Furthermore, akka with its actor model eliminates any concurrency issues when it is deployed on a server. Moreover, akka offers a great potential by configuring its execution environment and may even be distributed onto a cluster of servers to be more efficient. Thus, the offloading procedure sounds useful with the help of a server, it is also possible to run the actor system itself on the mobile device by using a background service, too. Nevertheless, a potential area of application is to avoid the bottleneck of a mobile network by outsourcing important communication structures onto a server with high-speed network access. However, security issues were not focused until now and should be analysed, especially regarding the WSDL access of the server communication. Another point

[1] https://github.com/whatever4711/dynOff

is a migration of the Java EE functions into a Java SE, since Android devices are not supporting a Java EE environment. Finally, with the stand-alone-server implementation it should be possible in future to extend it with own classloader functionalities, such that realtime code offloading is enabled.

References

1. AppScale Systems Inc. AppScale, 2013.
2. Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. CloneCloud: Elastic Execution Between Mobile Device and Cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.
3. Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.
4. H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya. Mobile code offloading: from concept to practice and beyond. *IEEE Communications Magazine*, 53(3):80–88, March 2015.
5. Google. Google App Engine, 2013.
6. Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular ACTOR formalism for artificial intelligence. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, IJCAI'73, pages 235–245, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.
7. IBM. *Understanding the Java ClassLoader*, 2011.
8. Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: A Computation Offloading Framework for Smartphones. In Martin Gris and Guang Yang, editors, *Mobile Computing, Applications, and Services*, volume 76 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 59–79. Springer Berlin Heidelberg, 2012.
9. S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *2012 Proceedings IEEE INFOCOM*, pages 945–953, March 2012.
10. Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley. *The Java Virtual Machine Specification - Java SE 8 Edition*, 2015.
11. Microsoft. Windows Azure, 2013.
12. Oracle. Java EE 6 Tutorial, 2013.
13. Typesafe Inc. akka.io, 2013.
14. Christian Ullenboom. *Java ist auch eine Insel Einführung, Ausbildung, Praxis*. Rheinwerk, Bonn, 2016.

Author Index

University of Bamberg Press

Nowadays, mathematical methods of systems and network monitoring, modeling, simulation, and performance, dependability and reliability analysis constitute the foundation of quantitative evaluation methods with regard to software-defined next-generation networks and advanced cloud computing systems. Considering the application of the underlying methodologies in engineering practice, these sophisticated techniques provide the basis in many different areas.

The GI/ITG Technical Committee "Measurement, Modelling and Evaluation of Computing Systems" (MMB) and its members have investigated corresponding research topics and initiated a series of MMB conferences and workshops over the last decades. Its 9th GI/ITG Workshop MMBnet 2017 „Leistungs-, Verlässlichkeits- und Zuverlässigkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen" was held at Hamburg University of Technology (TUHH), Germany, on September 14, 2017. The proceedings of MMBnet 2017 summarize the contributions of one invited talk and four contributed papers of young researchers. They deal with current research issues in next-generation networks, IP-based real-time communication systems, and new application architectures and intend to stimulate the reader's future research in these vital areas of modern information society.

eISBN: 978-3-86309-502-4

www.uni-bamberg.de/ubp