Technische Universität Berlin

Informed Access Network Selection to Improve Application Performance

vorgelegt von

M.Sc.

Theresa Enghardt

an der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Georgios Smaragdakis, Ph. D., TU Berlin
Gutachterin:	Prof. Anja Feldmann, Ph. D., Max Planck Institute for Informatics
Gutachter:	Prof. Dr. habil. Thomas Zinner, TU Berlin
Gutachter:	Prof. Dr.-Ing. Jörg Ott, TU Munich

Tag der wissenschaftlichen Aussprache: 05. November 2019

Berlin 2019

DOI: http://dx.doi.org/10.14279/depositonce-9410

This work was supported in part by the EU project CHANGE (FP7-ICT-257422) and Leibniz Prize project funds of DFG (Leibniz-Preis 2011 – FKZ FE 570/4-1).

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich diese Dissertation selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Datum Theresa Enghardt

Abstract

Adverse network conditions, such as long latency or low downstream capacity, may degrade application performance and lead to a bad Quality of Experience (QoE) for users, such as long Web page load times or interrupted video playback. As applications are highly diverse, connectivity via a single access network may not be able to satisfy all application needs. Since an end-user device often has multiple access networks available, there is the potential to select between access networks or to aggregate the capacity of multiple access networks. However, this potential is not yet fully utilized due to the limitations of existing systems and the networking Application Programming Interfaces (APIs) they provide. Many end-user devices today default to WiFi and use cellular only as a fallback, even though WiFi may provide suboptimal performance for some or for all applications. Existing ways of aggregating multiple access networks, such as Multipath TCP (MPTCP), are typically application-agnostic and may not provide benefits for all types of applications or for all kinds of traffic.

Therefore, the goal of this thesis is Informed Access Network Selection (IANS) enabling end-user devices to select the best suitable access network(s) based on application needs and network performance characteristics. To this end, we design Socket Intents as an abstraction for application needs, collect network performance characteristics, and design IANS policies for Web browsing and HTTP Adaptive Streaming (HAS). We implement IANS within the Socket Intents prototype, which enables applications to communicate their needs through an enhanced networking API. Using the Socket Intents prototype, we evaluate the performance benefits of IANS for Web browsing and HAS compared to using a single access network and using MPTCP.

For Web browsing, we find that IANS can improve relevant Web performance metrics, e.g., Above-The-Fold times, by between 500 and 1000 ms in the median compared to using the faster single access network. IANS shows the most significant speedups under asymmetric network conditions with short latency but low downstream capacity on one network and high downstream capacity but long latency on another network. In such cases, IANS outperforms MPTCP, as MPTCP can experience performance problems caused by self-induced congestion on the low downstream capacity network. For HAS, IANS shows the most significant benefits in scenarios with low available downstream capacities. Here, we see cases in which IANS improves HAS performance substantially, e.g., a “bad” Mean Opinion Score (MOS) of 2.1 on a single network to a “good” MOS of 2.8, while MPTCP hurts performance as it continues to use a low downstream capacity network.

In conclusion, we find that access network selection benefits from application awareness. Especially in scenarios with low downstream capacity on one or more available access networks, IANS improves application performance for Web browsing and HAS by selecting the most suitable single access network or combining multiple access networks if possible. Using an enhanced networking API allows us to make IANS available to different applications. Moreover, such an API may enable applications to use new protocols without requiring further modifications.

Abstract

Schlechte Bedingungen im Netzwerk, wie beispielsweise hohe Latenz oder niedrige Downstream-Kapazität, können die Performance einer Applikation vermindern und zu einem schlechten Nutzererlebnis führen, etwa zu langen Ladezeiten einer Webseite oder zu Unterbrechungen des Abspielens eines Videos. Da Applikationen sehr unterschiedlich sind, kann die Anbindung durch ein einzelnes Zugangsnetzwerk möglicherweise nicht die Anforderungen aller Applikationen erfüllen. Weil ein Endgerät oft mehrere Zugangsnetzwerke zur Auswahl hat, besteht das Potential, das passendste Netzwerk für eine Applikation auszuwählen oder die Kapazität mehrerer Zugangsnetzwerke zusammenzufassen. Jedoch wird dieses Potential noch nicht vollständig ausgeschöpft, da existierende Systeme und die von ihnen zur Verfügung gestellten Netzwerk-Applikationsschnittstellen (APIs) dies oft nicht unterstützen. Viele der heutigen Endgeräte benutzen automatisch WiFi und greifen auf Mobilfunknetze nur als Reserve zurück, obwohl WiFi möglicherweise suboptimale Performance für manche oder alle Applikationen bietet. Existierende Technologien um mehrere Zugangsnetzwerke zusammenzufassen, wie etwa Multi-Path TCP (MPTCP), haben kein Wissen über die Applikation und stellen möglicherweise nicht für alle Arten von Applikationen oder für allen Datenverkehr Verbesserungen bereit.

Daher ist das Ziel dieser Dissertation Informed Access Network Selection (IANS), die Endgeräten ermöglicht, basierend auf den Bedürfnissen der Applikation und der Netzwerkperformance das oder die am besten geeignete(n) Zugangsnetzwerk(e) auszuwählen. Um dies zu erreichen, entwerfen wir Socket Intents als eine Abstraktion für Applikationsanforderungen, ermitteln die Performanceeigenschaften der Netzwerke, und entwerfen IANS-Policies für Web-Browsing und HTTP Adaptive Streaming (HAS). Wir implementieren IANS innerhalb des Socket Intents-Prototypen, der Applikationen erlaubt, ihre Anforderungen durch eine erweiterte Netzwerk-API zu kommunizieren. Unter Benutzung des Socket Intents-Prototypen evaluieren wir die Performanceverbesserungen durch IANS für Web-Browsing und HAS im Vergleich zu der Benutzung eines einzelnen Zugangsnetzwerks und MPTCP.

Für Web-Browsing stellen wir fest, dass IANS relevante Web-Performance-Metriken verbessert, indem sie beispielsweise Above-The-Fold-Zeiten um zwischen 500 und 1000 ms im Median verkürzt, verglichen mit dem schnelleren einzelnen Zugangsnetzwerk. IANS zeigt die größten Verbesserungen unter asymmetrischen Netzwerkbedingungen mit kurzer Latenz aber geringer Downstream-Kapazität auf einem Netzwerk und hoher Downstream-Kapazität aber hoher Latenz auf einem anderen Netzwerk. In solchen Fällen erreicht IANS eine größere Verbesserung als MPTCP, da MPTCP Performance-Probleme herbeiführen kann, indem es das Netzwerk mit der geringeren Downstream-Kapazität überlastet. Für HAS zeigt IANS die signifikantesten Verbesserungen in Fällen, in denen nur geringe Downstream-Kapazität in den Zugangsnetzwerken verfügbar ist. Hier sehen wir Fälle, in denen IANS die Performance von HAS substanziell verbessert, etwa von einem “schlechten” Mean Opinion Score (MOS) von 2.1 auf einem einzelnen Netzwerk auf einen “guten” MOS von 2.8, während MPTCP die

Performance verschlechtert, da es ein Netzwerk mit geringer Downstream-Kapazität weiterhin benutzt.

Abschließend stellen wir fest, dass die Auswahl zwischen Zugangsnetzwerken von Applikationswissen profitiert. Insbesondere in Szenarien mit niedriger Downstream-Kapazität auf einem oder mehreren verfügbaren Zugangsnetzwerken verbessert IANS die Performance der Applikationen Web-Browsing und HAS, indem es das am besten passende Zugangsnetzwerk auswählt oder mehrere Zugangsnetzwerke kombiniert, falls möglich. Die Verwendung einer erweiterten Netzwerk-API erlaubt es uns, IANS verschiedenen Applikationen zur Verfügung zu stellen. Desweiteren kann eine solche API es Applikationen ermöglichen, neue Protokolle zu verwenden, ohne dass die Applikationen noch einmal modifiziert werden müssen.

Acknowledgments

The time of doing my PhD has been an intense few years during which I have worked with many fascinating people and learned lots of things. I have grown as a researcher, as an engineer, and as a person. My sincere thank you goes out to all the people who have accompanied and supported me on this path.

First of all, a huge thank you to my advisor, Anja Feldmann. Thank you for providing me with the opportunity to do this PhD with you. You have taught me how to write papers, how to do a proper performance evaluation, and how to analyze data. We have had intense and less intense deadlines. Working with you has been a great experience and has made a lasting impression on me. Thank you for having continued to be my advisor even all the way across Germany and for having welcomed me in Saarbrücken many times.

Thank you to my other professor, Thomas Zinner. Since you joined INET at TU Berlin, you have held this group together and given me an abundance of feedback and support. We have had great research discussions and collaborations on papers. In the process, I have gained substantial knowledge about video streaming, including the term HAS and how to model QoE – all while we still got our teaching done. Impressively, with you, navigating the challenges of TU Berlin and handling students has even been fun at times.

Thank you to my collaborator Philipp Tiesel for introducing me to the idea of Socket Intents as well as coming up with cool acronyms such as IANS, MUACC, and MAM. Thank you for many of the foundational and forward-looking design decisions within the Socket Intents prototype and for the numerous valuable contributions that you made to this project, our project. During our time of extensive collaboration on papers, drafts, code, and supervising students, we have gone through a lot together. Working with you has greatly influenced me.

Thank you to my colleagues at INET, with many of whom I have also become friends. Thank you to Franziska Lichtblau and Florian Streibelt for your hospitality when I visited MPI in Saarbrücken. Franziska, thank you for your feedback on my presentations and thank you for encouraging me to come to a RIPE meeting. Florian, thank you for helping me fix that code. Thank you to Mirko Palmer for sharing an office with me in Berlin and for the discussions on MPTCP and QUIC. Thank you to Matthias Rost for the help with algorithms and definitions. Thank you also to Thorben Krüger, Thomas Krenc, Lars Prehn, Georgios Smaragdakis, Habib Mostafaei, Susanna Schwarzmann, Apoorv Shukla, Aniss Maghsoudlou, Said Jawad Saidi, Balakrishnan Chandrasekaran, and all other members of the INET family who have walked parts of the way with me. A special thank you goes to our sysadmins, especially to Sarah Dierenfeld, Rainer May, Maxi Türke, Sabet Peters, Sebastian Lohff, Christian Struck, and Dimitri Bulgakov. Thank you for setting up and maintaining the IT infrastructure of this group, such as parts of the testbed that my experiments were running on. Thank you for putting out the fires, metaphorically.

Outside of the INET group, I have had the pleasure to meet and work with lots of people in the networking community. In particular, the IETF community has made a profound and lasting positive impression on me. Thank you especially to Brian Trammell and Mirja Kühlewind for encouraging me to participate in this community more and for making the transport area awesome. Thank you to all members of the TAPS Working Group, especially to those with whom I have the pleasure to collaborate on documents and/or code. In the wider IETF community, thank you to Jörg Ott for the research discussions and the feedback. A special thank you goes to Paul Hoffman. Thank you so much for telling me what I needed to hear and for inspiring me to grow.

In the wider networking community, such as the RIPE and the Linux networking community, thank you to Stefan Wahl for the immensely helpful comments on parts of my thesis, and thank you to Stephen Hemminger for the feedback and discussion on networking APIs.

Last but not least, thank you to my lovers, friends, and family. Thank you to my girlfriend Maya for accompanying me and being my balance through all these years and transformations. Thank you to my friends in Berlin, especially to Seba and Kai who are regularly bringing our group of friends together. Thank you to my friend Lena for the long phone calls and the fun biking tours. Thank you to my friend Konstantin for proofreading parts of this thesis. Thank you to my flatmates Paula and Miri for bearing with me even though I am rarely home, for making our living environment comfortable, and for being good company. Thank you to my Mom and Dad as well as my brothers for the continued support.

Publications

Pre-published Papers

Parts of this thesis are based on the following peer-reviewed papers that have already been published.

I thank my collaborators for the valuable contibutions they have made. All collaborators to this thesis are listed here, either as co-authors of joined publications or with their kind of collaboration.

International Conferences

Philipp S. Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. "Socket Intents: Leveraging Application Awareness for Multi-access Connectivity". In: Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. *ACM CoNEXT 2013*. ACM, 2013, pp. 295-300. ISBN: 978-1-4503-2101-3. DOI: 10.1145/2535372.2535405.

Theresa Enghardt, Thomas Zinner, and Anja Feldmann. "Web Performance Pitfalls". In: Choffnes D., Barcellos M. (eds) Passive and Active Measurement. *PAM 2019*. Lecture Notes in Computer Science, vol 11419. Springer, Cham, pp. 286-303. ISBN: 978-3-030-15986-3. DOI: 978-3-030-15985-6.

Theresa Enghardt, Philipp S. Tiesel, Thomas Zinner, and Anja Feldmann. "Informed Access Network Selection: The Benefits of Socket Intents for Web Performance" In: Zincir-Heywood N., Badonnel R. (eds) 15th International Conference on Network and Service Management. *CNSM 2019*. IFIP Open Digital Library, IEEE Xplore. ISBN: 978-3-903176-24-9

Workshops and Poster Sessions

Theresa Enghardt, Philipp S. Tiesel, and Anja Feldmann. "Metrics for access network selection". In: Proceedings of the Applied Networking Research Workshop. *ANRW 2018*. ACM. 2018, pp. 67-73 ISBN: 978-1-4503-5585-8. DOI: 10.1145/3232755.3232764.

Under submission

Parts of this thesis are based on the following paper that is currently under submission.

International Conferences

Theresa Enghardt, Thomas Zinner, and Anja Feldmann. "Using Informed Access Network Selection to Improve HTTP Adaptive Streaming"
Under submission.

Internet-Drafts

Parts of this work have served as input to the following Internet-Drafts, which are work items of the IETF TAPS Working Group with the goal of being published as RFCs:

Brian Trammell, Michael Welzl, Theresa Enghardt, Gorry Fairhust, Mirja Kuehlewind, Colin Perkins, Philipp Tiesel, and Christopher Wood. An Abstract Application Layer Interface to Transport Services (work in progress). Internet Draft draft-ietf-taps-interface-05. IETF, Nov. 2019
`https://tools.ietf.org/html/draft-ietf-taps-interface-05`

Anna Brunstrom, Tommy Pauly, Theresa Enghardt, Karl-Johan Grinnemo, Tom Jones, Philipp S. Tiesel, Colin Perkins, Michael Welzl. Implementing Interfaces to Transport Services (work in progress). Internet Draft draft-ietf-taps-impl-05, Nov. 2019
`https://tools.ietf.org/html/draft-ietf-taps-impl-05`

Parts of this work have been submitted as individual Internet-Drafts and presented at IETF meetings:

Philipp Tiesel, Theresa Enghardt, and Anja Feldmann. Socket Intents (work in progress). Internet Draft draft-tiesel-taps-socketintents-01. IETF, Oct. 2017
`https://tools.ietf.org/html/draft-tiesel-taps-socketintents-01`

Theresa Enghardt and Cyrill Krähenbühl. A Vocabulary of Path Properties (work in progress). Internet Draft draft-enghardt-panrg-path-properties-03. IETF, Nov. 2019
`https://tools.ietf.org/html/draft-enghardt-panrg-path-properties-03`

Other collaborations

The design and implementation of the Socket Intents prototype was done in close collaboration with Philipp Tiesel.

The idea for the initial version of a precursor of the Threshold Policy, called the Earliest Arrival First Policy at the time, was developed in collaboration with Mirko Palmer in the context of his Master thesis, which I co-supervised. Since then, the Threshold Policy has undergone many iterations and refinements, which result in the final version presented in this thesis.

The network performance characteristics gathering contains code based on contributions by Puneeth Nanjundaswamy (download rate monitoring), Sören Becker (RTT monitoring), and Marcin Bosk (WiFi utilization monitoring). All of these contributions were made as part of students projects or Bachelor theses which I co-supervised.

Parts of the extension of the GPAC player to use Socket Intents are based on code contributed by Patrick Kutter in the context of his Master project and Master thesis, which I co-supervised.

Software

The following software developed as part of this work has been made publicly available:

Socket Intents prototype.
`https://github.com/fg-inet/socket-intents`

Web measurement tools.
`https://github.com/theri/web-measurement-tools`

Contents

1 Introduction

Access networks are critical components of the Internet: When a host sends traffic to another host through the Internet, this traffic usually has to go through an access network. As access networks provide varying network performance characteristics, e.g., different latency and upstream as well as downstream capacity [1, 2], they are often the bottleneck [3, 4]. Even if the bottleneck is within the backbone [5], paths via different access networks may traverse different bottlenecks. In such cases, the used access network may have a significant impact on application performance: Bad conditions, such as long latency or low downstream capacity, may lead to degraded application performance, and, thus, to an unsatisfactory Quality of Experience (QoE) for users [6].

This problem has aggravated over the years because both user expectations regarding application performance have grown and application demands have increased, e.g., in terms of short latency and high downstream capacity. For example, while video streaming accounts for a significant share of Internet traffic, i.e., more than 40% [7] of mobile traffic share is due to a combination of popular video on demand services, video streaming QoE is impacted by network conditions such as limited downstream capacities, long latencies, and packet loss. Furthermore, Web pages have increased in size [8], which leads to an increased downstream capacity demand within access networks for Web browsing.

Moreover, application demands are diverse. While many mobile Web pages are large, they also include many small resources [9]. In addition, mobile applications often issue queries of only a few kilobytes to interactively display the result to the user. For such small resources and short queries, application performance may benefit from a network with short latency. However, applications may also load large resources, e.g., a photo to display to the user. Such applications may require a network with high downstream capacity to keep user-visible delays short. The available access network may not provide either of these network performance characteristics. Instead, it may provide short latency, but low downstream capacity, or it may provide high downstream capacity, but long latency.

To improve performance, either access network operators have to upgrade their networks, or end-user devices have to use the available networks more efficiently. As upgrading access networks can be costly [10], there is a need to the optimize access network usage of end-user devices. One such option is for the end-user device to use multiple access networks at the same time. In the last ten years, this has become

feasible, as multiple access networks are often available [11–13]: While Internet connectivity used to be scarce, i.e., only available in a few places, access network coverage has grown. Increasingly, multiple access networks are available to a single end-user device at the same time. Moreover, access network technology has improved, e.g., the capacity of cellular networks has increased [14] and data plans have also increased. Therefore, using cellular has become a feasible option even for applications with demands for high downstream capacity or low latency. Finally, end-user devices often have multiple network interfaces. Therefore, increasingly, multiple access networks are available, and end-user devices are able to use them.

Yet, most end-user devices today only use a single access network. For example, many mobile end-user devices default to WiFi if available and use cellular only as a fallback [15]. Using WiFi may lead to suboptimal application performance, as WiFi does not always provide better performance than cellular [2]. For example, a WiFi network might be overloaded with too many end-user devices sending traffic on the same channel, increasing latency. In addition to the wireless link itself, there might also be limited capacity on the uplink. If the WiFi network provides long latency or insufficient downstream capacity for the applications on the connected end-user devices, this may lead to inadequate performance for some or for all applications. Furthermore, WiFi networks are often limited in range. If a mobile end-user device moves out of range while an application still has active connections, the end-user device has to fall back to another access network, e.g., cellular. With Transmission Control Protocol (TCP), the application has to re-establish all connections via the cellular network, which may lead to user-visible delays within the application.

To overcome the limitations of using only a single access network, multipath transport protocols such as Multipath TCP (MPTCP) [16] have been developed. MPTCP can use multiple access networks for a single connection and, thus, aggregate their downstream capacities. Moreover, MPTCP can provide seamless fallback in case connectivity over one network fails. Unfortunately, MPTCP has seen limited deployment so far, as it needs support from both end-user devices and servers. Moreover, some access networks may deploy middleboxes, which prevent MPTCP usage [17]. Even if available, MPTCP does not provide performance benefits for all traffic: Prior work suggests that while MPTCP provides performance benefits for loading large resources, it may not speed up loading small resources [17, 18]. In addition, MPTCP performance may be sensitive to access networks which provide asymmetric network performance characteristics, e.g., if one network provides shorter latency than another [19]. If this access network suffers from bad performance, e.g., due to low downstream capacity and congestion, this might have a severe impact on the performance of the MPTCP connection.

Given the limitations of using a single access network for all traffic or aggregating the performance of multiple networks in an application-agnostic way, there is the need to develop new ways of selecting an access network with respect to application requirements and network performance characteristics. Therefore, end-user devices should choose a single access network that provides the best possible performance for any given application, or aggregate the capacity of multiple networks.

1.1 Goals

The objective of this thesis is Informed Access Network Selection (IANS) using application needs and network performance characteristics to improve application performance.

To determine whether we have achieved this objective, we have to accurately assess application performance. Hereby, we focus on the two currently most prevalent applications, Web browsing and video streaming. For Web browsing, we investigate how to measure Web performance so that the results are realistic and reproducible by studying different data sources and tools in detail. We complement this analysis by showing performance metrics relevant for HTTP Adaptive Streaming (HAS).

As application needs and traffic properties are diverse, we suggest to use this information for IANS. Therefore, we investigate how an application can express what it knows about its own traffic, i.e., what to optimize for when selecting an access network.

To meet application needs by selecting the more suitable access network, we have to know about the current network performance characteristics. Therefore, we examine what network performance characteristics are available on a host and how to get access to them.

Combining application requirements and network performance characteristics, we study how to select an access network based on this information. In particular, we explore strategies how to improve application performance for Web browsing and HAS by designing IANS policies for each.

We design and implement the Socket Intents prototype to enable applications to express their needs regarding a new connection or transfer, to get current network performance characteristics, and to select an access network based on this information.

Finally, we investigate the performance benefits of IANS for Web browsing and HAS both in a controlled testbed and "in the wild".

1.2 Contributions

Web Performance Pitfalls. First, to accurately assess Web performance, we examine different Web metrics regarding their data sources and measurement tools. Hereby, we discover several pitfalls which may hinder realism and reproducibility, and provide guidelines on how to mitigate them. For example, we find that initial redirects can account for a significant share of the load times of a Web page. Therefore, when measuring Web performance, one should make a conscious choice whether to include or exclude redirects. Furthermore, when investigating different

data sources for resource size and resource count of a Web page, we find that different data sources provide different degrees of reliability: While the Content-Length is most reliable, Resource Timings are an unreliable data source.

Socket Intents. Next, we design Socket Intents to enable applications to express what to optimize for when selecting an access network for them. Socket Intents are hints about what the application knows, expects, or wants to achieve regarding a new connection or transfer. They do not represent Quality of Service requirements or resource reservations, but are used to match application needs to the most suitable access network(s). As applications do not necessarily know what kind of network they prefer, different Intents exist to support different applications to express their needs in varying levels of detail.

Gathering Network performance characteristics. To learn about the performance of the currently available access networks, we analyze which network performance characteristics are relevant for access network selection and how to gather such network performance characteristics on a host. We find that latency and available downstream capacity are most relevant for the applications we consider, Web browsing and video streaming. Furthermore, hosts are able to gather latency and downstream capacity estimates with reasonable effort: For latency, we use Smoothed Round Trip Times (SRTTs), while for downstream capacity, we use maximum and currently observed data rates. As network performance characteristics may vary over time, we use estimates observed on different time scales.

Informed Access Network Selection Policies. Finally, we combine application needs and network performance characteristics within IANS policies, which select the most suitable access network for a new connection or transfer. We design IANS policies with the goal of improving the performance of Web browsing and HAS. For Web browsing, the Threshold Policy distributes individual Web resources onto access networks according to resource size as well as latency and available downstream capacity of the access networks. For HAS, the Optimist Policy and Pessimist Policy estimate the load times of video segments based on recent downstream capacity estimates. Then, the Optimist Policy additionally considers switching to an alternative network based on the "best case" load time using long term downstream capacity estimates. The Pessimist Policy additionally considers switching to an alternative network based on the "worst case" load time using short-term downstream capacity estimates. Finally, the Selective MPTCP Policy enables MPTCP only for bulk transfers if sufficient downstream capacity is available.

Socket Intents Prototype. We implement IANS, including Socket Intents, network performance characteristics gathering, and IANS policies, within the Socket Intents prototype. The prototype offers Application Programming Interfaces (APIs) for applications to use IANS per connection as well as per individual transfer. Using one of these APIs, we write a Web proxy to enable IANS for Web browsing and modify a video player to enable IANS for HAS.

Application Performance Benefits From Informed Access Network Selection. We evaluate the performance benefits of IANS for Web browsing and video

streaming via HAS in a testbed and using the actual Web servers the Web pages are hosted on. First, we confirm that our Socket Intents prototype accurately gathers network performance characteristics for the network scenarios we use in our evaluation. Then, we investigate the benefits of IANS in a systematic study. We find that IANS can significantly improve Web performance, e.g., reduce Above-The-Fold time (ATF) by between 500 and 1000 ms in the median for scenarios with asymmetric network performance characteristics. In such an asymmetric scenario, one network provides short latency but also low downstream capacity, while another network provides high downstream capacity, but also long latency. Under such conditions, IANS even outperforms MPTCP. For scenarios with symmetric network performance characteristics, IANS improves Web performance for cases with low downstream capacity. For scenarios with high downstream capacity on both networks, using a single network already yields good performance, so neither IANS nor MPTCP are able to improve ATF. For HAS, IANS can significantly improve QoE by avoiding to use a low downstream capacity network, thus, avoiding stalling of the video playout or loading low representations.

1.3 Structure of the Thesis

The rest of this thesis is structured as follows: In Chapter 2, we provide background on different access network technologies and their network performance characteristics. Furthermore, we define a multiple access network scenario and review Related Work that either shifts traffic between different access networks or aggregates them using multipath transport protocols. We also provide background on systems support for using multiple access networks. In Chapter 3, we examine how to accurately measure performance of two major applications, Web browsing and HAS. In Chapter 4, we show how application traffic properties and preferences can be expressed using Socket Intents. In Chapter 5, we analyze which network performance characteristics may play a role in access network selection and how to gather them on a host. In Chapter 6, we introduce IANS policies for Web browsing and HAS. In Chapter 7, we describe the design and implementation of the Socket Intents prototype, which includes Socket Intents, network performance characteristics gathering, and IANS policies. In Chapter 8, we evaluate the application performance improvement that these IANS policies can offer in different network scenarios. Finally, in Chapter 9, we discuss our findings and provide an outlook.

2

Background and Related Work

In recent years, end-user devices increasingly often have the ability to connect to multiple access networks at the same time. Two common access network technologies, which often provide Internet connectivity to end-user devices, are WiFi and cellular. In this chapter, we explain the aspects of WiFi and cellular networks which are relevant for the network performance characteristics they provide to end-user devices. Then, we present prior work examining and comparing the network performance characteristics of these access networks.

We then describe scenarios in which multiple access networks are available and can complement each other. We show two approaches to use these networks concurrently proposed in Related Work. First, mobile offloading allows to shift traffic from cellular to WiFi networks. Second, multipath transport protocols such as Multipath TCP (MPTCP) allow transferring data over multiple networks at the same time.

For taking advantage of multiple access networks in practice, technologies utilizing them have to be supported on actual systems. Yet, existing host Operating Systems (OSes) have a varying degree of support for using multiple networks. We first review the architecture of networking implementations on contemporary OSes and Application Programming Interfaces (APIs) and then present their support for multihoming and multipath technologies.

2.1 Access Networks

Network coverage providing access to the Internet has become nearly ubiquitous in many parts of the world. One major technology to facilitate this access is 802.11 Wireless Local Area Network (LAN), called WiFi in the rest of this thesis. For WiFi networks, we consider both wireless links, which often provide connectivity between end-user devices and Access Points (APs), as well as common uplinks to the Internet. The rationale for including uplinks in our analysis is that they may play a significant role in the network performance characteristics that a WiFi network offers to an end-user device. Another major type of access network technology is mobile cellular networks, called cellular in the rest of this thesis. As different cellular networks are deployed, we consider 2G up to 5G cellular networks. For both WiFi and cellular networks, we provide technical background on the underlying technologies as a foundation for understanding their network performance characteristics.

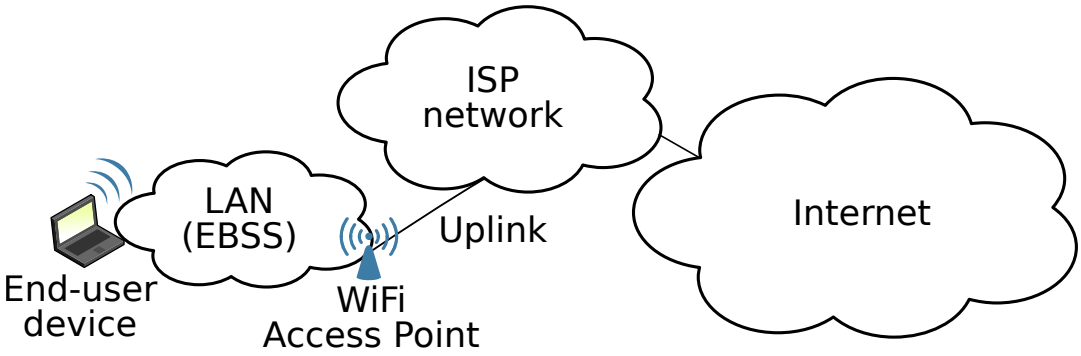


Figure 2.1: Architecture of a WiFi network.

2.1.1 WiFi Networks

We define a WiFi network as an access network which provides Internet connectivity to an end-user device over an 802.11 wireless link. Figure 2.1 shows the architecture of a WiFi network in infrastructure mode. An end-user device first becomes part of a LAN, which corresponds to an 802.11 Extended Basic Service Set (EBSS). The end-user device joins the LAN by associating to a WiFi AP, thus, establishing a wireless link. Once the end-user device is part of the LAN, it can also reach hosts outside of its LAN. For this, the end-user device needs to send its traffic via an uplink, which connects the LAN to a larger network, e.g., an Internet Service Provider (ISP) network. Finally, the ISP network provides connectivity to the rest of the Internet.

2.1.1.1 Wireless Links Using 802.11

In many WiFi networks, at least the initial link from the point of view of the end-user device is a wireless link. The physical and Medium Access Control (MAC) layers of such wireless links are defined in the 802.11 standards[1].

More specifically, 802.11 refers to a family of different standards, e.g., 802.11b, g, a, n, and ac. On the physical layer, these standards specify which frequencies within the radio spectrum and what modulation schemes the wireless link may use. Both frequencies and modulation schemes influence the network performance characteristics of wireless links.

Frequencies used by wireless links are usually part of the unlicensed radio spectrum. For this reason, anyone can operate a WiFi network on the frequencies made available by regulation within an administrative area free of charge and without an individual license. As anyone can operate a WiFi network, a wireless link may be subject to interference from other wireless links on the same frequencies. Furthermore, WiFi networks may be subject to interference from non-802.11 technology, such as Bluetooth, cordless telephones, or even microwave ovens. As such interference may distort the wireless signal, it leads to unpredictable performance variations on wireless links.

[1] See, e.g., `https://ieeexplore.ieee.org/servlet/opac?punumber=7786993`.

Actual interference on a wireless link may vary depending on the used frequency band. Most commonly, wireless links use the 2.4 GHz, the 5 GHz, and/or the 60 GHz bands. The 802.11 standard divides these frequencies into channels with a center frequency and a bandwidth. The frequencies actually used for communication depend on the used channel. In general, the 2.4 GHz band provides only a few non-overlapping channels, even if a "narrow" bandwidth of 20 MHz is used. Moreover, this frequency band is often the default used by legacy stations over 802.11b and g. The presence of such stations increases the chance of interference on the channel. While the 5 GHz band has more non-overlapping channels, wireless links on this channel have a more limited reach. Moreover, in parts of the 5 GHz band, WiFi is only the secondary user and has to give priority to other services such as radar or military radio. On the 60 GHz band, the reach of a wireless link is even more limited.

The reason for the difference in reach of wireless links on different frequency bands is related to radio propagation characteristics. A signal transmitted on a wireless link experiences a decrease in signal strength with increasing distance from the source, which is called path loss. In a line of sight through free space, usually over the air, free-space loss expresses the power ratio between two isotropic radiators [20], i.e., a sending and a receiving antenna. The Free-Space Path Loss (FSPL) formula[2] includes the wavelength of a signal λ as well as the distance between sender and receiver d:

$$FSPL = (4\pi d/\lambda)^2$$

As the wavelength of a signal is the speed of light c divided by the signal frequency f, path loss depends on the square of the frequency:

$$FSPL = (4\pi df/c)^2$$

Therefore, path loss is higher for signals on higher frequencies. Accordingly, wireless links on the 5GHz and 60 GHz spectrum, in theory, have a higher path loss occuring over a given distance than wireless links on the 2.4 GHz spectrum. However, in practice, path loss is affected by more factors than distance and frequency, e.g., depending on the environment in which the signal propagates. For example, in an outdoor scenario, weather conditions may influence path loss: As the 2.4 GHz band includes the resonance frequency of water, the presence of water affects the path loss of signals using this frequency. Moreover, path loss increases if a signal does not propagate through free space in a direct line of sight, but has to pass through materials other than air, such as a wall or a window. Here, depending on the material, parts of the signal are reflected or absorbed, which further decreases the strength of the signal passing through.

After the signal propagates from sender to receiver, whereby the signal is subject to path loss, the power level at which the signal arrives at the receiver is called the

[2]This formula is derived from the Friis transmission formula, which includes antenna characteristics such as their effective aperture.

Received Signal Strength (RSS). As receivers have a detection threshold, i.e., a minimum power level below which they can no longer decode the signal, this places a lower bound on the RSS. This limits the path loss that a wireless link can permit before signal decoding becomes infeasible, thus, limiting the range of wireless transmissions on the wireless link. Moreover, signal decoding at the receiver depends on the relation between RSS and noise level, the Signal-to-Noise-Ratio (SNR). If the noise level is high, e.g., due to interference from other stations on the same frequency, this may render signal decoding infeasible even with a high RSS because the SNR is too low. In addition to interference by other stations, interference can also be caused by signal propagation on different physical paths, e.g., after parts of the signal have been reflected by a wall. If multiple versions of the same signal arrive at the receiver at different times, they cause interference with each other, which aggravates decoding the signal. However, transmitting different signals on multiple physical paths may also be used to enhance the capacity of wireless links using multiple antennas, which is called multiple-input multiple-output (MIMO). For example, MIMO is part of modern WiFi standards such as 802.11n or 802.11ac. Still, path loss impacts the capacity of such links, as maximum sending power is regulated.

Even when sender and receiver on a wireless link are within reach of each other, so the receiver can decode a received signal, path loss affects achievable data rates. Data rates depend on the modulation scheme with which signals on wireless links are physically encoded. High data rates require the signal to use a "higher" modulation scheme, which encodes more bits within a single symbol. While more efficient, such a modulation scheme makes signal decoding more error-prone. Thus, when decoding the signal, the receiver requires a high SNR. The SNR is the difference between the RSS and the background noise at the receiver and the quantization noise of the A/D converter. Generally, a high SNR allows more efficient modulation schemes for a signal. Higher modulation leads to higher potential data rates for receivers that are closer to the sender. Furthermore, as modulation schemes are specified in the different 802.11 standards, stations which support more recent 802.11 standards support more efficient modulation schemes. In contrast, stations which only support legacy 802.11 standards use less efficient modulation schemes.

In addition to providing lower data rates, less efficient modulation schemes also occupy more airtime on the wireless channel. Airtime is relevant for the capacity, i.e., the actual throughput that a station can achieve, on the wireless link: All stations on the channel have to share the same frequencies and only one station can transmit at a time. If a legacy station does not support higher modulation schemes, it uses a lower modulation scheme. Using a lower modulation scheme, sending the same frame takes longer, so other stations are blocked from sending for a longer amount of time.

As more stations, legacy or not, occupy a channel, capacity for a single station decreases more than linearly. The reason for this lies in the way the 802.11 MAC Layer manages how multiple stations access the channel: Carrier Sense Multiple Access/ Collision Avoidance (CSMA/CA). A station that wants to transmit data first has to listen whether another transmission is already in progress. Once the station senses the channel free, it waits to send by starting its own timer. The waiting period con-

sists of an Inter Frame Spacing (IFS) and a Contention Window. The length of the IFS depends on the type of 802.11 frame to be transmitted: Short IFS for control frames such as acknowledgments, Distributed Coordination Function IFS for data frames. The Contention Window consists of a random number of backoff slots. The station picks a random number between 0 and CW_{min}, the initial maximal value of the Contention Window. It waits for this amount of backoff slots, pausing its timer if it senses that another station has started transmitting, and resuming the timer once the channel is free again. Once its timer reaches zero, the station transmits and waits for an acknowledgment. By randomizing the Contention Window, the probability of collisions is decreased. However, collisions are still possible, because two stations might still pick the same number. Moreover, frame loss can still occur, e.g., due to random signal corruption or interference by non-802.11 stations. Consequently, if the station does not receive an 802.11 acknowledgment within a certain period of time, it retransmits the frame. To reduce the risk of collision, the station performs exponential backoff: For the i-th retransmission attempt, it picks the length of the Contention Window between 0 and $2^i * CW_{min}$. Once the transmission is successful, i.e., the station receives an acknowledgment, it resets i to 0. This mechanism increases robustness to collisions and improves the chance of successful data transmission even on a highly utilized wireless link. However, this robustness comes at the cost of inflated latency and decreased capacity.

As each wireless channel is subject to the above limitations, in WiFi networks with a high number of users, it is common to deploy multiple APs on different channels. Each of these APs comprises a Basic Service Set with its own identifier, its BSSID. One or multiple BSSes can form a logical network of an EBSS. Each logical network has its own SSID, i.e., a natural-language network name, which is displayed to a user when connecting their end-user device to the WiFi network. In Figure 2.1 we call an EBSS, or logical WiFi network, a LAN.

APs of the same EBSS can be interconnected via a Distribution System, which can be wired or wireless. If an end-user device is mobile, it can roam within the same EBSS, but not across different EBSSes. This limits the reach of the logical WiFi network. If an end-user device leaves the range of the WiFi network, this may lead to disruption of communication, as the end-user device has to either associate to a different WiFi network or use a different access network technology.

In summary, network performance characteristics offered by WiFi networks can be highly variable in terms of latency and capacity. This depends, for example, on the number of stations connected to the same BSS, and on their level of activity. With more stations and more transmissions, the likelihood of collision increases, which leads to exponential backoff in the contention window.

2.1.1.2 Uplinks

While wireless links may limit data rates available to end-user devices and incur latencies, they are not always the performance bottleneck [3]. Uplinks are another

possible location of the bottleneck within an access network. Here, we consider access networks as divided into a core network, a distribution network, the "last mile", and the LAN. In this case, the uplink covers the last mile. In the context of a WiFi network, we define the uplink as the logical link between the Local Area Network and the ISP network, see Figure 2.1. This logical link might consist of multiple physical links. Often, those are the links between the Customer Premises Equipment (CPE), e.g., a router or switch deployed on the end-user site, and the ISP network. The performance of an access network is impacted both by the capabilities of the CPE and by the network performance characteristics of the technology used on the uplink. Next, we describe different technologies which are commonly used for uplinks.

One such uplink technology is Digital Subscriber Line (DSL). DSL utilizes twisted-pair copper cables to provide a dedicated physical link between a DSL modem and a DSL Access Multiplexer (DSLAM). While the DSL modem is often part of a LAN on the end-user site, the DSLAM is often located on premises controlled by or accessible to the ISP. For example, this can be a street cabinet, a distribution center, or the basement of a building. The DSLAM encodes and decodes data between the access modem and upstream networks and it aggregates signals from multiple DSL modems, i.e., multiple end-user sites. Maximum available data rates vary based on the DSL standard[3]. With Asymmetric DSL (ADSL), data rates differ on the upstream and downstream. Maximum data rates range from up to 12 Mbit/s downstream and 1.8 Mbit/s upstream, which is ADSL 1, to 24 Mbit/s downstream and 3.5 Mbit/s upstream, which is ADSL 2+. With Very High Speed DSL (VDSL), data rates increase to 52 Mbit/s downstream and 16 Mbit/s upstream. Finally, with VDSL2, data rates may exceed 100 Mbit/s in both upstream and downstream directions. In practice, achievable data rates are limited by the distance between modem and DSLAM. This is because the signal is attenuated on the cable, i.e., its SNR decreases with distance. Furthermore, a signal may be distorted due to crosstalk between multiple cables to the same DSLAM. Thus, data rates are limited by the distance between the DSLAM and the DSL modem. Furthermore, a DSL uplink may incur additional latency due to Interleaving: One endpoint of the DSL link may interleave data frames before splitting them onto different subcarriers on the physical link when transmitting them. Because errors on a physical link often occur in bursts, this technique spreads the error across multiple data frames, which aids Forward Error Correction. While this increases robustness to line noise, it also adds latency.

Another common uplink technology uses coaxial cables to connect a LAN to a cable operator network. Here, a cable modem within the LAN connects to a Cable Modem Termination System (CMTS), which receives signals and converts them to Ethernet. This technology is standardized in the Data Over Cable Service Interface Specification (DOCSIS), with the most recent version being DOCSIS 3.1. Data rates, again, depend on the version of the standard and the number of channels, i.e., frequencies, which can be used for up- and downlink. With DOCSIS 2.0, data rates of up to 42.88 Mbit/s are specified on the downlink and 30.72 Mbit/s on the uplink. DOCSIS

[3]DSL is standardized by different organizations, e.g., ANSI in the US, ETSI in Europe, or the ITU-T, which is international.

3.0 can achieve more than 1 Gbit/s on the downlink and 200 Mbit/s on the uplink by bundling multiple channels and DOCSIS 3.1 increases maximum data rates even more, providing up to 10 Gbit/s in both directions. However, data rates for individual customers may be lower due to rate limitation based on the service plan of the customer implemented at the CMTS. Additionally, coaxial cables are a shared medium. Therefore, in practice, data rates may vary based on the network utilization of all users connected to the same CMTS.

In some enterprise or campus networks, the uplink may be realized using Ethernet over a twisted pair copper cable or fiber. With fiber, data is transmitted through fiber-optic glass strands using laser pulses and amplitude modulation. Fiber can provide high data rates, with a capacity of, e.g., 560 Tbits/s for the band between wavelengths of 1300 and 1620 nm [21]. Data rates in practice often depend on the terminal equipment rather than the fiber itself. While providers often advertise high speeds such as 1 Gbit/s, data rates in practice may be lower due to host issues [22]. Moreover, deploying fiber is very costly. In access networks, fiber network architectures are named after how far the ISP deploys fiber along the path to the end-user device. For example, if fiber is deployed up to the LAN, i.e, including the uplink to the end-user site, this corresponds to Fiber To The Home (FTTH). If fiber is deployed up until the building that the LAN is located in, this is called Fiber To The Building (FTTB). As an alternative to deploying fiber for the uplink, a network operator can increase the capacity of its cable or DSL network by deploying Fiber to the Curb (FTTC), i.e., up to the street cabinet or distribution center. Often, instead of deploying a dedicated fiber for each customer, ISPs deploy a shared fiber which serves multiple sites along a part of the path. In this case, the signal has to be split at some point to be distributed to the individual sites. This split can be realized as an Active Optical Network (AON), in which an electrically powered switch or router forwards the packets to and from each site. Alternatively, with a Passive Optical Network (PON), an unpowered optical splitter is used to distribute the signal, which is more cost-effective. Due to its high capacity, fiber is expected to be the technology of choice for aggregation within access networks and for providing backhaul to core networks. Common maximum data rates are 10 Gbit/s in the backhaul and 1 Gbit/s to customer routers. For fiber, data rates are usually limited by the terminal equipment instead of the fiber itself. Therefore, ISPs are able to increase data rates by upgrading the terminal equipment.

Finally, the uplink can be realized using satellite communications. Here, the ISP deploys a constellation of satellites which send and receive data from ground stations and relay data between different satellites. As such satellite communication systems may provide broad or even global coverage, they can be used to connect remote places to the Internet, where no other uplink technology is available. Network performance characteristics of satellite communication systems depend on the distance between the satellites and ground stations. Historically, such systems were often deployed in geostationary orbit (GEO), in which the satellite appears stationary in the same position to the ground station. While this has the advantage that the antenna of the ground station does not have to track the satellite, the long distance between satellite and ground station results in long latencies. More recently, communication satellites

are using Medium Earth Orbit (MEO) and Lower Earth Orbit (LEO) as well. Here, satellites orbit the earth at a higher speed and do not remain in the same position from the point of view of the ground station. Therefore, more satellites are required to provide continuous connectivity. However, due to the decreased distance, latencies are shorter. Moreover, with decreased distances, path loss decreases, which results in higher achievable data rates and/or lower required signal strength at senders. Satellite communication uses high frequencies and may achieve high downstream capacity. For example, satellite systems on the K_u band between 12 and 18 GHz may achieve a downstream capacity of 506 Mbit/s[4].

While uplink technologies are diverse and deployed with increasing coverage, not all uplink technologies are available everywhere. Broadband coverage reports [11, 12] provide data on the availability and data rates of different uplink technologies across the world. In general, DSL is available in many parts of the world. This is because telephone network operators have deployed twisted-pair copper cables to connect homes to the telephony network. Then, the same operators can repurpose their existing infrastructure to provide Internet connectivity. In the same way, cables originally deployed to provide cable television can be repurposed by cable network operators. While DSL is widely available throughout Europe [13], cable has the largest market share in the US. Additionally, cable is available to 80% or more of the population in Canada, Belgium, Korea, the Netherlands, and Switzerland, according to the 2013 OECD communications outlook [11]. However, the availability of cable is below 20% for other countries, such as France, Italy, Mexico, New Zealand, Spain, and Turkey [11]. In general, economic factors play a large role in deploying access networks. In addition to equipment costs, deploying communication links may be costly and an ISP may not have a financial incentive to, e.g., serve rural regions with high-speed Internet access. Therefore, even with the advance of access technologies, deployment may vary depending on the locality and still pose a challenge.

Even with increased deployment of access network technologies, as application traffic increases as well, the actual network performance may vary depending on the number of concurrent users and/or applications. Therefore, the performance bottleneck of an application may be within the access network, i.e., within the LAN or the uplink. Determining whether the bottleneck is within the LAN or the uplink is challenging [3]. If the bottleneck is within the core network, paths through different access networks may traverse different bottlenecks.

2.1.2 Cellular Networks

In many parts of the world, end-user devices can connect to cellular mobile networks to access the Internet. Figure 2.2 shows the generic architecture of a cellular network. The end-user device connects to a cell by associating to a base station provided by a mobile network operator, the ISP. Similar to WiFi networks, the first link from the point of view of the end-user device is wireless. The cellular base station is then

[4]See `https://www.newtec.eu/frontend/files/userfiles/files/App%20Note%20IP%20Trunking%20Rev01.pdf`, last accessed 04. June 2019.

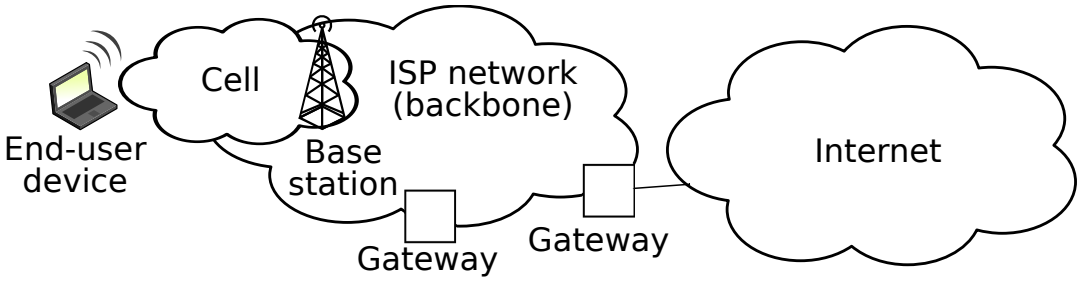


Figure 2.2: Architecture of a cellular network.

connected to the mobile ISP's network, i.e., the backbone, over a wired link. In the backbone, the base station is usually connected to a gateway that provides data services. Usually, this gateway connects to another gateway, which, in turn, links the mobile ISP's network to the rest of the Internet.

In contrast to WiFi, the cellular base station is more tightly integrated into the operator network than a WiFi AP is integrated into an ISP network. Moreover, a single cellular tower may cover multiple cells by using directional antennas, which are sending and receiving on different frequencies in different directions. As the link between base station and operator network is usually realized via fiber, this makes this link less likely to be the bottleneck. However, this link may still be the bottleneck in some cases, as fiber may not always be deployed for economical reasons [23]. Alternatively, capacity bottlenecks may be located on the wireless link or within the wired network between base station and remote host [5]. Moreover, cellular networks may present a latency bottleneck, for example, due the power state transitions within cellular devices' radios [24] and signaling overhead within the backbone [25].

The wireless link within the cellular network is subject to similar limitations as a wireless link within a WiFi network. For example, it experiences path loss, see Section 2.1.1.1. Different from WiFi, a cellular wireless link usually operates within licensed radio spectrum. Only the network operator who has a license to a certain frequency band can operate a base station using the frequencies in this band. Thus, there is usually no interference by other cellular networks on the same radio frequency. Consequently, the capacity on the same channel does not have to be shared by cellular devices in the same way as in WiFi. Rather, the channel can be divided into different subcarriers, e.g., based on frequency and time. A cellular standard may define a frame structure of time slots. These time slots determine, e.g., at what times the base station sends control information to cellular devices, and at what times cellular devices may send data. Furthermore, the base station can control what time slots and frequencies to make available to which cellular device. In this way, the network controls how much capacity is allocated to each connected cellular device. This way of allocating resources enables cellular networks to provide a more reliable upstream and downstream capacity than WiFi networks.

Similar to WiFi, the radio modulation technology used for data transmission depends on the used standard. Furthermore, each standard describes the architecture of the

backbone network. It specifies the components that the backbone contains along with their functionality. The 3rd Generation Partnership Project (3GPP) defines cellular network standards in multiple generations.

In cellular networks, the earliest standard to provide data transmission services is General Packet Radio Service (GPRS). GPRS is part of the second generation (2G) of cellular network standards, Global System for Mobile Communications (GSM), and is often called 2.5G. It re-uses the existing cellular infrastructure designed for voice communications and provides very low data rates of between 56 and 114 kbit/s. Within 2G, Enhanced Data Rates for GSM Evolution (EDGE) increases data rates to up to 236 kbit/s by using more efficient modulation schemes and combining multiple time slots.

The third generation (3G) of cellular standards introduces separate network components for voice, i.e., circuit switched services, and data, i.e., packet switched services. While 3G radio technology used on the wireless link is completely different from 2G, higher protocol layers reuse functionality from 2G. Data rates depend on the standard within 3G. The original Universal Mobile Telecommunications System (UMTS) provides nominal data rates of 384 kbit/s downstream and 128 kbit/s upstream. High Speed Packet Access (HSPA), a more advanced release, advertises with a capacity of up to 84 MBit/s downstream and 22 MBit/s upstream.

Long Term Evolution (LTE) and its advanced releases represent the fourth generation (4G) of cellular network standards. Again, 4G provides a different physical layer on its wireless links than the previous generation. The 4G physical layer allocates resource blocks which are arranged in a frame structure over time. 4G networks are based on IP, which greatly simplifies the protocol stack compared to 3G. Similar to the previous generations, there are different releases with different theoretically available data rates.

Finally, the fifth generation (5G) of mobile cellular networks currently is still in development. 5G envisions to provide connectivity for a diverse set of applications with different performance requirements in terms of latency, data rate, and energy efficiency. For different use cases, there exist different traffic profiles, each with its own parameter set. For example, to deliver high bitrate video streams to end-user devices in urban areas, 5G may use the 36 GHz spectrum to deliver even higher data rates and lower latencies than LTE. For other use cases, such as Internet of Things (IoT), the required data rates are lower, but such cellular devices may need to send data over a longer range with higher energy efficiency. To achieve the goal of providing the best possible performance based on the use case and application requirements, 5G virtualizes network resources and provides "slices" optimized for a specific service. For example, 5G can offer connectivity using different frequency bands: While low frequency bands provides a longer range, high frequency bands provide higher upstream and downstream capacities. In addition to using licensed frequency bands, 5G may also coexist with WiFi on the unlicensed spectrum, making network slices available on the same frequencies. Moreover, 5G aims to reduce operating costs using Network Function Virtualization (NFV): It decouples functions performed within the

network from the hardware they are implemented on, running functions such as Network Address Translation (NAT), firewalls, or intrusion detection on virtual machines instead.

Over the generations, cellular networks provide greatly varying network performance characteristics. Historically, cellular networks have offered limited upstream and downstream capacities and long latencies. However, the general tendency is that performance is improving.

2.1.3 Access Network Performance

When multiple networks are available, there is the question which of these networks to use. To answer this question, it is crucial to understand the performance provided by WiFi and cellular networks in practice.

Sommers et al. [1] compare cellular and WiFi performance when both are available to end-user devices. In their study, they observe end-user devices in 15 metropolitan areas, of which nine are located in the US, three are located in Europe, and three in the Asia-Pacific region. The authors evaluate data gathered over the course of 15 weeks between February 2011 and June 2011. To measure latency and capacity, they transfer files of fixed sizes and perform pings over HyperText Transfer Protocol (HTTP) between mobile end-user devices and Speedtest servers. According to their measurements, in 2011, WiFi generally provides a better absolute up- and downstream capacity and a higher degree of consistency in performance. Also, WiFi provides a shorter absolute latency, but cellular has a higher consistency in latency. Throughput and latency vary widely between network technologies and providers. Finally, they find that overall consistency of performance in wireless networks is much lower than in wired broadband networks.

Deng et al. [2] study the performance of WiFi and cellular LTE networks using a dedicated mobile app between September 2013 and May 2014. They compare data from 750 users in 16 different countries across the world. In cases where end-user devices have both WiFi and cellular available, via both networks, they upload and download a large file over Transmission Control Protocol (TCP) from a dedicated server in the US to measure capacity. Additionally, they estimate Round Trip Time (RTT) using pings between the end-user device and the server. Contrary to the assumption that WiFi networks outperform cellular, which was common at the time, their study shows that LTE outperforms WiFi 40% of the time, with a potential capacity difference of more than 10 Mbit/s in either up- or downstream. In 20% of cases, even ping RTTs are lower for LTE than for WiFi.

While access networks have evolved since 2014, unfortunately, there are no more recent large-scale academic performance studies available. However, industry reports [26] indicate an increase in upstream and downstream capacity both for fixed (e.g., WiFi with a fixed-line uplink) and mobile (e.g., LTE) networks, but still vary greatly among geographic locations and network providers. Here, fixed networks provide a higher capacity than cellular in most geographic regions. Note that such

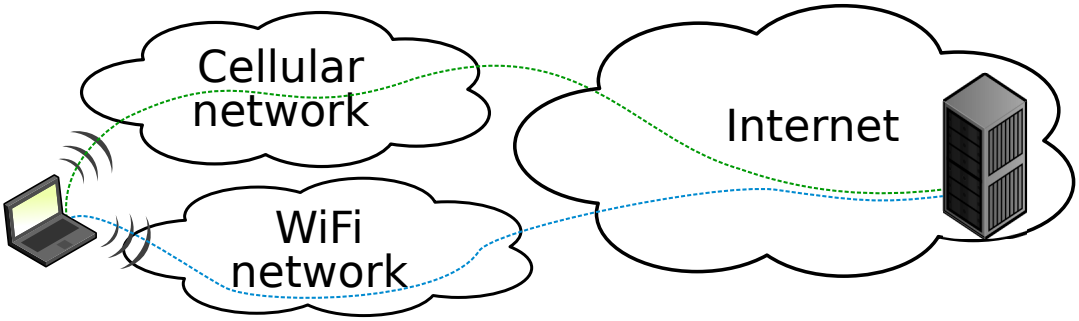


Figure 2.3: Scenario where multiple access networks are available.

reports are not directly comparable with the studies by Related Work because they do not restrict data points to scenarios in which both WiFi and cellular are available to the same end-user device at the same time.

In summary, both performance studies show that WiFi and cellular performance for end-user devices are highly variable and that recently, there is often not one obvious "better" access network. While cellular performance is usually worse than WiFi in early studies, this changes with the introduction of LTE. Nowadays, WiFi and cellular networks are likely to complement each other. Our work builds on these observations by monitoring network performance characteristics on an end-user device and using this information as input for access network selection.

2.2 Using Multiple Access Networks

Increasingly, multiple access networks are available to an end-user device at the same time. In this section, we explore scenarios in which multiple access networks are available and define the scenario that this thesis focuses on.

Given the increased availability of multiple access networks and their varying network performance characteristics, related work has explored how to take advantage of them. While mobile data offloading selects between access networks, multipath transport protocols use the available networks in parallel.

2.2.1 Multiple Access Network Scenario

We define a multiple access network scenario as a situation in which an end-user device is able to connect to a remote host via multiple access networks at the same time, as shown in Figure 2.3. An end-user device is a host which a human uses to communicate with others or to access content using applications such as Web browsing or video streaming. Common examples of end-user devices are desktop PCs, laptops, smartphones, and tablets. End-user devices are often mobile.

An access network is a network which connects an end-user device to the Internet. This thesis focuses on two types of access networks: WiFi networks as defined in Section 2.1.1 and cellular networks as defined in Section 2.1.2. As shown in Figure 2.3, a multiple access network scenario implies that the end-user device can reach a remote server through each access network, i.e., using different paths. These paths may be either completely disjoint or share one or more hops.

As such, a multiple access network scenario is a special case of a multihomed host [27]. A multihomed host is a host that has multiple IP addresses, e.g., if it is connected via multiple network interfaces to the same or different networks at the same time. In a multiple access network scenario, an end-user device is multihomed. Hereby, the end-user device is connected via its network interfaces to multiple different access networks. From each connected access network, the end-user device receives one or multiple IP addresses and additional configuration, such as a default gateway to reach the Internet.

A multiple access network scenario requires that different access networks, e.g., a WiFi and a cellular network, are deployed in such a way they are available in the same geographic location. These networks may be operated by the same provider or by different providers. Furthermore, the end-user device needs to be connected to these multiple networks simultaneously. This requires that the end-user device has multiple network interfaces built in or can use them through additional hardware, e.g., a cellular modem implemented within a USB stick[5]. Furthermore, the end-user device has to support multihoming to some degree, e.g., it must be able to establish connections over multiple local network interfaces at the same time. Such basic support is usually available on contemporary end-user devices, even when more sophisticated technologies to leverage multiple access networks are not supported. See Section 2.3 for a detailed discussion of multihoming support on end-user devices.

As end-user devices often support multihoming, a multiple access network scenario can occur in many cases in which multiple access networks are deployed in the same location. For example, this is the case in many end-user homes or in public places such as cafés. In such places, often, there is both a WiFi network as well as a cellular network available. Here, either cellular or WiFi networks may sometimes provide sub-optimal performance, e.g., due to network overload on the WiFi, limitations on the uplink, or an overloaded cellular network. Such degraded performance may motivate moving from WiFi to cellular or vice versa. For example, limited capacity on cellular networks is the main motivation of offloading, see Section 2.2.2. In some cases, the uplink in the WiFi network limits available capacity to a point where it becomes reasonable to provide additional capacity using cellular. For example, such capacity limitations on the uplink motivate ISPs to offer "hybrid connectivity" supplemented by LTE, e.g., in some rural areas. While this use case is not covered by our multiple access network scenario, it highlights that the need to aggregate multiple access networks exists in practice.

[5] Additional nontechnical requirements, such as an active subscription to a mobile network provider, are out of scope for this thesis.

In addition to home networks or other small LANs, multiple access networks may be available in larger networks, such as enterprise networks, campus networks, or urban areas with public WiFi APs. Here, uplink capacity limitations may still play a role in some cases. Moreover, if end-user devices move within such networks, this may result in changing network performance characteristics over time. Here, which network provides the best connectivity may change over time, or a network may become unavailable due to the end-user device moving out of range.

In all these cases, making a good choice between the available access networks or using multiple access networks at the same time may be beneficial for an end-user device.

2.2.2 Distributing Traffic Using Mobile Offloading

Networks vary not only in terms of performance but also in terms of load and cost to operate. If a mobile network is heavily utilized, a network operator may choose to offload traffic from their cellular networks to WiFi. Their goal is to avoid degrading Quality of Experience (QoE) while handling an increased number of users with an increased network capacity demand.

Aijaz et al. [28] survey several approaches to offloading that can be deployed by mobile network operators in cellular 3G and 4G networks. For offloading via WiFi, they contrast unmanaged data offloading with managed data offloading. In the former case, end-user devices transparently move traffic onto WiFi and completely bypass cellular networks. In the latter case, operators places gateways within WiFi networks to maintain control over subscribers. With integrated data offloading, operators further increase the degree of cooperation between WiFi and cellular networks by transferring data between end-user devices and cellular core networks via WiFi. The benefits for operators to use a more integrated solution are increased control of subscribers and the ability to deliver content from the core network via WiFi. The integration of WiFi into cellular networks is standardized both in the 3GPP, via technologies such as I-WLAN, and in the Internet Engineering Task Force (IETF), via IP flow mobility for Proxy Mobile IPv6 (IFOM).

Mallawi et al. [29] provide a comprehensive survey of offloading techniques. They describe several approaches: First, Device-to-Device communication, where end-user devices serve as content caches for other end-user devices, potentially using wireless interfaces other than cellular, e.g., WiFi or bluetooth. Second, they describe small cells, whereby an operator deploys additional cellular base stations with reduced coverage, i.e., femtocells. Then, cellular devices can associate to this smaller cell instead of congesting the larger cell. Third, WiFi Alternative Path, whereby operators deploy WiFi APs. For the latter, providers may need mechanisms to enable internetworking between WiFi and cellular, e.g., Mobile IP (MIP) and Proxy MIP. The survey also explores strategies to determine what traffic to offload: For example, operators may choose to offload traffic only for certain users or only traffic of specific types of service such as Voice over IP (VoIP) or video streaming. Furthermore, in scenarios

where mobility solutions are not deployed, operators may choose to not offload sessions for which disruption of the connection results in a negative impact on QoE. In the survey, the authors recommend to offload live streaming, buffered streaming, and low-priority voice traffic to WiFi. They then discuss where the offloading decision should be made: On end-user devices or in the core network. On end-user devices, multiple aspects have to be taken into account, such as operator objectives, end-user needs, service requirements, access point limitations, and others. These objectives may be contradictory, e.g., when balancing the best connectivity with energy saving. In the core, any entity making offloading decisions has to gather information from end-user devices, e.g., the user's subscription, network capabilities, network conditions, and end-user device characteristics. The authors of the survey mention several possible sources of information, but indicate that further work is required.

As the work in this thesis does not need any integration between WiFi and cellular networks and does not interact with the network infrastructure, it is most similar to WiFi Alternative Path as an "Over-The-Top" solution. In our approach, offloading decisions are made on end-user devices and primarily consider end-user needs for good application performance.

Different strategies to determine what traffic to offload are explored by Wiethölter et al. [30]. They compare algorithms to determine for which mobile subscribers to shift traffic to cellular or WiFi networks. A common offloading strategy is based on RSS with random selection of users. As an alternative, the authors propose a "Cost-Function Approach" based on clients' throughput efficiency and incurred WiFi channel load. By simulating scenarios with mobile networks of different cell sizes and mobile users with Voice over IP and FTP traffic, the authors find that their approach outperforms other selection schemes especially in smaller cells. However, the RSS-based approach provides slightly higher goodput in scenarios with larger cell sizes, which the authors deem more unrealistic.

While offloading focuses on shifting traffic from cellular to WiFi, Rossi et al. [31] propose 3G Onloading, which shifts traffic from WiFi to cellular. The authors motivate this approach by the observation that WiFi network capacity is sometimes limited by the uplink capacity behind a WiFi AP, e.g., an ADSL line with limited bandwidth. Onloading can be realized either as a network-integrated solution for a single provider, for multiple providers, or without any network integration, i.e., as an OTT solution. Onloading bases its offloading decisions on estimations of "leftover bandwidth" based on diurnal traffic patterns, i.e., cellular and Wifi network usage peaks at different times of day. In cases where a WiFi network has limited capacity, Onloading uses an available cellular network to provide more capacity at the beginning of a transfer. The cellular network is only used in cases where the user still has data volume available. Both in their simulations and using their prototype in different European residential locations, the authors find that Onloading can improve upload and download times for video on demand and reduce video pre-buffering times.

Gadgil et al [32] evaluate the performance of IFOM in LTE and WiFi networks. They simulate an urban deployment of cellular networks and WiFi APs and consider VoIP, video, and FTP traffic. They keep real-time traffic such as VoIP and video on cellular

networks while offloading non-real time flows. For example, they shift Web traffic and FTP flows to WiFi when it provides a better link quality, i.e., a higher capacity for an individual end-user device. In this case, IFOM-enabled end-user devices benefit from a higher capacity while keeping latencies for real-time traffic low even under high load.

A solution with limited cooperation between network operator and end-user device is proposed by Gerasimenko et al. [33]. Here, the network provides information about its own load, and end-user devices decide to offload traffic based on either a baseline RSS-based threshold algorithm or a load-aware algorithm. In the latter algorithm, end-user devices use the predicted network capacity as advertised by the network, exclude APs with low RSS, and combine their capacity estimates with actually observed capacity. In their simulation, the authors find that the load-aware algorithm they propose provides higher capacity. However, they have to limit excessive switching between different access networks using a hysteresis mechanism.

In contrast to network-integrated solutions, Wamser et al. [34] evaluate an OTT solution. Here, a gateway outside of the access network distributes traffic across multiple access networks. As this gateway can monitor application quality metrics, e.g., for video streaming, it can distribute traffic across access networks based on application needs. The proposed algorithms for traffic distribution are based on the required capacity for a video stream and on an estimate of the corresponding video playout buffer at the client. In their testbed measurements, the authors find that the proposed algorithms improve QoE for end users in cases where one network cannot provide sufficient capacity. Furthermore, their buffer-based algorithms greatly reduce energy consumption and cellular data volume consumption.

Another solution which does not require network integration is Delphi [35], which proposes a transport-layer module on mobile end-user devices for access network selection. They identify objectives for different applications, such as minimizing delay for VoIP, maximizing capacity for video streaming applications and background traffic, or minimizing energy consumption for applications that periodically need to synchronize. They then apply local learning to network performance characteristics such as RSS, RTT, WiFi APs, packet loss rate, network load, and observed capacity, and share this information between nodes in the same WiFi network. Finally, they select the most suitable network according to high capacity.

In summary, existing offloading solutions focus on network-integrated or OTT solutions with in-network support, e.g., by application service providers. Informed Access Network Selection (IANS) does not require any support within the network or by remote hosts. Instead, IANS gathers information and makes access network selection decisions only on end-user devices while taking application information into account in order to improve end-user QoE. Similar to the above studies, our work leverages the observation that different kinds of traffic require different access network selection strategies.

2.2.3 Multipath Protocols

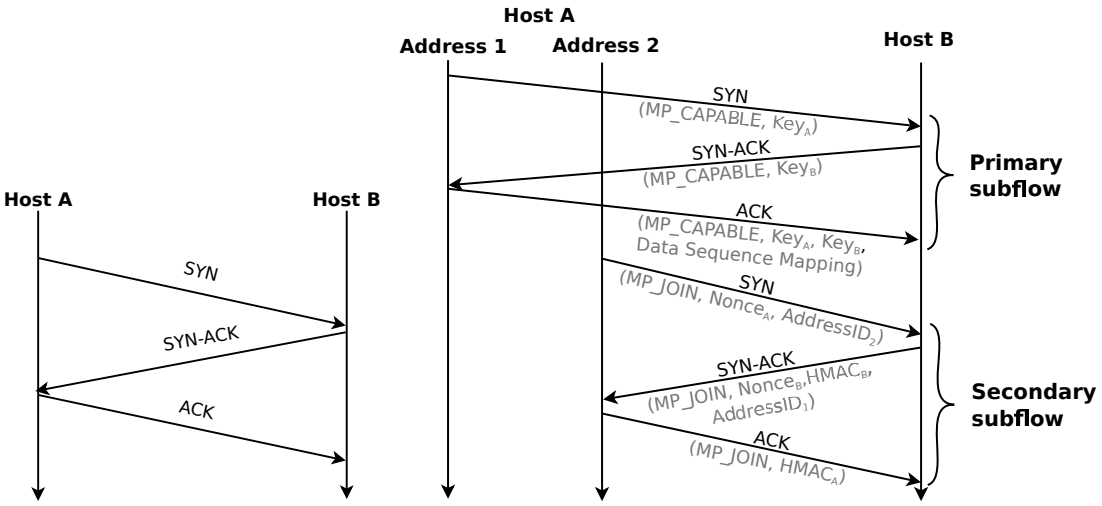
In addition to offloading traffic from WiFi to cellular or vice versa, it is possible to combine both networks using multipath transport protocols. For example, MPTCP allows a single connection to utilize multiple access networks. We first provide details on how MPTCP achieves this and then present MPTCP design considerations and performance studies. Finally, we discuss multipath protocols other than MPTCP.

2.2.3.1 MPTCP Basics

MPTCP [16] adds multipath capabilities to TCP. By itself, TCP only allows opening a connection between a single pair of local and remote IP address, which the connection is then bound to. Consequently, with TCP, it is not possible to open a single connection over multiple networks at once. Furthermore, if one endpoint of a TCP connection moves to another network, this changes the IP address that the connection is bound to, so the connection breaks.

MPTCP enables a single transport-layer connection to be established between multiple pairs of local and remote IP addresses and/or ports. If multiple local IP addresses correspond to multiple local network interfaces, which are connected to multiple access networks, the connection can use these networks simultaneously. Hereby, each pair of local and remote IP address corresponds to one subflow. After establishing an MPTCP connection with multiple subflows, the host can send and receive data via all subflows, and, thus, combine the capacities of multiple networks. In addition, MPTCP may increase resilience to connection disruption. If a local IP address changes or a network interface loses its connectivity, this only impacts one subflow instead of the entire connection. Then, the host can still use other subflows on the same connection and also establish a new subflow for the same connection over a different local network interface.

Figure 2.4 shows how a TCP connection and an MPTCP connection are established between two hosts A and B. In a TCP handshake, see Figure 2.4a, host A first sends a TCP segment with the SYN flag set. Host B replies with a segment setting both SYN and ACK. Finally, host A sends an ACK, at which point the TCP connection is established. To establish an MPTCP connection, host A first has to establish a primary subflow as shown in Figure 2.4b. Similar to a TCP handshake, host A first sends a TCP segment with the SYN flag set. However, in the TCP header of this segment, host A also sets a TCP option named MP_CAPABLE and includes its key Key_A. Assuming host B supports MPTCP, when host B replies with a SYN-ACK, it also sets the MP_CAPABLE option in the TCP header, which contains its key Key_B. When sending the final ACK to establish the subflow, host A repeats both keys and provides an initial Data Sequence Mapping. A Data Sequence Mapping matches sequence numbers of individual subflows to Data Sequence Numbers (DSNs) for an entire MPTCP connection. Therefore, DSNs enable an MPTCP receiver to reorder data received across different subflows. Once the primary subflow is established, either host can open additional subflows, e.g., using a different local IP address. In



(a) TCP handshake. (b) MPTCP handshake with primary and secondary subflow.

Figure 2.4: TCP and MPTCP handshakes.

Figure 2.4b, host A establishes a secondary subflow from IP address 2 by sending a SYN with the MP_JOIN option. This option also contains a token generated from Key_B, a nonce, and A's own address ID. Note that MPTCP utilizes its own address IDs instead of using the IP address from the IP header. The reason is that the IP address in the header of a received packet may not be the same IP address as originally set by the remote host. Rather, the address may be changed along the path due to NAT. When host B replies with a SYN-ACK, it adds the MP_JOIN option specifying its own nonce, address ID, and Hash-based Message Authentication Code (HMAC). In the final ACK establishing the secondary subflow, host A sends its own HMAC. To compute their HMACs, both hosts use Key_A and Key_B from the handshake of the primary subflow and using both nonces from the handshake of the secondary subflow. The benefits of the HMAC are threefold: They confirm that the parties establishing the secondary subflow are the same as the parties involved in establishing the primary subflow. They provide verification that a peer can receive MPTCP segments at a new address before using it. Finally, the HMACs protect from replay attacks for establishing the secondary subflow.

On each host, an entity called the path manager decides over which interface to open the primary subflow, and whether and when to open secondary subflows. Once multiple subflows are open, a scheduler decides how many bytes to send over each subflow and whether to replicate any data over multiple subflows.

MPTCP is designed with several considerations in mind, as described by Raiciu et al. [17]: Application compatibility, network compatibility, and fairness. To be compatible with existing applications, MPTCP provides the same service as TCP: It presents a reliable in-order bytestream to applications. Thus, applications do not need to be aware of MPTCP. Rather, once MPTCP is supported by the OS, it can be enabled for all applications which use TCP without requiring any modifications

to the applications. However, note that MPTCP is not yet available in all OSes. See Section 2.3 for a discussion of multipath support on hosts in general and MPTCP support in particular.

For compatibility with existing networks, MPTCP is designed as an extension to TCP. One reason for this design decision is a change in the Internet architecture: Traditionally, transport protocols such as TCP or MPTCP were implemented only at communication endpoints, i.e., on hosts. Intermediate nodes within networks only implemented protocols up until the network layer and were agnostic to transport protocols. However, over time, actually deployed networks have deviated from this traditional Internet architecture. In today's Internet, middleboxes are aware of higher layers than the network layer, e.g., the transport layer. For example, firewalls may filter traffic based on ports within transport protocol headers. Middleboxes perfoming NAT may change IP addresses as well as ports within an ongoing connection. Some proxies do not just modify, but even terminate transport-layer connections. MPTCP makes an effort to work with this reality of deployed networks. It attempts to increase the chance to traverse middleboxes by resembling TCP as much as possible on the wire. However, middleboxes on a path may still be incompatible with MPTCP, e.g., remove MPTCP options from the TCP header. In such cases, an MPTCP connection falls back to using single-path TCP.

Finally, another design goal for MPTCP is fairness: On a bottleneck shared by multiple MPTCP subflows, the entire MPTCP connection should not take more capacity than its fair share. This means that MPTCP should behave similar to a single-path TCP flow at a shared bottleneck. MPTCP meets this requirement by using coupled congestion control [36].

2.2.3.2 MPTCP Performance

The performance goals of MPTCP are to increase capacity and to improve resilience to network failure. In order to assess whether MPTCP meets these goals, there are several studies on MPTCP performance.

After designing MPTCP, Raiciu et al. [17] evaluate the performance of MPTCP over WiFi and 3G networks for a benchmark of 100 emulated clients, which request files of different sizes from a server. In their study, the authors observe a tradeoff between small and large files due to the initial costs to set up an additional subflow: For files smaller than 30 KB, MPTCP decreases performance due to the overhead of the second subflow. For files larger than 100 KB, MPTCP consistently improves performance compared to TCP.

Chen et al. [18] evaluate MPTCP performance over WiFi home networks and several cellular providers. For small files, they find that using a single path over WiFi is best. For larger files, cellular often performs better due to lower loss, and MPTCP is able to successfully offload traffic. They do not observe a penalty for smaller files, as in such cases, the subflow over the cellular network is not even established.

In their study of WiFi and LTE performance, Deng et al. [2] measure MPTCP performance compared to single-path TCP with primary subflows opened over either WiFi or LTE. They find that MPTCP may penalize short flows and that selecting the correct network for the primary subflow is critical. However, they observe that applications whose performance is dominated by long flows can gain from using MPTCP if they pick the more suitable network for the primary subflow.

Han et al. [19] study Web performance using HTTP and SPDY, a precursor to Quick UDP Internet Connection (QUIC) [37, 38], in a proxy-based setup over WiFi and LTE. They find that MPTCP improves performance for SPDY in all cases and for HTTP in most cases. Furthermore, they observe that Web performance is usually dominated by the path with lower latency, in cases where multiple paths provide similar bandwidth and loss.

Paasch et al. [39] design and implemented a framework to compare different MPTCP schedulers. They compare the following schedulers: First, they consider Round-Robin, which distributes data across all subflows, saturating each subflow by filling its congestion window. Second, they look at Lowest-RTT-first, which sends data over the subflow with the lowest estimated RTT first, filling its congestion window and then proceeding to the subflow with the next higher RTT. Third, they examine the Retransmission and Penalization scheduler, which aims to compensate for delay difference and reduce head-of-line blocking by opportunistically replicating segments over subflows with available capacity. Finally, they study the Bufferbloat Mitigation scheduler, which limits the amount of data transmitted on each subflow to avoid inflating RTTs. The authors find that the latter two schedulers show benefits in environments with a high Bandwidth-Delay Product (BDP). Furthermore, they observe that the lowest-RTT-first scheduler reduces latency for applications.

To improve MPTCP performance over paths with asymmetric conditions, Related Work proposes several MPTCP schedulers: STMS [40] reduces burst transmission for asymmetric paths such as WiFi and LTE. DEMS [41] decouples paths for chunk delivery and leverages a small amount of redundancy on subflows. ECF [42] prevents under-utilization of the lower RTT path by taking RTTs and congestion windows on the connection level and on the subflow level into account. RAVEN [43] improves RTT estimates by penalizing older samples and selectively leverages redundant transmissions on multiple paths. MP-DASH [44] schedules video segments according to user preferences, segment size, and delivery deadline, with the aid of a video adapter as an interface to the Adaptive Bit-Rate algorithm (ABR).

2.2.3.3 Other Multipath Protocols

An early example of a transport protocol which supports multihoming is Stream Control Transmission Protocol (SCTP) [45]. Similar to MPTCP, SCTP allows a single connection to be associated with multiple local or remote IP addresses. However, SCTP does not allow to aggregate capacity across access networks. Instead, it only

utilizes the primary path over one network and uses the secondary path over another network as fallback.

To aggregate capacity for constant bit rate media streams, Singh et al. add multipath support to the Real-Time Transport Protocol (RTP) [46]. The authors split constant bit rate media streams across multiple paths based on path characteristics in order to achieve higher bit rate and an increased robustness against disruptions.

Multi-Source Multipath HTTP (mHTTP) [47] distributes Web resource loads across multiple TCP connections over different access networks and to different endpoints, e.g., Content Delivery Network (CDN) nodes serving identical content. mHTTP uses HTTP range requests to fetch chunks of a resource over different TCP connections using HTTP/1.1. To reduce resource load time, a scheduler computes chunk sizes to be fetched over each connection based on capacity estimates. The mHTTP prototype implementation shows similar load time reductions as MPTCP for large resources, but no benefits for small resources.

Multipath HTTP/2 (MP-H2) [48] further develops the idea of distributing individual resource loads across multiple access networks using HTTP range requests. As the authors build on HTTP/2, they utilize HTTP/2 streams over the same TCP connection instead of multiple TCP connections to load parts of a resource. Moreover, they design an advanced scheduler which bases its chunk size calculations on capacity estimates as well as RTT. To gather RTT estimates, the authors leverage HTTP/2 features. Moreover, they dynamically adjust the byte range of chunks during transmission using native HTTP/2 features. The MP-H2 prototype implementation shows better performance than mHTTP, but slightly worse performance than MPTCP. However, like mHTTP, MP-H2 does not need server-sided deployment or network support, so it overcomes deployment limitations of MPTCP.

Finally, there is the proposal of Multipath QUIC (MPQUIC) [49], which leverages a QUIC extension to use different paths. In their performance evaluation, the authors find that while MPQUIC performs similar to MPTCP when there is no packet loss, it improves performance in lossy scenarios.

MPTCP and other multipath transport protocols are orthogonal to IANS: Combining multiple access networks using MPTCP is yet another valid choice for IANS instead of selecting a single access network. Furthermore, the access network selection algorithms we propose can be used to optimize decisions within an MPTCP path manager and/or scheduler.

2.3 Systems Support

While techniques to use multiple access networks can provide substantial performance benefits, such techniques are not always deployed within current systems, e.g., OSes. As most systems were originally designed without the use case of utilizing different access networks in mind, such systems may need to be modified to support this use case.

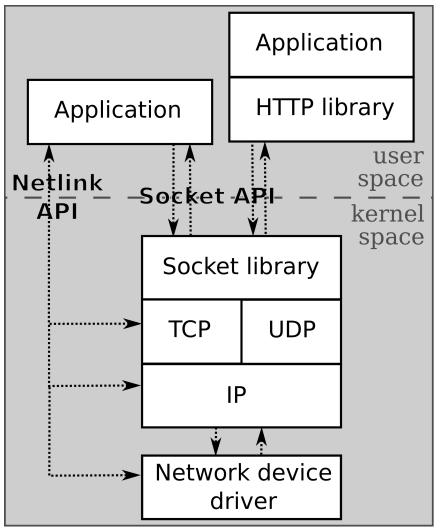


Figure 2.5: Networking within a host.

We first provide background on how networking is implemented within hosts and then examine to what extent OSes support using multiple access networks. As both information available to OSes and information available to applications may play a role in selecting the most suitable access network, we take a closer look at networking APIs between OSes and applications.

2.3.1 Networking Within Hosts

Many hosts, including most end-user devices, are running an OS with a networking implementation similar to Figure 2.5. While applications are implemented in user space, most transport and network layer protocols are implemented in the kernel[6]. Applications may also use user space libraries implementing protocols such as HTTP or Transport Layer Security (TLS). The interface through which user space components communicate with in-kernel protocol implementations is the Socket API. While the Socket API originated as a POSIX standard to be used on UNIX-based OSes, other OSes include similar APIs. For example, Windows include the Winsock2 API which deviates from the Socket API only in a few cases[7]. The Socket API is implemented in the Socket library, which can utilize the network stack implemented in the kernel, i.e., protocols such as TCP, UDP, and IP. Finally, protocols on the data link layer and physical layer are implemented within the hardware of the network interface card or within network interface drivers. In addition to the Socket API, most OSes enable user space applications to query information from in-kernel

[6] QUIC is a notable exception to this rule, as it is a transport protocol often implemented in user space. However, QUIC is built on top of User Datagram Protocol (UDP), which is usually implemented in the kernel.

[7] See Winsock documentation: `https://docs.microsoft.com/en-us/windows/desktop/winsock/porting-socket-applications-to-winsock`.

protocol implementations or network interface drivers, e.g., via the Netlink API on Linux.

To be able to communicate across a network using the TCP/IP stack implemented in the OS kernel, an application first creates a socket as a communication endpoint, e.g., for a TCP connection or for UDP datagrams. Once created, the application can bind the socket to a local IP address and/or port, thus, implicitly choosing the network interface this address is configured on to use for this connection[8]. To establish a TCP connection, the application connects the socket to a remote address and remote port. To use UDP, the application may omit this step and directly send data to a remote address and port. If there is no local address specified for the socket at this point, the OS picks a local IP address, e.g., based on the default route in the local forwarding table. Then, the application can send and receive data through the socket similar to reading from and writing to a file with sequential access. Underneath, the TCP/IP stack encapsulates payload, e.g., within UDP datagrams or TCP segments, and further encapsulates these datagrams or segments within IP packets. The stack then hands the packets to a network interface driver, which controls the network hardware, e.g., a WiFi card, and eventually sends the packets over a network interface. In the reverse direction, when the host receives packets, the network card hands them to the network stack, which filters and decapsulates them. Finally, the stack forwards the payload contained in the packets to the socket that belongs to the connection or datagram endpoint, so the application can read the received data.

As an alternative to in-kernel implementations, protocols can also be implemented in user space. One notable example of a transport protocol often implemented in user space is QUIC, a new transport protocol built on top of UDP. A key advantage of implementing QUIC in user space is the ability to deploy updates, e.g., of new versions of the protocol, without having to issue a new release of the OS kernel. However, user space protocol implementations may not be available through the Socket API, but provide their own abstraction, i.e., networking API. As there is not yet a standard API for QUIC, current QUIC implementations are often tightly integrated into applications that use them. Alternatively, some QUIC implementations provide a "codec-style" API: Applications supply their payload as unencrypted plaintext to the QUIC implementation, which then returns an encrypted QUIC packet for the application to send over UDP, e.g., using the Socket API. Here, while QUIC is implemented in user space, it utilizes the UDP implementation of the kernel. In contrast, some user-space protocol implementations completely bypass the OS kernel and directly communicate with network interface drivers for performance reasons. Often, the goal is to increase data rates for applications by speeding up packet processing. Such user-space protocol implementations may use libraries such as the Data Plane Development Kit (DPDK).

Mobile OSes, often running on mobile phones and tablets, slightly deviate from the model in Figure 2.5 in two significant ways: First, there may not be direct access to

[8] This assumes that the system is configured with a routing policy to send traffic with a specific source address over the corresponding access network.

the network performance characteristics for all connected networks. While applications may be able to query WiFi network performance characteristics, information about cellular networks may not be available in the same level of detail. The reason is that mobile OSes often have only indirect access to the cellular modem, as the network interface driver of the cellular modem is running on a different processor than the OS: While the OS runs on one or multiple application processors, the code that implements the radio stack for access to cellular networks runs on the baseband processor. This code implements cellular protocols with functionalities such as network registration, authentication, and mobility. Furthermore, the code running on the baseband processor communicates with the SIM card as well as the cellular modem. The OS running on the application processor can communicate with the code running on the baseband processor, e.g., using asynchronous serial communication using AT commands, and can, thus, still send data via cellular. However, as the OS kernel has only indirect access to the cellular modem, it can only provide limited information to user space. Second, mobile OSes may expose a different API than the Socket API to applications, through which applications can open connections, communicate, and query information about network performance characteristics.

2.3.2 Support For Multiple Access Networks Within Operating Systems

A host may have multiple network interfaces available, through which it can reach the Internet. To make use of these multiple interfaces, the first step is for the host's OS to support multihoming. In this context, multihoming means that a host can have different IP addresses configured on its interfaces and can use multiple interfaces to communicate simultaneously. Most OSes support configuring one or multiple network interfaces with one or multiple IPv4 and IPv6 addresses. Often, each interface can have one IPv4 and multiple IPv6 addresses. For IPv6, the OS may perform source address selection [50] between multiple addresses. However, IPv6 source address selection is primarily concerned with reachability, and is usually neither aware of different interfaces, access network performance over these interfaces, nor application requirements.

If a host has multiple network interfaces with one or multiple IP addresses and if it can reach a remote host via each of these interfaces, the host can usually establish network connections to this host using multiple interfaces at the same time. For instance, an established TCP connection is bound to a combination of local and remote IP address. An application may explicitly bind different TCP connections to different local IP addresses, e.g., using the `bind()` call in the Socket API. If these IP addresses are configured on different access networks, the application uses multiple access networks as a result. However, using multiple local IP addresses for the same connection via MPTCP requires additional support from the OS. As MPTCP implementations are typically tied to TCP implementations, which are usually implemented in the kernel, MPTCP typically requires kernel support. This support is not yet present in all OSes. Moreover, MPTCP has to be supported on both hosts involved in the communication. While client-sided support may be present, e.g., in

major mobile OSes such as Apple iOS [51], server-sided support is still rare [52]. When MPTCP is supported, the OS can automatically enable MPTCP for all TCP connections. As MPTCP has been designed to transparently substitute single-path TCP, see Section 2.2.3.1, application support is not required. Here, the MPTCP path manager, which is usually implemented in the OS kernel, chooses on which interfaces to establish MPTCP subflows.

For protocols which do not support multiple paths, such as TCP, the OS may choose between different local network interfaces, i.e., different access networks, within the routing subsystem in the kernel. Here, some OSes choose a static default interface, e.g., WiFi [15]. Alternatively, some mobile OSes implement a centralized connection manager [15], which may enable cellular on a per-application basis.

As an alternative to letting the OS select an access network, an application can specify a local IP address to use through the Socket API. By doing so, the application implicitly chooses to use the local network interface that this address is configured on, thus, the access network that the interface is connected to. However, to select the most suitable access network, an application may first have to gather information about network performance characteristics, e.g., through the Netlink API. Whether an application actually does this depends on the application: Some applications already implement their own networking abstractions on top of the Socket API, e.g., to manage multiple connections, re-establish connections in case of disruptions, and choose between IPv4 and IPv6. Such applications may implement access network selection in the same connection management logic. Moreover, libraries implementing application layer protocols, e.g., HTTP, may provide such functionality to applications[9]. However, such solutions on top of the Socket API are only available to applications that implement their own connection management logic or use specific libraries that implement this functionality. Therefore, per-application solutions do not represent OS support for using multiple access networks. Moreover, per-application solutions come with disadvantages, such as an increase in application complexity, code redundancy, and the inability to optimize access network selection across multiple applications. Similarly, libraries that utilize networking, such as TLS libraries, become more complex if each such library has to implement access network selection.

As an alternative, access network selection can be implemented outside of the context of an individual application, e.g., in the OS kernel. However, this makes it more difficult to consider application needs: The OS is usually unable to distinguish between connections with different requirements, as the Socket API does not allow applications to provide such information. Annotating a connection with additional information requires enhanced networking APIs.

There have been academic efforts to design networking APIs that provide additional information regarding application needs. Early efforts include QSockets [53], which

[9] For example, the OkHttp library, see `http://square.github.io/okhttp/`, not only implements connection management but also selects between multiple access networks depending on connection establishment delay.

allow an application to specify its Quality of Service (QoS) requirements on a per-connection basis, with the goal to inform packet schedulers. For QSockets, selecting between access networks is not yet a use case. Intentional networking [54] allows applications to specify traffic properties for access network selection. Their API provides a message abstraction with dependency information.

More recent efforts include NEAT [55], which provides a platform- and protocol-independent API that includes access network selection. Protocol-independent means that when using the NEAT API, instead of selecting a specific transport protocol, applications only provide the transport services they require. The NEAT system then selects a transport protocol as well as a path over one of the locally available access networks. Hereby, NEAT can optimize choices of protocol and path. The advantage of NEAT is that it can potentially use transport protocols unknown to the application developer without any application modification. NEAT is related to standardization efforts in the IETF TAPS Working Group. TAPS is in the process of standardizing a new transport API [56], which supports selecting between different transport protocols, remote endpoint IP addresses, and locally available access networks. The work described in this thesis has served as input to the TAPS API, similar to NEAT.

To overcome the limitations of the Socket API in practice, some mobile OSes expose more advanced networking APIs[10]. These APIs enable applications to query information about network interfaces, enable them to select a network with certain capabilities[11], or facilitate failover if network connectivity is lost on a particular interface.

Our work extends existing efforts to make multiple access networks available. We propose to dynamically select between multiple access networks based on both application needs and network performance characteristics. Our solution requires applications to use a new networking API, but does not require support from OSes, networks, or remote hosts.

[10] For example, Android exposes the Connectivity Manager API, see `https://developer.android.com/reference/android/net/ConnectivityManager#requestnetwork`, and Apple exposes the NWConnection API, see `https://developer.apple.com/documentation/network/nwconnection`. The NWConnection API is an implementation of the TAPS API.

[11] For example, on Android, possible network capabilities include: Not metered, not congested, not restricted, not roaming, not VPN, and multipath preferences, i.e., whether to use a secondary network interface as fallback or whether to use multiple network interfaces in parallel.

3

Assessing Application Performance

One major objective of Informed Access Network Selection (IANS) is to improve application performance. Yet, to assess whether this objective is achieved, we have to be able to assess application performance.

In this thesis, we focus on two major applications: Web browsing and HTTP Adaptive Streaming (HAS). Measuring Web performance in a manner that the results are both realistic and reliable is not straight-forward. Thus, we first provide an overview of different performance metrics relevant for Web browsing. Then, we conduct an in-depth analysis of Web performance measurement and outline what pitfalls to avoid. Finally, we discuss several performance metrics for video streaming using HAS.

3.1 Web Browsing

Since Web browsing is one of the most prevalent applications in today's Internet, understanding its performance is critical. Hereby, both metrics as well as experiments have to realistically reflect possible performance improvements as experienced by actual users. Moreover, they need to be reproducible. However, quantifying Web performance is challenging due to Web page diversity, heterogeneous end-user device types and browsers, choice of metrics, including network-centric, browser-centric, and user-centric metrics, and the lack of well-established standards. Given this diversity, we first provide an overview of several Web performance metrics. We, then, check these metrics regarding their data sources and tools.

3.1.1 Performance Metrics Definitions and Data Sources

Most often, Web performance is quantified in terms of load times measured during the page load process. Other fundamental aspects of a page load are the number and the sizes of resources that are transferred, which are often used to compute integral metrics such as Byte Index. Here, we provide definitions for these metrics, and data sources from which they can be derived.

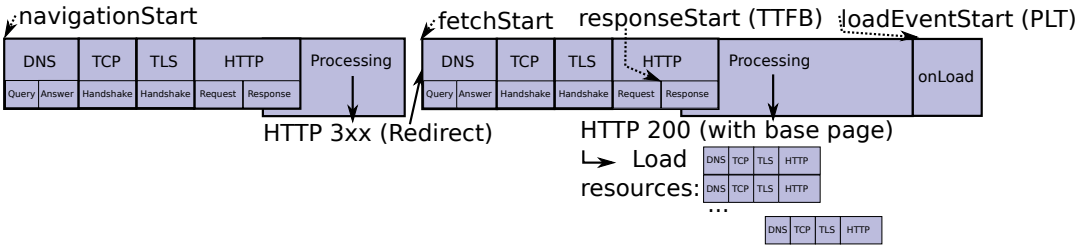


Figure 3.1: Browser events and timings.

3.1.1.1 Load Times

To understand the different load time metrics for Web pages, it is useful to revisit the page load process: To load a Web page, a browser usually loads the base document, parses it, constructs a Document Object Model (DOM), loads the referenced resources, processes them, and displays the results. Figure 3.1 shows a detailed view of this process including browser events, which are the basis of several commonly used load times metrics.

Typically, Page Load Time (PLT) is defined as the time until the onLoad event. However, in the eye of the user, the actual Web page load is often finished earlier, e.g., when the content is displayed on the screen. Thus, other timings include domContentLoaded, when all resources referenced in the base document have been loaded, Time To First Paint (TTFP), when the first content is rendered, or Above-The-Fold time (ATF), when the part of the page visible on the user's screen has been fully rendered. Depending on the Uniform Resource Locator (URL) being retrieved, load times may include initial redirects.

Load times are available from different data sources. For instance, load times based on browser navigation events are available through the standardized Navigation Timings Application Programming Interface (API)[1]. Moreover, TTFP is currently being standardized[2]. Being standardized implies that these metrics are available for different browsers based on a similar definition. HTTP Archive (HAR) files[3] also include PLT and domContentLoaded times. Different from Navigation Timings-based load times and TTFP, ATF is not standardized. Estimating ATF requires not only load times but also positions of the resources within the Web page [57]. Resource load times are available through the Resource Timings API[4] or from HAR files. Resource positions within the Web page and dimensions are available, e.g., using jQuery, a JavaScript library to traverse and query the DOM.

[1] See Navigation Timings specification version 1: `https://www.w3.org/TR/navigation-timing/` and version 2: `https://www.w3.org/TR/navigation-timing-2/`.
[2] See Paint Timings specification: `https://www.w3.org/TR/paint-timing/`.
[3] See HAR specification draft: `https://w3c.github.io/web-performance/specs/HAR/Overview.html`.
[4] See Resource Timings specification version 1: `https://www.w3.org/TR/resource-timing-1/` and version 2: `https://www.w3.org/TR/resource-timing-2/`.

3.1.1.2 Resource Count and Size

To better understand the complexity of Web pages and the page load process, it is useful to consider the resource count and the sizes of the resources which are part of the page.

For an accurate resource count, it is important to note that contemporary Web pages often fetch resources continuously even after the initial page load has completed. Thus, resource count should only include those resources loaded until the onLoad event. Possible definitions include the number of HyperText Transfer Protocol (HTTP) request-response pairs observed during the page load or the number of resources in the DOM. With regards to resource size, networking-related studies usually use the encoded size, i.e., the number of bytes transferred over the network, as opposed to the decoded size, i.e., the number of bytes after decompression. However, as resources are transferred over HTTP, each resource transfer implies an overhead, namely the HTTP headers. Unfortunately, it is often unclear if a given resource size definition includes the header or not. The total page size is the sum of all resource sizes. Byte Index is the integral of sizes of resources loaded over time, see [58]. Consequently, it relies on the accuracy of resource sizes and resource counts.

As data sources, one way to derive resource count is to count the number of HTTP request-response pairs using the list of entries in a HAR file. Another list of resources involved in constructing a Web page is reported by the Resource Timings API. HAR files as well as Resource Timings version 2 provide encoded and decoded body size of each resource. In addition, HAR files contain HTTP headers, possibly including a Content-Length header, and header sizes. Note that for HTTP/2, logged header sizes do not correspond to bytes on the wire anymore due to HTTP/2 header compression. Resource Timings also include the transfer sizes of header and body. An alternative is to extract the number of resources from a packet capture trace if it is possible to successfully decrypt all packets. However, exact resource sizes can be off due to Transport Layer Security (TLS) padding.

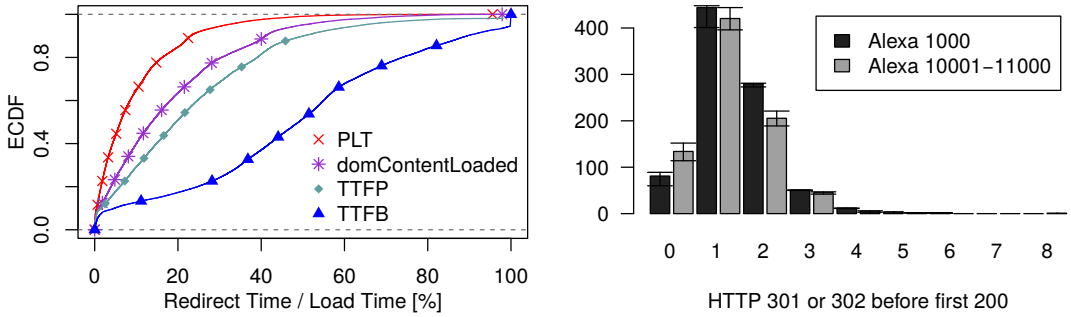
3.1.2 How To Reliably Measure Web Metrics

As Web performance measurement and comparison relies on the accuracy of metrics such as load times and resource count and sizes, we need to check their accuracy. To do so, we compare different data sources for the same page load.

We find that initial redirects can substantially inflate load times. Moreover, some data sources provide more reliable resource sizes and resource counts than others. Consequently, Byte Index can differ based on the data source used to compute it.

3.1.2.1 Web Metrics Comparison Methodology

To assess the impact of data sources and tools on Web performance metrics, we measure these metrics from different data sources for the same page load.



(a) Nav. Timings: Redirect share of load time

(b) HAR: Number of redirects

Figure 3.2: Effects of initial redirects.

We load pages from a Thinkpad L450 with Debian Stretch, using the following tools: 1.) Firefox 61.0.2 with Selenium 3.14.0 and geckodriver 0.21.0, 2.) Firefox 61.0.2 with Marionette, and 3.) Chrome 69 with Chrome DevTools. For realistic browser behavior, which includes the rendering engine, we open Web browsers including the graphical user interface rather than in headless mode. To avoid bandwidth limitations, our vantage point is directly connected to a university network. To minimize the effects of Domain Name System (DNS) caching and delay to the resolver, instead of popular open resolvers, we use a DNS recursive resolver close to our vantage point in terms of network distance.

Since the most commonly used workload is the Alexa top list, despite its limitations [59], we also use a snapshot of the global Alexa Top 1000 and the Alexa 10001 to 11000. To generate our hitlist, we query the Alexa Top 1000 on 18. September 2018 and the Alexa Top 1 Million, from which we derive the Alexa 10001-11000, on 30. September 2018. We then repeatedly access each page 10 times with the different tools. This ensures that all experiments for a single page are done within a reasonable time window. Overall, we execute our Web metrics experiments between 18. September and 11. October 2018. For each page, we first initialize a new browser profile with a cold browser cache. We then fetch the page and wait for it to load by instructing the browser automation tool to wait for the onLoad event. As data sources, we export Navigation Timings, Resource Timings, TTFP, and the HAR file using the native HAR export of the browser via har-export-trigger 0.6.1. In parallel, we also run a packet capture to derive the baseline of our validation. If one of the data sources does not yield any data, we log an error and exclude the page load attempt from the data set.

3.1.2.2 Redirects

In the first step of our metrics comparison, we take a closer look at load times and notice the influence of initial redirects. As shown in Figure 3.1, initial redirects are included in load times, if they occur. However, whether including redirects is realistic depends on the measurement objective: Timings excluding redirects may be more representative of page loads by actual users due to browser optimizations. For example, redirects do not occur in practice if the user types the first few letters

and then clicks on a URL suggested by the browser. Similarly, some browsers prevent redirects by automatically using HTTPS due to HSTS or by adding "www" to domain names the user types[5]. In contrast, load times including redirects are representative of page loads if a user types in the full URL and presses Enter. Therefore, a conscious choice should be made and the URL hitlist should be adjusted accordingly.

To assess the impact of redirects we first count the number of server-sided redirects[6] for both the Alexa 1000 and 10000-11000, see Figure 3.2b. The most common cause for a redirect is that a page is not available over unencrypted HTTP and, thus, the browser is redirected to the HTTPS version. Given that many pages have migrated to HTTPS, e.g., 75% of Web pages loaded by Firefox users[7] in September 2018, this is not surprising. Other reasons for redirects include pointers to subdomains, e.g., for localized versions of the content based on geolocation. Often both occur and lead to two redirects. Next, we revisit page load times: For Navigation Timings, redirect times are the time between navigationStart and fetchStart. For HAR files, we use the time from the start of the page load until the first HTTP 200 response. To quantify the contribution of redirect times to the overall load times, we show, in Figure 3.2a, the relative percentage of load times of redirects for all Web pages. Redirects account for 6.1% of PLT for 50% of the pages and for 23% of PLT for 10% of pages. This implies that the PLT with or without redirects differs by this amount. The difference is even larger for user-centric load time metrics as these are usually shorter. For instance, TTFP differs by 19.1% for 50% of pages and by 47% for 10% of pages. Indeed, the time for the redirects is about the same as the Time To First Byte after the redirect for about 50% of pages. The reason is that most redirects typically involve an additional DNS resolution, Transmission Control Protocol (TCP) connection establishment, TLS handshake, and HTTP request. At the time of this experiment, in September and early October 2018, TLS 1.3 was still not deployed, which causes our redirects to include one additional round trip. In summary, we make the following observations: 1.) Redirects account for a significant share of PLT and a substantial share of user-centric load time metrics such as TTFP. 2.) Studies should make a conscious choice to in- or exclude redirects. For example, load times including redirects reflect load times for the first visit to a Web page after typing the domain name into the browser address bar, while excluding redirects more accurately reflect loads times for returning users and including browser optimizations.

It is possible to exclude redirects upfront, e.g., by adjusting the hitlist to post-redirect URLs. However, post-redirect URLs may change, e.g., due to geolocation or HTTPS migration. Such changes may lead to more page load failures, compared to starting from the "base" URLs of http:// and the top-level domain name. Alternatively, redirects can be excluded in retrospect by computing the timings relative to fetchStart instead of navigationStart for Navigation Timings resp. relative to the start time of the first HTTP 200 resources for HAR files. In our measurements evaluating access

[5] See, e.g., `https://support.mozilla.org/en-US/kb/search-web-address-bar`.

[6] Server-sided redirects use HTTP status 301 or 302. Client-side redirects use status 200 and contain the redirection URL in the response content, which we do not log.

[7] See `https://letsencrypt.org/stats/#percent-pageloads`.

network selection, see Chapter 8, we choose to exclude redirects by adjusting the hitlist.

3.1.2.3 Resource Count and Size

Next, we take a closer look at resource sizes. In particular, we explore if different data sources are consistent with the baseline from the the packet capture trace and if they yield similar results.

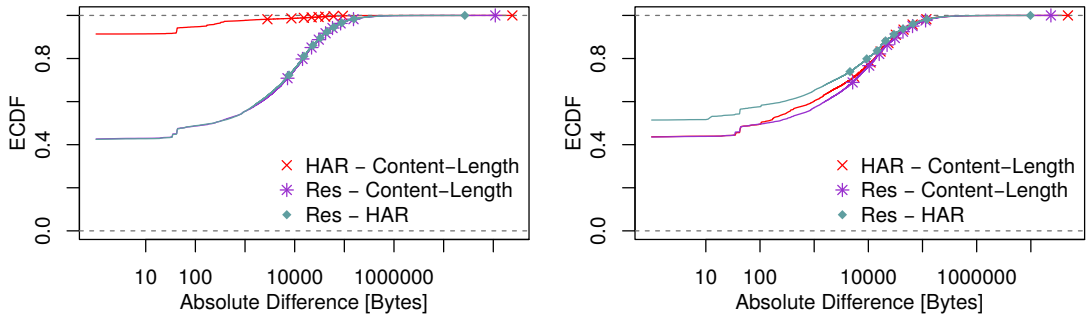
To validate the resource sizes recorded by the different data sources, see Section 3.1.1.2, we compare them against a baseline computed from the packet capture trace. This, unfortunately, is only possible for resources loaded over unencrypted HTTP/1.0 or HTTP/1.1. If TLS is used, resource sizes may be incorrect due to padding. For computing the baseline, we extract HTTP request and response pairs from the packet capture trace and exclude resources with missing bytes. For the remaining resources, we separate the TCP payload into the HTTP header and body to count bytes separately. Finally, we match the resource to the corresponding HAR and Resource Timing (Res) data based on timestamps. Hereby, we exclude ambiguous cases, i.e., where multiple HAR entries match a resource from the trace. The resulting comparison is summarized in Table 3.1. If the Content-Length header is present, its information is mostly consistent with the traces. None of the other data sources is that good. Rather, we find that the accuracy varies widely across data sources and browsers. When manually investigating the most significant mismatches, we find that Resource Timings set resource size to 0 for most cross-origin resources[8], even though a nonzero amount of bytes was transferred. In HAR files, the body size is often set to -1 if the browser does not succeed in loading a resource. In several cases, Firefox counts too many bytes in case of redirects. Apparently, Firefox returns the size of the redirect destination instead of the actual resource size of the redirect. This is most likely a bug in Firefox.

Next, we explore the consistency of the results for all resources including those that are transferred over an encrypted connection. Figure 3.3 shows the resource size differences for the same resource and various data source combinations, i.e., HAR file body size (HAR), Content-Length header taken from HAR file, and Resource Timings encoded body size (Res). Since Content-Length is a close approximation to the baseline for unencrypted resources, we use it as a baseline. Res provides the exact same resource size as Content-Length in only 42.5% of cases for Firefox and in 43.4% of cases for Chrome. This is consistent with the results for unencrypted resources, see Table 3.1. For HAR, Firefox provides a resource size which matches the Content-Length for 91.3% of cases, see Figure 3.3a. Thus, we conclude that HAR's accuracy is better than for unencrypted resources. In contrast, Chrome provides a resource size which matches the Content-Length in only 39.4% of cases for all resources. When

[8] Unless the "Timing-Allow-Origin" header is set, see Resource Timings version 2 specification: `https://www.w3.org/TR/resource-timing-2/`

Table 3.1: Object sizes: Accuracies for unencrypted resources.

Comparison	Browser	Match	Counted too many bytes			Counted too few bytes		
		Cases [%]	Cases [%]	99th quantile [KB]	Max [KB]	Cases [%]	99th quantile [KB]	Max [KB]
Content-Length	Firefox	100	0	0	0	0	0	6.8
	Chrome	100	0	0	0	0	0	0
HAR body size	Firefox [10]	72.6	13.4	66.28	2170	14	0.13	852.4
	Chrome	91.9	0.5	0	303.4	7.6	0.3	2925
Res body size	Firefox	39.6	0.8	0	2910	59.6	196.6	5092
	Chrome	46	0.5	0	276.5	53.5	181.5	5092



(a) Firefox (b) Chrome

Figure 3.3: Resource sizes: Differences due to data source for all resources.

investigating the difference, we find that Chrome sets HAR body and header size to -1 for all HTTP/2 resources[9]. This is most likely a bug in Chrome.

From this, we conclude: 1.) Content-Length provides the most accurate resource size but is not always available. For example, HTTP servers may not set Content-Length for dynamically generated resources which are transfered using chunked encoding. 2.) Resource Timings are an unreliable data source for resource sizes, as they do not provide sizes for cross-origin resources, except when explicitly allowed. 3.) HAR body size is inaccurate for a significant number of resources, due to bugs in both Firefox and Chrome (whereby Firefox is more accurate than Chrome).

Amazingly, we find that not only resource sizes differ by data sources but also resource counts (for the same page load)! For the Alexa 1000 dataset, resource counts from HAR and Res always differ by at least one resource and by 7 or more resources for 50% of cases. For 10% of the cases, they are off by more than 67 resources. Numbers for Alexa 10000 are similar. Among the main contributor to this difference is that Resource Timings do not include resources loaded within commonly embedded

[9] In Firefox, only resources transferred using HTTP/2 Server Push lack body size and timings.

[10] In HAR files, Firefox logs body size including headers, contradicting the HAR specification at `https://w3c.github.io/web-performance/specs/HAR/Overview.html`. See `https://dxr.mozilla.org/mozilla-central/source/devtoools/server/actors/network-monitor/network-response-listener.js#428`, accessed 28.09.2018, for the relevant source code. Thus, we subtract header size from all resource sizes.

Table 3.2: Further Web performance pitfalls.

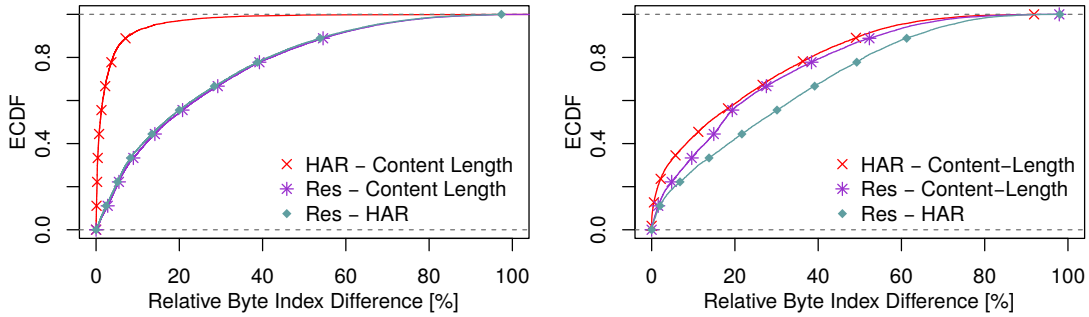
Pitfall	Description	Guidelines
Failed page loads	For some URLs in our workload, the browser never invokes the onLoad event. The most common reasons are no DNS response, not being able to establish a TCP connection, or certificate errors.	Exclude URLs which do not point to Web pages.
Data source availability	Data sources do not export any data in some cases, e.g., no HAR file, no Resource Timings, or neither. Furthermore, Chrome often exported data too early in our experiments, i.e., before the onLoad event.	Carefully choose browser and automation tools. In our experiments we find that Firefox instrumented by Marionette is more likely to provide complete data than Firefox with Selenium or Chrome.
Outdated tools	Major Web browsers have update cycles shorter than a typical research project. Thus, tools quickly become out-of-date.	Consciously address the trade-off of updating: Software updates may fix bugs and provide performance optimizations. However, they may also introduce compatibility issues.
Unrelated traffic	Modern Web browsers often load data unrelated to page loads, e.g., software updates or blocklists. This traffic can cause significant performance overhead	Disable features which result in unwanted traffic.
New protocols	Newly introduced protocols relevant to Web, such as HTTP/2 or Quick UDP Internet Connection (QUIC), can invalidate prior assumptions about Web traffic. Moreover, they may require updates to the measurement and evaluation setup or trigger so far unknown bugs in the evaluation.	Revisit assumptions once new protocols are introduced.

HTML Inline Frames (iframes). Rather, these resources are recorded in the Resource Timeline for the iframe.

Again, we conclude that Resource Timings are an unreliable data source for resource counts as well as resource sizes. By specification, they do not include resources of embedded frames and often do not provide sizes for cross-origin resources. These observations inform our measurement methodology, see Section 8.1, in the following way: Our resource counts and sizes are based on Content-Length in cases where it is available. In other cases, we use HAR body size as logged by Firefox.

3.1.2.4 Impact on Byte Index

To illustrate the consequences of using inaccurate data sources for resource counts and sizes, next, we quantify the impact of resource size and count differences on Byte Index [58]. Byte Index captures page load progress, i.e., loaded bytes over time. In Figure 3.4, we plot the relative difference between Byte Index for the same page load, calculated from HAR body sizes, Resource Timings body sizes, and Content-Length header (using HAR body size if the Content-Length header is missing). For Firefox, see Figure 3.4a, Byte Index is almost identical for Content-Length and HAR body size. However, Byte Index differs by 17.1% for Res in 50% of the pages loads, and by 56.4% for 10%. For Chrome, see Figure 3.4b, the Byte Index derived from both



(a) Firefox (b) Chrome

Figure 3.4: Byte Index: Difference due to data source.

Res and HAR differs substantially from the Byte Index derived from the Content-Length.

Thus, we conclude that computing Byte Index from Resource Timings leads to results that are quite inaccurate. As we use more reliable data sources in our evaluation, see Section 8.1, we are confident that the Byte Index we compute more accurately reflects the actual page load process.

We summarize further pitfalls, as well as our guidelines, in Table 3.2. See our PAM paper [60] for details. To help others to avoid these pitfalls, we make our Web measurement tools publicly available[11].

3.2 Video Streaming

In addition to Web browsing, we evaluate the performance of another major application: Video streaming. Video streaming has multiple use cases, including live streaming and video on demand, and uses different technologies, e.g., Real-Time Transport Protocol (RTP), WebRTC, and HTTP Adaptive Streaming (HAS). We focus on HAS because it can adapt dynamically to network conditions. As, with IANS, we can provide the best possible network conditions by selecting the most suitable access network, we see a high potential for IANS to improve HAS performance.

First, we provide an overview of HAS. Then, we describe the performance metrics which are relevant for HAS and the potential impact of these metrics on Quality of Experience (QoE).

3.2.1 Overview

HAS provides a way to continuously transfer media content over HTTP. Using HTTP allows HAS to host content on Web servers instead of having to deploy dedicated server software. Moreover, HAS can leverage features such as HTTP caching and

[11] https://github.com/theri/web-measurement-tools

Content Delivery Networks (CDNs). HAS was standardized as Dynamic Adaptive Streaming over HTTP (DASH) [61].

HAS enables to deliver media content such as video or audio. Each media content component is called an Adaptation Set. With HAS, the media content delivered to a client adapts to a number of factors, such as the end-user device type, screen resolution and the available network conditions, e.g., the available downstream capacity. To achieve this, each Adaptation Set may contain different representations of the same media content, e.g., the same video encoded for different quality levels, which results in different bitrates. The HAS client can now request the representation that best matches the available resources, such as the downstream capacity.

For media content to be available as an Adaptation Set for HAS, the content first has to be divided into segments of a certain length. Each segment now contains a part of the video content, which is encoded using a video or audio codec. Encoding segments of the same content with different quality levels, screen resolutions, or codecs, results in different representations of the content. Each segment is then stored on a Web server and can be loaded by HAS clients using HTTP. To find out which representations exist for a specific content, a client first loads a manifest file, e.g., a Media Presentation Description (MPD) file in DASH. This MPD file includes a list of representations, their resolutions and bitrates, as well as the URLs of the initial segments. For each representation, the initial segment includes the URLs of the actual segments containing the content. Moreover, the MPD file contains information about the codec, total duration, and suggested minimum buffer level before starting playback.

MPD files, initial segments, and segments containing Adaptation Sets are hosted on Web servers. To start loading content, a HAS client, such as a video player, first loads the corresponding MPD file. From this file, the client learns the necessary information to start loading the initial segments for each representation. The client now chooses the initial representation for the first segment, loads this segment and decodes the content into the playout buffer. With enough content in the playout buffer, e.g., based on the suggested minimum buffer level in the MPD file or local configuration, the client starts playing the content to the user. At the same time, the client continuously loads more segments. After each segment, the client can adapt to the available downstream capacity by switching to a different representation, as each new segment is loaded using a new HTTP request. The algorithm to switch representations after each segment is called Adaptive Bit-Rate algorithm (ABR). For example, the ABR may decide to load the next segments with a higher quality if there is sufficient downstream capacity available or it may load the next segment with a lower quality level if the available downstream capacity has decreased. If the client fails to load the segment fast enough, so the buffer runs out, the media playout freezes, which we call stalling. To avoid stalling while still providing the best possible video quality, ABRs typically utilize information such as an estimate of the recent download rate and/or the current status of the media playout buffer. Which ABR to use is not standardized in HAS, and each client can choose its own algorithm.

A survey of these ABRs is out of scope for this thesis, but can be found in [62]. In our evaluation for IANS, we use the following ABRs:

- **Buffer Based Approach (BBA)** [63]: BBA determines the next representation to be loaded based on the current buffer level. BBA-0 starts with the lowest representation and keeps loading this representation as long as the buffer level is low. If the buffer level is high, it switches to a higher representation based on a linear function of the buffer level. Variants of this algorithm, such as BBA-1 and BBA-2, account for variable segment sizes and optimize the start phase of the playout: BBA-1 handles segments encoded with variable bit rate (VBR), which vary in size even for the same representation. BBA-2 allows to switch to a higher representation faster at the start of the video playout depending on segment download time.

- **Buffer Occupancy based Lyapunov Algorithm (BOLA)** [64]: Similar to BBA, BOLA determines the next representation based on the current buffer level. Hereby, it computes a function based on the buffer level using a utility maximization function, which aims to minimize stalling and maximize video quality. The function includes a parameter to set the relative importance of stalling to video quality.

When selecting the most suitable network using IANS, our goal is to provide the HAS client and its ABR with the best possible network conditions in order to improve HAS performance.

3.2.2 Measuring Video Streaming Performance

As we aim to improve HAS performance using IANS, we measure its performance using the following metrics:

- **Initial playout delay**: The time between starting to load the video and starting the actual playback of the media content to the user. This delay depends on how much content the client buffers before starting the playout. When deciding how much to buffer, a client has to balance initial playout delay with the risk of stalling events during playout. While the manifest file may contain a suggestion on how much to buffer, ultimately, the client decides when to start playing out the media content.

- **Number of stalling events**: The number of times when the client has to stop the playback because insufficient content is buffered.

- **Durations of stalling events**: How long playout stops during stalling events.

- **Video quality**: The representation, e.g., the resolution and bit rate, of the media being played out to the user.

- **Number of representation switching events**: The number of times that the client's ABR switches between representations. For example, switching to a lower representation may prevent stalling, and switching to a higher representation may improve video quality. Additionally, it is possible to switch between representations encoded with different codecs.
- **Amplitudes of representation switches**: The number of representations by which the client switches down or up in each switching event.

In contrast to Web browsing, for HAS, there is no common standardized API through which different video players are supposed to provide these metrics with a common definition. Instead, each video player has to implement its own logic to measure these metrics.

Different performance metrics have a varying impact on QoE. Seufert et al. [65] find that the number of stalling events is the most significant factor. The durations of stalling events also have an impact, but fewer stalling events with longer durations have a lower impact on QoE than more stalling events with shorter durations. In the absence of stalling, representation switching events with high amplitudes have the highest impact on QoE, in particular if the new representation corresponds to a low video quality. The number of representation switching events does not affect QoE as much, but, still, for a high QoE, representation switching events should not be too frequent, e.g., not occur more often than every 2 seconds.

Building on these findings, the ITU-T model P.1203 [66, 67] estimates QoE from the measured performance metrics. Seufert et al. [68] further study the impact of the HAS performance metrics on the P.1203 model. Due to its capability to distill multiple performance metrics into a single number which eases comparison, we later use the P.1203 model in our evaluation, see Section 8.1, to determine whether our IANS policy improves HAS performance.

4

Expressing Application Needs Using Socket Intents

Informed Access Network Selection (IANS) not only depends on the network performance but also on the application needs: Different kinds of application traffic benefit from different network performance characteristics. Applications have different preferences and requirements as to what constitutes "good network performance" in their particular context. While applications are likely to know what to optimize for, it is inconvenient to perform access network selection within each application: Having to observe current network performance and select a network to use introduces complexity and overhead, as each new application has to replicate the effort of other applications. To avoid reimplementing access network selection within each application, we propose to use a separate component, an access network selection policy, which performs access network selection on behalf of all applications. The access network selection policy is realized as a separate piece of software running in user space. See Section 6.1 for a more detailed discussion of access network selection policies. However, there is the question of how this access network selection policy should learn about application needs. To enable applications to express their needs regarding access network selection, we introduced the concept of Socket Intents [69] and refined it within an Internet Draft in the Internet Engineering Task Force (IETF) [70].

Socket Intents are hints about what the application knows, expects, or wants to achieve regarding its own traffic. By expressing its Intents, an application can share its knowledge about its own communication patterns and express preferences regard-

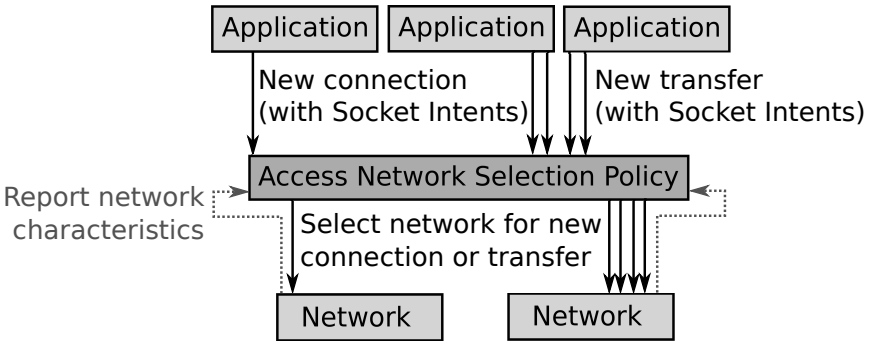


Figure 4.1: Access network selection using Socket Intents.

ing performance. For example, an application can specify its Intents for each new connection or for each individual transfer, see Figure 4.1, to an access network selection policy. An individual transfer can be, for example, each resource that a browser loads as part of a Web page, or individual segments for adaptive video streaming. Then, the access network selection policy decides which network to use, whereby it can base its decision on Intents and network performance characteristics. If an access network selection policy optimizes its decisions based on Socket Intents and current network performance characteristics, see Chapter 5, we call it an IANS policy. Finally, the connection or transfer is placed on the selected network. As the presence of traffic enables gathering more accurate network performance characteristics, IANS policies represent a control loop.

As application needs differ, applications can specify different Intents to provide input to IANS policies. Table 4.1 provides a list of currently defined Intents. An application can opportunistically set the Intents that it knows about or the Intents that it deems useful out of those that it knows. In contrast to Quality of Service (QoS), Intents do not represent hard requirements or resource reservations. They are considered in a best-effort manner. Still, it is possible to set Differentiated Services Code Point (DSCP) values [71] based on Intents. However, Intents cannot provide guarantees for the application. Rather, Intents help to use the available resources in the most efficient way from the point of view of the application. To do so, Intents provide guidance in cases where there are trade-offs, e.g., when choosing between a short latency and a high downstream capacity network.

Applications have an incentive to specify their Intents as accurately as possible to take advantage of the most suitable resources. For example, if an application indicates a preference for short latency by setting the Traffic Category Intent to Query, an IANS policy should prioritize shorter latency in its choice of networks. If an application then sends traffic that does not match its Intents, such as bulk traffic which may benefit from networks with high downstream capacity more than from those with short latency, the application hurts its own performance. Furthermore, a system implementing access network selection knows about the available network resources and about the Intents expressed by different applications trying to use these resources. Thus, the system can balance different requirements and penalize misbehaving applications.

Given that the knowledge actually available to applications may depend on the type of application and its context, different Intents are available, see Table 4.1. Intents range from abstract to concrete. Applications can, thus, choose how much detail to specify. For example, an application can just indicate its generic Traffic Category: When downloading a large file, an application can set the Traffic Category Bulk for a particular connection, which may lead to choosing a network with higher downstream capacity. When sending small queries, an application can set the Traffic Category Query, hoping that a network with short latency will be selected.

Alternatively, an application can be more specific about how much it prioritizes latency using the Timeliness Intent: This Intent uses loosely defined terms, such as

Interactive or Background, that indicate a qualitative assessment of latency requirements without having to quantify them. The goal of these terms is to enable application developers to intuitively assess how much each connection or transfer prioritizes latency, compared to other criteria such as Cost Preference. For example, if the file to be accessed is a system update which is not time critical, the application can set the Timeliness to Background. This may lead to selecting a network which does not have the shortest latency, but also does not incur any additional performance penalties or other costs. Note that if an application tries to prioritize multiple Intents, e.g., both Timeliness and Cost Preference, there may not be an obvious correct choice for the IANS policy. For such cases, IANS policies should include defaults or tiebreaker rules. However, specifying a prioritization using Intents may also help the IANS policy to further balance this traffic with other traffic by other applications. For example, a video call application can set its Timeliness to Interactive and its Disruption Resilience to Sensitive. For this application, an access network with short latency and high consistency can be chosen, which is unlikely to be disrupted.

Some applications may possess a priori knowledge about a new connection, such as Bitrate Received or Size to be Received, e.g., due to configuration or metadata. In this way, an IANS policy can distribute small and large transfers according to their latency and downstream capacity requirements, e.g., by estimating load times. Within video streaming, the application can set the Bitrate Received of the selected representation or the maximum allowed Duration for the next video segment, i.e., the current buffer level. Using this information, the IANS policy can select a network which provides sufficient downstream capacity to load the video segment within the allowed duration.

Intents can also serve as input for optimization of other features than access network selection, e.g., for configuring transport protocols: If the Timeliness is set to Interactive, it may make sense to disable TCP's Nagle algorithm. If the Timeliness is set to Background, it may make sense to choose a scavenger congestion control algorithm. If a Bitrate Sent is provided, this information can be used to de-peak the data rate.

We implement Socket Intents in the APIs of the Socket Intents prototype, see Section 7.1. Our goal is to evaluate the benefit of IANS for application performance. Therefore, we focus on Socket Intents which enable sophisticated IANS policies to improve the performance of a single application. For Web browsing, we use the Size to be Received for individual resources within a Web page load, see Section 6.3. For HAS, we use the Traffic Category, the Duration, and the Bitrate Received for individual video segments, see Section 6.4.

Table 4.1: Socket Intents Definitions.

Intent Type	Value	Definition
Traffic Category	Query	Presumably short requests and responses. Optimize for short latency.
	Control	Presumably long, but low-volume flow. Optimize for short latency.
	Stream	Steady data rate traffic over time. Optimize for low variation.
	Bulk	Transfer of a single file. Optimize for high capacity.
Timeliness	Stream	Real-Time media. Keep delay and packet delay variation as low as possible.
	Interactive	Keep delay low, but some packet delay variation is tolerable.
	Transfer	Keep delay and packet delay variation low, but they are not critical.
	Background	Delay and packet delay variation are not a concern.
Disruption Resilience	Sensitive	Connection loss means application failure.
	Recoverable	Connection loss is inconvenient, but the application can recover from it.
	Resilient	Connection loss is not a concern for the application.
Cost Preference	No Expense	Prohibit using a network that incurs monetary cost.
	Optimize	Avoid using a network that incurs monetary cost, potentially at the expense of performance.
	Balance	Use a network that incurs monetary cost if it is expected to provide a performance benefit.
	Ignore	Monetary cost is not a concern.
Energy Efficiency	Optimize	Use a network that incurs low energy consumption overhead.
	Ignore	Energy consumption overhead is not a concern.
Size to be Received	numeric	Bytes the application expects to receive.
Size to be Sent	numeric	Bytes the application expects to send.
Duration	numeric	Maximum allowed time for a transfer.
Bitrate Sent	numeric	Bytes per second the application expects to send.
Bitrate Received	numeric	Bytes per second the application expects to receive.

5

Network Performance Characteristics for Informed Access Network Selection

When making an informed choice between multiple access networks, an end-user device should consider both application needs as well as current network performance characteristics. Yet, network performance characteristics can vary across access networks as well as over time, see Section 2.1. Therefore, to make an informed choice between different access networks, we require accurate and up to date estimates of the network performance characteristics of the access networks that an end-user device is connected to. However, measuring such network performance characteristics is non-trivial, as they are not necessarily readily available on end-user devices and can be highly volatile.

First, we explore what network performance characteristics we would ideally wish to know to inform Informed Access Network Selection (IANS). As these desired network performance characteristics are not available in practice, we approximate them based on network performance characteristics that are available on a host. We then present what network performance characteristics we collect and how we aggregate them to approximate our desired network performance characteristics.

5.1 Desired Network Performance Characteristics

We consider a multiple access network scenario, in which an end-user device can communicate with a remote host via different access networks, recall Section 2.2.1. These access networks may vary in network performance characteristics such as latency, available downstream capacity, or loss, as underlined by Related Work, see Section 2.1.3.

When selecting an access network for a new connection or transfer, i.e., a flow, our objective is to choose the network with the "best" performance. Ideally, for each potential choice of access network, we would like to possess a priori knowledge about the network performance that the flow would experience if we selected this network. In particular, we would like to know the following network performance characteristics:

- **Latency**: We consider end-to-end latency as the time between sending a packet on one host and receiving the same packet on another host. We are interested

in latency because it directly influences user-visible delays within applications. For example, latency often has a significant impact on connection establishment delay or on the completion time of a data transfer. Latency has several components, including transmission and propagation delays on each link on the path between the hosts, and processing and queueing delays on each node between the hosts. If the delays added by one node or link on a path dominate end-to-end latency, this component is the latency bottleneck of the path.

- **Latency variation**: As different packets experience different latencies, we consider the variation of the latencies of different packets exchanged between the same hosts. We are interested in latency variation as it can be an indicator of congestion on the path that the packets traverse. One possible cause for an increase in latency variation is an increase in queueing delay for some packets before being sent on the bottleneck link.
- **Capacity**: We consider the upstream capacity as the maximum number of bytes per second that can be transmitted end-to-end between two hosts within a flow over a specific path. Furthermore, downstream capacity is the maximum number of bytes that can be received from another host within a flow over a specific path. We are interested in downstream capacity because it may have a high impact on the completion times of a data transfer. Available end-to-end capacity depends on the capacity bottleneck, i.e., the link on the path through which the lowest number of bytes per second can be transmitted. In access networks, this bottleneck may be located on the wireless link, e.g., the first hop of a WiFi or cellular network. Its capacity limitation may be related to low Received Signal Strength (RSS), which may impact the available modulation scheme, and, thus, the data rate, see Section 2.1.1.1. Furthermore, the available capacity on a link depends on the amount of cross-traffic, i.e., the number of bytes per second being sent by other hosts across a common link.
- **Loss**: We consider loss as the percentage of packets sent by a host which are not received by the remote host, e.g., due to being dropped on the path. We are interested in packet loss as it can have a high influence on the completion time of a data transfer, i.e., if part of the data has to be retransmitted. For example, end-to-end packet loss may occur when an end-user device repeatedly fails to decode a received signal on a wireless link[1] or when a node along the path drops a packet from its queue.

We show an example of these network performance characteristics in Figure 5.1. Here, an end-user device has the choice between WiFi and cellular to communicate with a remote host, as the end-user device can reach the remote host with via both networks. More specifically, between the end-user device and the remote host, there is a path via WiFi (green) and a path via cellular (blue). Thus, in principle, the end-user device can send the flow, which consists of packets p_1, p_2, and p_3, via either network, thus, via either path. Yet, if the end-user device selects WiFi, the packets will experience latencies accordingly, i.e., $latency_{WiFi_1}$, $latency_{WiFi_2}$, and $latency_{WiFi_3}$. If the

[1] This assumes that lower layer retransmission on the wireless link itself has failed.

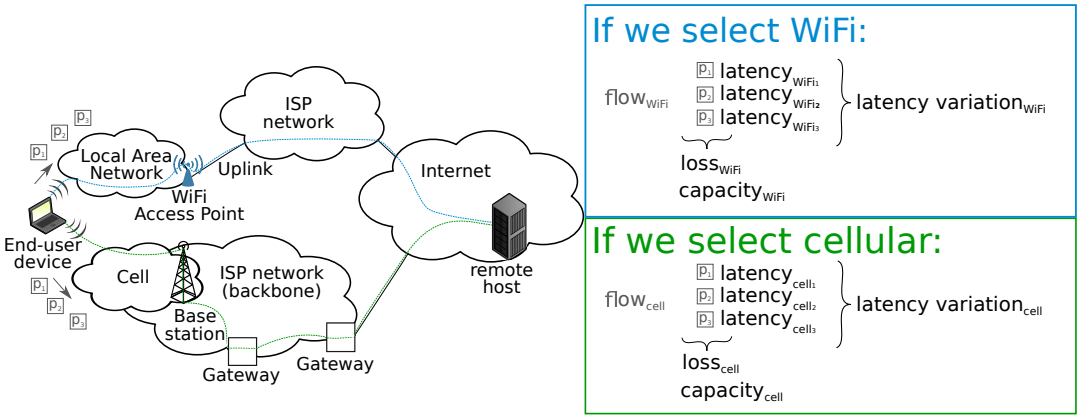


Figure 5.1: Network performance characteristics: Desired.

end-user device selects cellular for the flow, the packets experience different latencies, i.e., $latency_{cell_1}$, $latency_{cell_2}$, and $latency_{cell_3}$. Here, each latency is the delay between the end-user device sending a packet and the remote host receiving the packet, e.g., between sending p_1 at the end-user device and receiving p_1 at the remote host over either the path via WiFi ($latency_{WiFi_1}$) or the path via cellular ($latency_{cell_1}$). As latencies may vary for different packets, these latencies result in $latency\ variation_{WiFi}$ or $latency\ variation_{cell}$. For example, in Figure 5.1, the $latency\ variation_{WiFi}$ is the variance between $latency_{WiFi_1}$, $latency_{WiFi_2}$, and $latency_{WiFi_3}$ as experienced by packets p_1, p_2, and p_3. Moreover, both paths have limited capacity of how many bytes per second can be transmitted from end-user device to remote host and vice versa. For example, in Figure 5.1, the $capacity_{cell}$ in the upstream direction is the maximum number of bytes that the end-user device can send to the remote host over cellular as $flow_{cell}$ within one second, such that the data is received by the remote host. The $capacity_{WiFi}$ in the downstream direction is the maximum number of bytes that can be transmitted from the remote host to the end-user device over the path via WiFi. Moreover, the entire flow, $flow_{WiFi}$ or $flow_{cellular}$ is subject to packet loss, $loss_{WiFi}$ or $loss_{cell}$. For example, in Figure 5.1, the $loss_{cell}$ in the upstream direction is the difference between packets sent by the end-user device and packets received by the remote host (i.e., the number of packets that were sent but not received) divided by the number of packets sent by the end-user device.

These network performance characteristics may vary between WiFi and cellular if the performance bottleneck is within the access network or on a link that is not shared between the path via WiFi and the path via cellular. In contrast, if the bottleneck is in the core or on a shared link, network performance characteristics between WiFi and cellular may be similar.

Note that the path between end-user device and remote host may change during the duration of a flow. Therefore, end-to-end network performance characteristics may vary across packets for reasons unrelated to access network performance.

In addition, there are other path properties which can impact performance. Such path properties include the presence of middleboxes on the path, such as Network

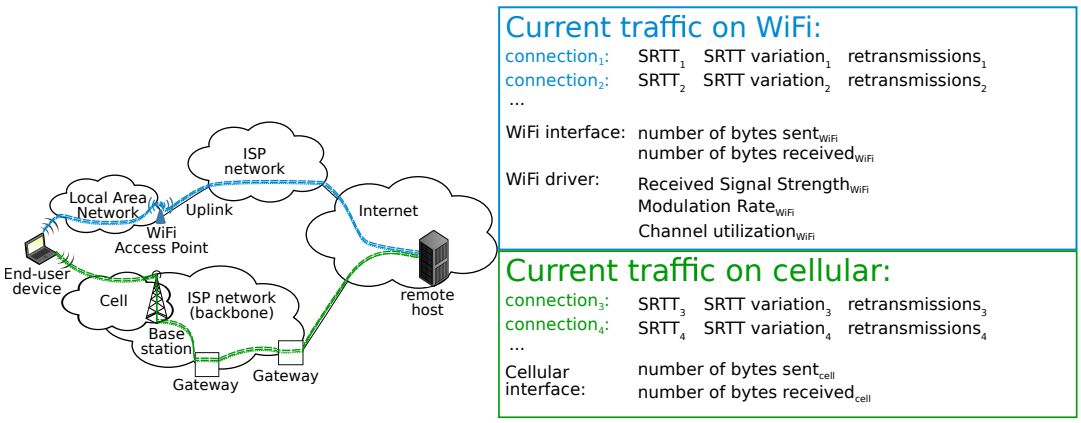


Figure 5.2: Network performance characteristics: Available in practice.

Address Translation (NAT), proxies, or firewalls. Moreover, such middleboxes may influence whether the capacity of an access network can be bundled with another, e.g., whether an Multipath TCP (MPTCP) subflow can be established over this path. Such path properties, which indirectly influence network performance across paths, are considered in our Internet-Draft [72], but are out of scope for this thesis.

Note that while we focus on network performance characteristics, other characteristics can be relevant for access network selection as well. For example, if an end-user device is limited by a data volume quota on cellular, but has no such restrictions on WiFi, a user may deem this more important than performance. We include such differences, e.g., expressed as the Cost Preference Socket Intent, see Chapter 4. However, detecting network characteristics such as data volume quotas is out of scope for this thesis.

5.2 Network Performance Characteristics Collected in Practice

Ideally, we would like to have a priori knowledge about the network performance characteristics that each access network would provide for a new flow. Since such knowledge is not available in practice, we estimate future network performance characteristics based on current traffic [73].

We focus on passive measurements, i.e., by observing existing traffic, as opposed to active measurements, i.e., sending probing traffic, for the following reasons: First, we want to minimize overhead. Second, we want to avoid increasing congestion on the bottleneck. Third, we do not want to use up a user's data volume quota, e.g., on cellular networks. Moreover, existing traffic may reflect network performance characteristics for future traffic more reliably, as this traffic is relevant to applications.

To illustrate what network performance characteristics we can observe on a host, Figure 5.2, again, shows the multiple access network scenario. There are two Trans-

mission Control Protocol (TCP) connections[2] established over each path between the end-user device and the remote host, e.g., $connection_1$ and $connection_2$ over WiFi (blue) as well as $connection_3$ and $connection_4$ over cellular (green). For each of these connections, the end-user device can access the following Smoothed Round Trip Times (SRTTs) from its TCP stack: $SRTT_1$ for $connection_1$ and $SRTT_2$ for $connection_2$ for the connections via WiFi as well as $SRTT_3$ for $connection_3$ and $SRTT_4$ for $connection_4$ via cellular. From the TCP stack, the end-user device also has access to SRTT variations, e.g., $SRTTvariation_1$ for $connection_1$, and retransmissions counts, e.g., $retransmissions_1$ for $connection_1$. In addition to information about current connections, the end-user device can query its network interfaces to find out the $number\ of\ bytes\ sent_{WiFi}$ and $number\ of\ bytes\ sent_{cell}$, as well as the $number\ of\ bytes\ received_{WiFi}$ and $number\ of\ bytes\ received_{cell}$. For WiFi, we can also query the WiFi driver for physical layer network performance characteristics such as $Received\ Signal\ Strength_{WiFi}$, $Modulation\ Rate_{WiFi}$, and $Channel\ utilization_{WiFi}$. Such physical network performance characteristics are not always available for cellular, as cellular modems may not reveal such information.

Note that, the available network performance characteristics are influenced by current traffic patterns, e.g., the amount of data and the number of connections currently present on the network. While SRTTs and retransmission counts are related to current connections, both data rates and physical layer network performance characteristics are based on *all* traffic seen on the end-user device. Note that focusing on existing traffic implies that we only get network performance characteristics for networks we use, i.e., on which any traffic is seen. To limit computational overhead, instead of updating our estimates for every packet that is sent or received, we record the current estimates every n seconds. The choice of the sampling period n represents a trade-off between updating estimates more often, but with higher computational overhead for short n and updating estimates less often, but with lower overhead for long n.

We gather network performance characteristics of the existing traffic and use them to estimate the following desired network performance characteristics:

- **Round Trip Time (RTT)**: One-way end-to-end latency is difficult to measure, e.g., due to clock synchronization issues. Therefore, we estimate one-way latency using Round Trip Time (RTT), i.e., two-way latency. RTT is the time between sending a packet and receiving the corresponding acknowledgment or response. For TCP connections, SRTT samples are readily available from the TCP stack. For other reliable transport protocols, such as QUIC, SRTT samples may be available as well, depending on the implementation and the provided interface. If the latency bottleneck is located within the access network, all connections over a specific access network traverse this bottleneck. In this case, we can aggregate SRTT samples for all connections on the same access network and treat the resulting aggregated SRTT values as estimates for the two-way latency to be expected over this access network. If the bottleneck link is not

[2]For Quick UDP Internet Connection (QUIC), similar performance estimates are available to the protocol implementation, but there is not yet any readily available programming interface to query such estimates from the protocol implementation.

within the access network, then SRTTs may vary greatly among connections to different destinations. In this case, we still can aggregate SRTTs per destination address or subnet.

We aggregate RTTs in the following ways:

- **Minimum RTT ($\mathbf{RTT_{min}}$)**: As a lower bound of the expected RTT, we compute the minimum of all SRTTs of connections for a particular network. For example, in Figure 5.2, the minimum RTT for WiFi is the minimum of $SRTT_1$ and $SRTT_2$. We collect the minimum RTT as an estimate for a lower bound of the two-way latency, i.e., excluding additional latency due to potentially transient effects within the network such as queueing delay.

- **Median RTT ($\mathbf{RTT_{median}}$)**: As a typical value of the expected RTT, we compute the median of all SRTTs of connections for a particular network. For example, in Figure 5.2, the median RTT for WiFi is the median of $SRTT_1$ and $SRTT_2$. We collect the median RTT as an estimate for the expected two-way latency that a flow would experience, including potentially transient effects within the network.

- **RTT Variation**: Since we estimate latency using SRTTs, we estimate latency variation using SRTT variations. Similar to SRTT samples, SRTT variations within each TCP connection are available from the TCP stack. For each connection, the SRTT variation represents the variation of the SRTTs of this connection over time.

  We aggregate RTT variations in the following ways:

  - **Variation within connections ($\mathbf{RTTvar_{within}}$)**: We consider the SRTT variances over time for all current TCP connections over the same access network as our metrics for variation and calculate the median. A high variance within many or all connections can indicate congestion, as those connections may experience an increase in queueing delay at the bottleneck. For example, in Figure 5.2, the median variance within connections for WiFi is the median of $SRTTvariation_1$ and $SRTTvariation_2$.

  - **Variation among connections ($\mathbf{RTTvar_{among}}$)**: We consider the current SRTT values for all TCP connections over the same access network and calculate their variance. A high variation among connections can either indicate congestion that impacts some connections, or performance bottlenecks which are not traversed by all connections. For example, in Figure 5.2, the variation among connections for WiFi is the variance of $SRTT_1$ and $SRTT_2$.

- **Data rate**: To estimate the available capacity, we observe the current data rate sent and received on each access network via the number of sent and received bytes. To get the number of sent and received bytes, the end-user device can query the network interface counters for the interfaces connected to each access

network periodically, i.e., every n seconds. For the current data rate, we compute the difference between successively read counters and divide the difference by the time between readings. If all traffic across an access network shares the same bottleneck, e.g., if the bottleneck is within the access network, then the data rate is representative of the capacity that the traffic currently utilizes.

We aggregate data rates to compute the following capacity estimates:

- **Current data rate ($\mathbf{DRate_{cur}}$)**: We observe the most recent data rate. For example, in Figure 5.2, the current upstream data rate for WiFi is the difference between the current sample of $number\ of\ bytes\ sent_{WiFi}$ and the previous sample of $number\ of\ bytes\ sent_{WiFi}$ read n seconds ago, divided by the sampling period n.
- **Maximum data rate ($\mathbf{DRate_{max}}$)**: We store the maximum of all throughput data rates observed over a certain time period. In our current setup, we have observed that a time window of 5 minutes provides adequate estimates. For example, in Figure 5.2, the maximum upstream data rate for WiFi is the maximum of all upstream data rates within the last 5 minutes.
- **Number of concurrent connections (conns)**: We observe the number of currently established TCP connections over the same access network as an estimate for how many connections might share the same bottleneck in the worst case. For example, in Figure 5.2, the number of concurrent connections for WiFi is 2, because there are two connections.
- **Free capacity ($\mathbf{c_{free}}$)**: To estimate the current free capacity, we assume that a new connection or transfer will get a fair share of the maximum capacity. Therefore, we divide maximum capacity by number of concurrent connections. To account for idle connections or connections that do not use their fair share of the capacity due to slow start or being application-limited, we weight the number of connections by the currently utilized capacity.

- **Packet Loss**: Similar to one-way latency, packet loss is not directly observable on hosts. However, protocols that provide reliable transport, e.g., TCP, Stream Control Transmission Protocol (SCTP), and QUIC keep track of which data was actually received by the other end to retransmit lost data. Therefore, this information can be used to infer packet loss for connections that use these protocols. In contrast, as unreliable protocols like User Datagram Protocol (UDP) do not keep track of whether packets were received, we cannot use UDP flows to estimate packet loss.

  In the scope of this work, we consider estimating packet loss in the upstream direction. To do so, we query the counter of segments that the TCP stack deemed lost for each connection and divide this number by the total number of segments sent[3]. For example, in Figure 5.2, the upstream packet loss on $connection_1$ is

[3] For Linux, the number of lost segments is available within `struct tcp_info` as `tcpi_lost` since Linux 3.16, while the total number of segments sent is available as `data_segs_out` from Linux 4.6.

the number of $retransmissions_1$ divided by the number of packets sent on the connection. Note that these loss counters are only estimates, as there is no indication whether retransmits are caused by link errors, congestion, or re-ordering.

In the downstream direction, estimating packet loss is often not possible, as TCP implementations usually do not keep track of lost and re-ordered bytes on the receiver side. Even if they did, with TCP we could only estimate the amount of lost and re-ordered bytes, but have no certainty about how many packets were lost[4]. To complicate matters further, distinguishing between lost and re-ordered bytes/packets is only possible in retrospect. For instance, in case a packet arrives out-of-order, but within one RTT, it can be considered re-ordered. If it arrives later, it can be a considered being a retransmission of a lost packet. Due to these limitations, we currently do not consider packet loss estimates in our access network selection.

- **Wireless characteristics**: As the physical layer characteristics of a wireless link may directly influence latency or capacity on the wireless link, i.e., the first hop of the path, we gather such network performance characteristics in addition to the above latency and capacity estimates. For WiFi, physical layer network performance characteristics are often readily available from the network interface driver, while for cellular, they may not be available.

  We gather the following network performance characteristics for WiFi:

  - **Signal strength**: The RSS with which the last data frame was received.
  - **Modulation rate**: The modulation scheme which the driver used to send the most recently sent data frame, and the modulation which was used for the most recently received frame.
  - **Channel utilization**: The percentage of airtime during which the driver sensed the channel free. Channel utilization is communicated by WiFi Access Points (APs) within QBSS information elements in 802.11 probe response and beacon frames.

Note that the gathered network performance characteristics are most useful to estimate performance for paths to remote hosts that the end-user device is already communicating with. For other remote hosts, estimates may initially be less representative, but become more accurate as the end-user device starts considering traffic to the new host in its estimates as well.

Moreover, paths to different remote hosts may have different performance bottlenecks, e.g., due to congestion on a link in the backbone to one remote host, whereby the same link is not traversed by the path to the other remote host. We may recognize such differences based on high latency variations among connections to different

[4] As QUIC implements loss recovery based on packets instead of byte streams, we could estimate upstream packet loss for QUIC if packet loss counts were made available by the protocol implementation. Therefore, both QUIC packet loss and TCP segment loss can serve as estimates for packet loss.

remote hosts. In such cases, to get more accurate estimates, we may aggregate network performance characteristics per remote host or per destination subnet instead of aggregating them for all connections.

We implement collecting these network performance characteristics in the Socket Intents prototype, see Section 7.3. To see whether the gathered network performance characteristics reflect the network conditions in our evaluation, we perform a feasibility study in a testbed, see Section 8.5.

6

Access Network Selection Policies

Based on application needs expressed as Socket Intents and network performance characteristics we design access network selection policies. An access network selection policy selects which access network to use for each new connection or transfer. Hereby, it may choose a single access network or combine the available access networks using Multipath TCP (MPTCP). If an access network selection policy optimizes its decision based on Socket Intents as well as current network performance characteristics, we refer to it as an Informed Access Network Selection (IANS) policy.

In this section, we first explain the fundamental design of our access network selection policies and, then, cover different IANS policies. Hereby, we first present rule-based IANS policies which match application needs to suitable networks. Then, we present the Threshold Policy, which is designed specifically for Web browsing. Finally, we introduce the Optimist Policy, the Pessimist Policy, and the Selective MPTCP Policy, which are designed for HTTP Adaptive Streaming (HAS).

6.1 Policy Design

Access network selection policies are entities that decide which access network to use for each new connection or transfer. Hereby, access network selection policies can be informed via Socket Intents, recall Chapter 4, and estimates of the current network performance characteristics, see Chapter 5. For details of how we implement these inputs, see Chapter 7.

An access network selection policy makes its decision based on input from the application and produces an output, see Figure 6.1. As input, an application specifies its Socket Intents either for a new connection (A) or for a new transfer (B). In addition to the Socket Intents, the access network selection policy learns the domain name of the remote host that the application wants to communicate with and the remote port. In the case where an application specifies its Socket Intents for a new transfer (B), the access network selection policy also knows about any existing connections to the remote host which may be available for reuse. Therefore, it can take into account the overhead of opening a new connection and penalize access networks on which this would be necessary accordingly. In addition to application needs, the access network selection policy has access to estimates of the current network performance characteristics for each access network, see Section 5.2.

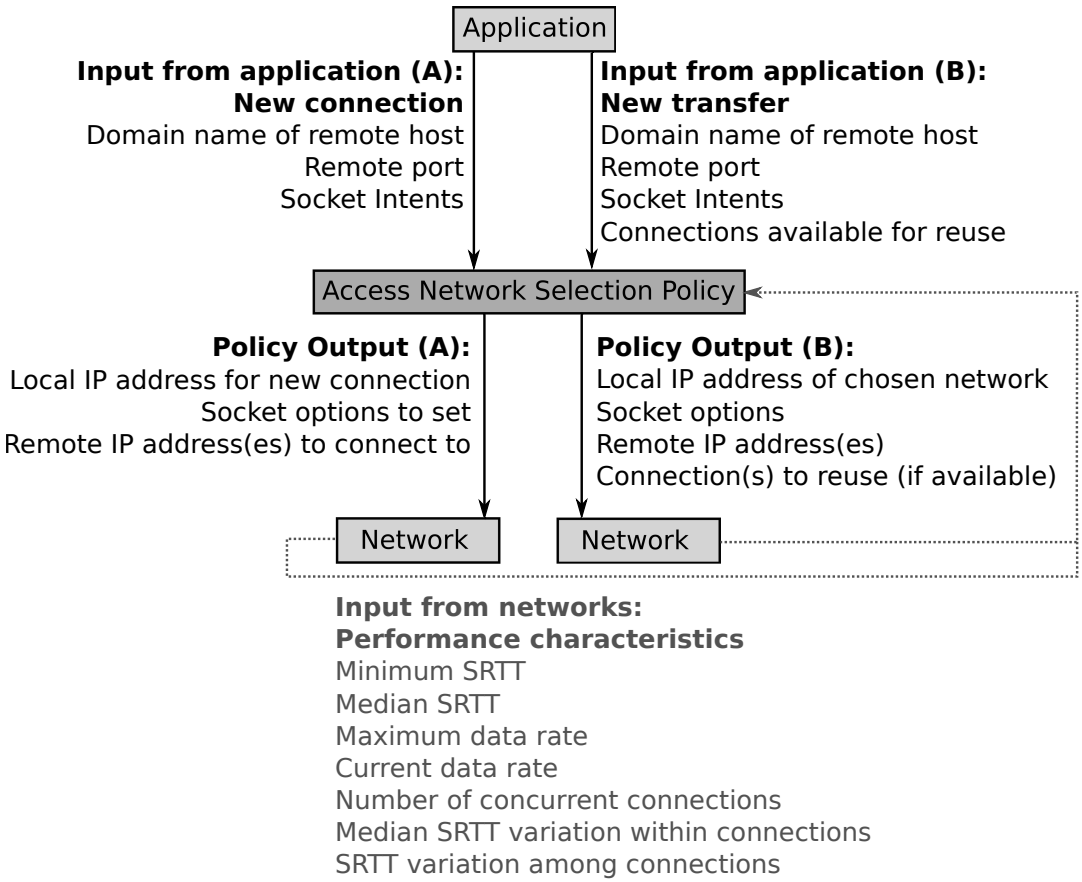


Figure 6.1: Policy model.

To choose an access network for a new connection (A), the access network selection policy returns a local IP address for this connection. Hereby, the IP address belongs to the network interface through which the end-user device can use the chosen access network. Thus, the connection will be established via this network. The access network selection policy can also combine all available access networks using MPTCP by suggesting to set the corresponding socket option for the new connection. In this case, the chosen access network is used for the primary subflow. Then, the MPTCP implementation may establish additional subflows over other access networks without any further interaction with the access network selection policy[1]. Finally, the access network selection policy has to resolve the domain name, rather than relying on the application to do so, because Domain Name System (DNS) resolution depends on access network selection: The same domain name may resolve to different IP addresses for different access networks, e.g., if different Content Delivery Network (CDN) nodes serving the same content are "closer" to the end-user device on each access network. Therefore, the access network selection policy resolves the domain name over the selected access network and then returns the resolved IP addresses. While this prevents the use of a remote IP address optimized for a different access

[1]Note that, within the context of our work, we designed an access network selection policy which explicitly interacts with an MPTCP path manager [74]. However, such interactions are out of scope for this thesis.

network, it also limits caching DNS responses on the end-user device by splitting the cache.

When choosing an access network for a new transfer (B), the access network selection policy may return one or multiple existing connections using the chosen access network to reuse. It also returns the information necessary to open a new connection, similar to (A), for cases in which there are no existing connections available for reuse on the chosen access network or if connection reuse fails. For example, reuse may fail if the remote host has terminated the connection, which an implementation can find out by probing, e.g., using Transmission Control Protocol (TCP) keepalives. In such cases, an application may establish a new connection with the same parameters, i.e., the same local and remote IP address as well as socket options, using the information provided by the access network selection policy.

While per-connection interaction with the access network selection policy (A) may be sufficient for applications whose Intents remain the same during the lifetime of a connection, other applications may need to interact with the access network selection policy more frequently, i.e., for each transfer (B). For example, for Web browsing, each transfer corresponds to a request for a Web resource and different Web resources may have different Intents, e.g., Size to be Received. Here, instead of selecting an access network for the entire connection to a Web server and then placing all transfers to this Web server on the connection, it is necessary to select an access network for each transfer.

Access network selection policies may use simplistic default configurations or they may optimize their decisions informed by Socket Intents and network performance characteristics, in which case we refer to them as IANS policies. IANS policies may (a) match traffic to networks using rules, or (b) perform complex computations to predict performance[2]. For rule-based IANS policies, an application specifies what network performance characteristics to optimize for, e.g., short latency or high downstream capacity. Then, the IANS policy selects a network with the desired network performance characteristics. For Web browsing, rule-based IANS policies are not sufficient because the application does not possess a priori knowledge of whether to optimize each transfer for short latency or high downstream capacity. Instead, the application can set the properties of each transfer, e.g., the Size to be Received. Based on these properties, IANS policies can compute what network performance characteristics to optimize a transfer for. For HAS, while most transfers are capacity-bound, IANS policies benefit from additional knowledge such as the Bitrate Received and the Duration.

6.2 Rule-based IANS policies

To match application traffic, i.e., a new connection or a new transfer, to the most suitable access network, an IANS policy is aware of of Socket Intents and network per-

[2]Other options, such as random choices, are out of scope for this thesis.

formance characteristics. Based on this information, the IANS policy can implement its decision logic to match Socket Intents to network performance characteristics. If such IANS policies match Socket Intents based on a set of rules, they are called rule-based IANS policies. We see the primary use case of rule-based IANS policies as distributing the traffic of different applications across different access networks. Hereby, each application specifies its needs through Socket Intents to take advantage of the most suitable access network for its communication, whereby the IANS policy is able to optimize its decisions across multiple applications. Rule-based IANS policies can select an access network for each new connection or for each new transfer. Selecting a network for each new transfer implies opening multiple connections and distributing transfers across them.

Rule-based IANS policies can apply one or multiple rules matching Socket Intents to an access network with specific network performance characteristics. For example, one or multiple applications may specify the Traffic Category Intent for each new connection or transfer. The Traffic Category implicitly expresses what network performance characteristics to optimize for when selecting an access network, e.g., low latency or high downstream capacity. Here, a rule-based IANS policy can apply the following rules to select an access network:

- If the Traffic Category is set to Query or Control, select the access network with the shortest latency.
- If the Traffic Category is set to Stream, select the access network with the lowest latency variation.
- If the Traffic Category is set to Bulk, select the access network with the highest available downstream capacity.

Alternatively, an application may specify both the Timeliness Intent and the Cost Preference Intent. While the Timeliness indicates whether the application prioritizes short latency and latency variation, the Cost Preference indicates how to balance application needs regarding good performance and low monetary cost. Here, a rule-based IANS policy can apply the following rules to select an access network:

- If the Timeliness is set to Background or the Cost Preference to No Expense: Select the network that does not incur any additional monetary cost (i.e., use a network without metered data plan).
- If the Timeliness is set to Transfer:
  - If the Cost Preference is set to Optimize or to Balance: Select the network that does not incur any additional monetary cost.
  - If the Cost Preference is set to Ignore: Select the network with the shortest latency, even if using this network incurs additional monetary cost.
- If the Timeliness is set to Stream:
  - If the Cost Preference is set to Optimize or to Balance: Select the network with the shortest latency only if it provides a performance benefit.

– If the Cost Preference is set to Ignore: Select the network with the shortest latency, even if using this network incurs additional monetary cost.

As another example, an application may specify the Bitrate Received Intent for a new connection. Here, a rule-based IANS policy can select an access network with a current downstream capacity that is above the Bitrate Received Intent. If multiple access networks fulfill this requirement, the rule-based IANS policy may decide among these networks based on "tiebreaker" rules, e.g., based on Energy Efficiency or short latency. Moreover, the rule-based IANS policy can specify a default access network to be selected in cases in which an application does not specify any Socket Intents or in case the specified Socket Intents do not match any rule.

6.3 Threshold Policy for Web Browsing

To improve the performance of Web browsing using IANS, we design the Threshold Policy to distribute resource loads of a Web page across access networks. The goal of this IANS policy is to shorten the load times of individual resources and, thus, to shorten the overall load time for the page. To achieve short load times for each resource, the Threshold Policy considers resource sizes based on the Size to be Received Socket Intent. In addition to the Size to be Received, the Threshold Policy takes current network performance characteristics into account, i.e., latency and downstream capacity on each network. Depending on the size of a resource, the Threshold Policy optimizes for different network performance characteristics: For loading small resources, whose load times are often latency-bound, it prefers access networks with short latency. For large resources, whose load times are dominated by downstream capacity, it prefers access networks that provide high downstream capacity. We name this IANS Policy the Threshold Policy based on the threshold resource size below which latency dominates and above which downstream capacity dominates. Note that, while designed for the use case of Web browsing, the Threshold Policy can be used for any application that sets the Size to be Received for each transfer.

6.3.1 Threshold Policy Algorithm in Detail

For each individual resource, the Threshold Policy selects the most suitable access network, i.e., the network with the shortest expected load time. First, we describe the fundamental decision algorithm. Then, we provide further details on the latency and capacity calculations in Section 6.3.2 and on the resource load time estimation in Section 6.3.3.

The fundamental decision algorithm of the Threshold Policy is shown in Algorithm 6.1. First, based on the Size to be Received given as size, the Threshold

Algorithm 6.1: THRESHOLD POLICY decision algorithm.

Input: Transfer with size, $\mathsf{use}_{\mathsf{TLS}}$,
Networks $n \in \mathcal{N}$ with $\mathsf{RTT}_{\mathsf{min}_n}$, $\mathsf{RTT}_{\mathsf{median}_n}$, reuse_n, $\mathsf{DRate}_{\mathsf{max}_n}$, $\mathsf{DRate}_{\mathsf{cur}_n}$, conns_n
Output: Network to use for transfer

```
1  net_short ← network with shortest RTT_min
                    // Compute load time components due to latency and capacity
2  t_latency ← getLatencyPart (RTT_min_short, reuse_short, use_TLS)      // See Alg. 6.2
3  t_capacity ← getCapacityPart (DRate_max_short, DRate_cur_short, conns_short, size)
4  if t_latency > t_capacity then                                        // Latency-bound
5  |  return net_short
6  else                                                                  // Capacity-bound
7  |  foreach network n ∈ N do
8  |  |  c_free_n ←getCapacity (DRate_max_n, DRate_cur_n, conns_n)      // See Alg. 6.2
9  |  |  t_load ←predictLoadTime (RTT_min_n, reuse_n, use_TLS, c_free_n, size)
   |  |                                                                 // See Alg. 6.3
10 |  return network with shortest t_load
```

POLICY determines whether this resource is small enough that its load time is dominated by latency, or whether the resource exceeds this threshold, so that its load time is dominated by downstream capacity.

If the resource load is dominated by latency, the resource should be loaded on the network with the shortest latency. Therefore, the THRESHOLD POLICY initially focuses on the network with the current shortest $\mathsf{RTT}_{\mathsf{min}}$, $\mathsf{net}_{\mathsf{short}}$. Here, we use $\mathsf{RTT}_{\mathsf{min}}$ rather than $\mathsf{RTT}_{\mathsf{median}}$ because it estimates a lower bound of the two-way latency to the remote host while being less impacted by current traffic. Based on the resource size and the current network performance characteristics on the shortest latency network $\mathsf{net}_{\mathsf{short}}$, the THRESHOLD POLICY calculates a latency part and a capacity part of resource load time, see Section 6.3.2 for details.

If the latency part exceeds the capacity part, the resource load is latency bound and the THRESHOLD POLICY chooses $\mathsf{net}_{\mathsf{short}}$ due to short latency. If the capacity part exceeds the latency part, the THRESHOLD POLICY predicts the load time of the resource for *all* available access networks, see Section 6.3.3 for details. Finally, the THRESHOLD POLICY chooses the network with the shortest predicted load time.

6.3.2 Latency and Capacity Computations

To determine the threshold of whether a resource load is latency bound or capacity-bound, the THRESHOLD POLICY computes a latency part and a capacity part. The latency part estimates the delay to initiate the resource transfer, e.g., connection setup delay, see Function `getLatencyPart` shown in Algorithm 6.2. This delay depends on whether a new connection has to be established or whether an existing connection to the same remote host can be reused to load the resource, i.e., reuse. If a new

Algorithm 6.2: Latency and capacity computations.

```
1  Function getLatencyPart (RTT, reuse, use_TLS):
2      if reuse then
3          t_latency ← RTT
4      else if use_TLS then
5          t_latency ← 4 · RTT
6      else
7          t_latency ← 2 · RTT
8      return t_latency
9  Function getCapacityPart (DRate_max, DRate_cur, conns, size):
10     c_free ← getCapacity (DRate_max, DRate_cur, conns)
11     t_capacity ← size/c_free
12     return t_capacity
13 Function getCapacity (DRate_max, DRate_cur, conns):
14     usageRate ← DRate_cur/DRate_max
15     c_free ← DRate_max/((conns · usageRate) + 1)
16     return c_free
```

connection has to be established, the connection setup delay depends on whether connection setup includes a Transport Layer Security (TLS) handshake, i.e., $\mathsf{use}_{\mathsf{TLS}}$. The Threshold Policy has this information available because it knows about the destination host and port of the transfer and about existing connections. Accordingly, the Threshold Policy multiplies the RTT by the estimated number of round trips to initiate a resource load by reusing an existing connection or to establish a new TCP connection with or without TLS handshake. The capacity part estimates the time to transfer all bytes of the resource back-to-back, excluding connection setup delay, see Function `getCapacityPart` shown in Algorithm 6.2. This time depends on the Size to be Received, i.e., size, which is divided by the currently available downstream capacity, $\mathsf{c}_{\mathsf{free}}$.

The currently available downstream capacity, $\mathsf{c}_{\mathsf{free}}$, is computed by the Function `getCapacity`[3] shown in Algorithm 6.2. The Threshold Policy uses the current maximum data rate on an access network, $\mathsf{DRate}_{\mathsf{max}}$, as an estimate of the maximum downstream capacity on the access network. It assumes fair sharing of the maximum downstream capacity among all concurrent connections. To split the maximum downstream capacity among concurrent connections, the Threshold Policy considers the number of currently established connections across the access network, conns. However, as connections may be idle, the total number of connections may be higher than the number of active connections currently transferring data. Therefore, the

[3]Note that this is just one possible algorithm to estimate $\mathsf{c}_{\mathsf{free}}$. We tested different variants of the Threshold Policy using different algorithms for $\mathsf{c}_{\mathsf{free}}$, e.g., which counted transfers seen by the policy or which used a threshold to determine whether to consider a network "free" or "busy". In our tests, we found that the algorithm shown here yielded the best results, i.e., shortest load times. Therefore, our final version of the Threshold Policy uses the algorithm shown in Algorithm 6.2.

Threshold Policy weights conns by usageRate, the percentage of the maximum downstream capacity which is currently occupied. The Threshold Policy uses the current data rate on the access network, $\mathsf{DRate}_{\mathsf{cur}}$, as an estimate of the currently occupied downstream capacity. It subtracts $\mathsf{DRate}_{\mathsf{cur}}$ from $\mathsf{DRate}_{\mathsf{max}}$ to estimate free available downstream capacity and then divides this by $\mathsf{DRate}_{\mathsf{max}}$ for the ratio to maximum downstream capacity.

6.3.3 Resource Load Time Estimation

Algorithm 6.3: Predict resource load time over a given network

```
1  Function predictLoadTime (RTT, c, reuse, use_TLS, size):
2      if reuse then
3          t_load ← RTT + (size/c)
4      else                          // Estimate connection setup time and slow start
5          if use_TLS then
6              t_setup ← 3 · RTT
7          else
8              t_setup ← RTT
9          slowStartRounds ← 0
10         size_Chunk ← INITCWND
11         size_maxChunk ← (c · 0.8) * RTT
12         while size_Chunk < size_maxChunk and size > 0 do
               // Emulate one slow-start round which fetches a chunk
13             slowStartRounds ← slowStartRounds + 1
14             size_Chunk ← size_Chunk · 2
15             size ← size − size_Chunk
16         DRate_used ← size_Chunk/RTT
17         t_load ← t_setup + slowStartRounds · RTT + (size/DRate_used)
18     return t_load
```

If a resource load is capacity-bound, the Threshold Policy estimates the resource load times for all available access networks, see Algorithm 6.3. Here, the Threshold Policy uses $\mathsf{RTT}_{\mathsf{median}}$ instead of $\mathsf{RTT}_{\mathsf{min}}$ to get a more realistic estimate of what two-way latency that a capacity-bound resource load experiences on this access network in case of concurrent transfers. Moreover, the Threshold Policy differentiates between resource loads where a connection can be reused and resource loads where a new connection has to be established. If it is possible to reuse a connection over this access network, i.e., reuse is True, the Threshold Policy assumes that only one round trip is necessary to initiate data transfer. To encourage connection reuse, then, the Threshold Policy assumes that all bytes of the resource are transferred using the available downstream capacity. Therefore, it computes the latency and capacity part for reused connections, as explained in Section 6.3.2, but based on $\mathsf{RTT}_{\mathsf{median}}$.

If a new connection has to be established, the Threshold Policy predicts the load time using a model of connection setup delay and TCP slow start. First, the Threshold Policy estimates the connection setup time as a multiple of RTT, depending on whether TLS is used, similar to computing the latency part in Section 6.3.2. The Threshold Policy then estimates the duration of slow start using a model of slow start rounds: Each round takes one RTT and loads a part of the resource of a certain $\mathsf{size}_{\mathsf{Chunk}}$, starting with the initial congestion window, INITCWND. Then, after each new round, $\mathsf{size}_{\mathsf{Chunk}}$ doubles, similar to the multiplicative increase of the congestion window during slow start. Slow start continues until either the entire resource has been loaded, i.e., the sum of chunk sizes has surpassed size, or the $\mathsf{size}_{\mathsf{Chunk}}$ has reached its maximum. The maximum chunk size, $\mathsf{size}_{\mathsf{maxChunk}}$, is the chunk size at which the connection has reached its fair share of the overall downstream capacity. We estimate $\mathsf{size}_{\mathsf{maxChunk}}$ as 80% of the Bandwidth Delay Product to account for fluctuations in downstream capacity. If the maximum chunk size is reached, but there are still bytes of the resource to be loaded, we assume that the rest of the resource is loaded at a rate which corresponds to the chunk size reached after slow start. Therefore, we compute the $\mathsf{DRate}_{\mathsf{used}}$ for the rest of the resource load based on the chunk size. Finally, we sum up $\mathsf{t}_{\mathsf{setup}}$, RTT multiplied by slowStartRounds, and, if applicable, the additional load time for the rest of the resource.

6.3.4 Variant: Threshold Policy with Penalty

Although the Threshold Policy already takes current downstream capacity into account, we design a variant of the Threshold Policy that explicitly accounts for external cross-traffic, the Threshold Policy with Penalty. Here, in addition to considering traffic sent and received by the end-user device the Threshold Policy is running on, the Threshold Policy with Penalty seeks to consider traffic generated by *other* end-user devices on the same access network. For example, this may include other stations on the same wireless link or end-user devices that are using the same bottleneck within an access network. As large amounts of cross-traffic may overload the access network, this network may provide inadequate performance. Therefore, the Threshold Policy with Penalty aims to avoid selecting a network with high current cross-traffic.

To avoid networks with high cross-traffic, the Threshold Policy with Penalty first has to detect the presence of external cross-traffic and then react to it, i.e., take cross-traffic into account when selecting an access network.

To detect external cross-traffic, here, we initially focus on the wireless link of the access network, as wireless links are often the bottleneck [3]. To detect cross-traffic on the wireless link, we use channel utilization, see Section 5.2. We learn channel utilization from a network interface driver, such as a WiFi driver, which reports the percentage of time during which it has sensed the channel busy, i.e., the time in which another station was sending on this channel. A high current channel utilization can indicate that the network is currently subject to external cross-traffic. Alternatively, it may indicate the presence of other sources of interference, e.g., microwaves generating

electromagnetic waves on frequencies which are also used for WiFi. In both cases, while the network interface driver senses the channel busy, the station cannot send or receive traffic through this network. Therefore, we interpret an increase in channel utilization as a decrease in available downstream capacity, regardless of whether this decrease was indeed caused by external cross-traffic or by other influences. Hereby, the Threshold Policy with Penalty considers an access network impacted by cross-traffic on the wireless link if its channel utilization is greater than zero. Based on this information, the Threshold Policy with Penalty adjusts its downstream capacity estimate for the access network, i.e., penalizes the network so it is selected less frequently.

As an alternative design, we consider defining a threshold above which we deem the network overloaded, such as a channel utilization of 60%, which we observe for a busy network in our tests. The reasons for the threshold of 60% instead of 100% are related to overhead and statistical effects when accessing the WiFi channel. As WiFi uses CSMA/CA, there is a sensing period between the transmission of two frames, during which no station is allowed to send, see also Section 2.1.1.1. During such sensing periods, the network interface driver does not sense the channel busy, but still, no station is able to send traffic. We test variants of the Threshold Policy with Penalty using such a static threshold, but find that it fails to adequately capture the fact that even low external cross-traffic may have an impact on available downstream capacity. Therefore, we do not further consider a static threshold between an "overloaded" and a "free" network.

One limitation of using channel utilization is that while it is often readily available for WiFi, it is often not available for cellular. The reason is that cellular modems often expose limited information about network performance characteristics such as channel utilization. Another limitation of channel utilization is that it cannot indicate external cross-traffic outside of the wireless link, e.g., on the uplink between the access network and the Internet.

Because of the limitations of channel utilization, as an alternative, we consider detecting external cross-traffic using latency variation. Latency variation estimates are available for all access networks with existing connections, i.e., they may be available for cellular as well as WiFi. We consider a high latency variation as an indicator of possible external cross-traffic because a high latency variation may reflect an increase or fluctuation in latency on some or all connections. This increase or fluctuation in latency, in turn, may indicate an increase in queueing delay on a bottleneck link, thus, a congested link. We collect two different estimates for latency variation, recall Section 5.2: One estimate is the Smoothed Round Trip Time (SRTT) variation within connections, which reflects fluctuations in SRTT samples of the same TCP connection over time. The other estimate is the SRTT variation among connections, which reflects differences of the SRTTs of multiple TCP connections over the same access network. Our rationale for considering both of these latency variation estimates is that we may get a reliable indication of cross-traffic by combining both estimates. Using latency variation estimates, we consider an access network impacted by external cross-traffic if latency variations are higher than the minimum latency variation

we have recently observed. However, a limitation of using latency variation is that it is impacted by self-induced congestion as well, i.e., congestion caused by the traffic that our end-user device itself is sending or receiving. As the Threshold Policy already accounts for the end-user device's own traffic, this can penalize a busy network twice. Therefore, our Threshold Policy with Penalty focuses on channel utilization as an indicator of external cross-traffic.

After having detected the presence of cross-traffic and/or a high degree of congestion on an access network, we penalize it, i.e., select the network not as often or not at all. For the Threshold Policy with Penalty, we introduce a penalty, i.e., we decrease the downstream capacity estimate of a network where we see congestion proportionally to the indicated congestion. This increases the load time estimate for this access network, thus, the network is selected less after. An alternative approach is to define a cross-traffic threshold above which we consider the network overloaded, therefore, we do not select a network while it is overloaded. We do not follow the latter approach because even with external cross-traffic, we may still be able to utilize an access network, and not utilizing the network at all may limit the usefulness of IANS. We study the feasibility of using channel utilization or latency variation for detecting cross-traffic within the Socket Intents prototype, see Section 8.5.

6.3.5 Application Support for the Threshold Policy

To be able to use the Threshold Policy, an application has to specify the Size to be Received i.e., the number of bytes that the application expects to receive in reply to a sent request, for a new transfer such as a Web resource load. To an application, the Size to be Received of a resource may be available from metadata within the Web page that references the resource. Alternatively, the application can query the Size to be Received. Hereby, the application first issues an HyperText Transfer Protocol (HTTP) HEAD request or an HTTP GET request for the first, e.g., 1000 Bytes[4] of the Resource. As this first request is small, the application can set an Intent to prioritize short latency. The reply to this request includes the size of the resource in the Content-Length header, so the application can then set the Size to be Received for loading the rest of the resource.

Because browsers are complex, fast-moving systems and we want to be flexible to use different browsers with our prototype, we implement Socket Intents support in a client-sided Web proxy, see Section 7.4.1. While this allows us to study IANS, the proxy adds some overhead in the form of an extra RTT for the initial request and an increased server load, since the server may have to answer two requests for a resource.

[4] When choosing this parameter, there is a trade-off between overhead and loading small resources: With a higher number of bytes initially requested, the initial overhead to load the first part increases, but there is also a higher possibility to load small resources in their entirety with the first load. To incur minimal overhead, we find that the size of the first request plus the corresponding HTTP headers should fit within the first packet (usually around 1500 Bytes) or first few packets, and definitely fit within the initial congestion window.

To avoid this overhead, future work should implement Socket Intents in the browser itself.

6.4 Optimist and Pessimist Policy for Video

To improve the performance of HAS, we design two IANS policies which select between single access networks: The Optimist Policy and the Pessimist Policy. The objective of these IANS policies is to select an access network which provides short load times for each transfer, e.g., each video segment, and which avoids excessively long load times that may lead to stalling of the video playout.

As video segments are usually hundreds of Kilobytes to multiple Megabytes in size, the transfers to load these segments are usually capacity-bound, so the IANS policies select the network with the higher downstream capacity. Hereby, both IANS policies predict the load times of each segment on all available networks similar to the Threshold Policy, see Section 6.3.3. For this prediction, the IANS policies use recent downstream capacity estimates. However, as network conditions may vary over time, the available downstream capacity estimates may not always be up to date, especially on networks which have not been recently used. On such networks, the recent downstream capacity estimate may be lower than the achievable downstream capacity due to a lack of traffic that saturates the network. Therefore, the Optimist Policy periodically tries using a network which it has not used recently, but for which it has seen high downstream capacities in the past, thus, shorter "best case" load times. Hereby, the Optimist Policy hopes that the network will be able to provide such high downstream capacities again.

If a network provides insufficient downstream capacity, choosing this network may lead to long load times. This, in turn, may lead to the buffer level within the HAS player decreasing, so it may switch to a lower representation or stall the video playout, which leads to a bad Quality of Experience (QoE). The Pessimist Policy tries to avoid this case by detecting a decrease in downstream capacity, i.e., an increase in predicted "worst case" load time. In cases where the predicted load time on a network exceeds the allowed duration, the Pessimist Policy considers switching to a different network which provides a shorter "worst case" load time.

Both the Optimist Policy and the Pessimist Policy receive the same Socket Intents from the application: First, they learn the Traffic Category of a transfer, i.e., Query for manifest files and initial segments or Bulk for video segments. From this, the IANS policies learn whether the transfer is latency-bound or capacity-bound. For video segments, which are capacity-bound, the IANS policies learn the Bitrate Received of the representation for the next segment selected by the Adaptive Bit-Rate algorithm (ABR) as well as the segment length. From this information, the IANS policy can calculate the expected size of the next segment, which it needs for estimating the load time. Moreover, the IANS policy knows about the Duration, i.e., the current buffer level as the maximum allowed duration of the transfer to avoid stalling. The Optimist Policy uses this information to better assess the risk to

switch to a recently unused network, while the Pessimist Policy assesses the risk to load a segment over a network with a high predicted "worst case" load time.

As IANS policies make a decision after the ABR has chosen the next representation of a video segment to be loaded, the IANS policies attempt to select a suitable network to satisfy the requirement of the ABR, i.e., to select a network via which the segment will load in time. While the IANS policies react to the ABR by attempting to provide short load times for the segment of a size chosen by the ABR, the ABR may also indirectly react to the IANS policy: Selecting the network with short load time for each segment enables a HAS player to accumulate more content in its playout buffer. The current buffer level may be used as an input for the ABR used by the HAS player to adapt to the current network conditions. Therefore, an increase in buffer level may lead to switching to a higher representation of the content, i.e., a representation which requires higher downstream capacity, but which also yields a higher video quality. This has the potential to create a feedback loop between the ABR and the IANS policy. However, the control loops are decoupled because the IANS policy decision does not override or directly influence the ABR decision. The only influence the IANS policy decision has on the ABR is an improve in network conditions, which can be achieved by others means as well, e.g., an increase in Received Signal Strength (RSS).

6.4.1 Optimist and Pessimist Policy Algorithm in Detail

For each transfer, i.e., each video segment as well as the initial manifest file, the Optimist Policy and Pessimist Policy select an access network according to Algorithm 6.4. For transfers with the category set to QUERY, such as manifest files and initial segments, which are small, the IANS policies select the access network with the shortest current latency. For transfers with the category set to BULKTRANSFER, i.e., the video segments, the IANS policies compute the expected size of the segments according to the representation bitrate and the length of one segment.

The Optimist Policy and the Pessimist Policy predict load times on all available access networks similar to the Threshold Policy, see Algorithm 6.3. While the Threshold Policy uses a single estimate of the currently available downstream capacity, the Optimist Policy and Pessimist Policy use four downstream capacity estimates on different time scales to account for downstream capacity changes over time and to compute different load time estimates:

- c_{short}: The maximum $DRate_{max}$ of the last 1 second to compute t_{short}
- c_{mid}: The maximum $DRate_{max}$ of the last 10 seconds to compute t_{mid}
- c_{long}: The maximum $DRate_{max}$ of the last 60 seconds to compute t_{long}
- $c_{verylong}$: The maximum $DRate_{max}$ of the last 600 seconds to compute $t_{verylong}$

The IANS policies first compare the t_{mid} estimates for all networks to determine a candidate network, net_{cand}, on which the transfer is likely to complete in a short time. However, as the t_{mid} may be out of date, the IANS policies then take the

Algorithm 6.4: Optimist and Pessimist Policy decision algorithm.

Input: Transfer with category, $\text{bitrate}_{\text{segment}}$, t_{segment}, t_{buffer}, use_{TLS}
Networks $n \in \mathcal{N}$ with $\text{RTT}_{\text{min}_n}$, reuse_n, c_{short_n}, c_{mid_n}, c_{long_n}, c_{verylong_n}
Output: Network to use for transfer

```
if category = QUERY then
    return network with shortest RTT_min
else if category = BULKTRANSFER then
    size ← bitrate_segment · t_segment                    // Estimate size of this segment
    foreach network n ∈ 𝒩 do
        t_short_n ← predictLoadTime (RTT_min_n, c_short_n, reuse_n, use_TLS, size)
            // worst case load time based on c_short_n of last 1 second
        t_mid_n ← predictLoadTime (RTT_min_n, c_mid_n, reuse_n, use_TLS, size)
            // expected load time based on c_mid_n of last 10 seconds
        t_long_n ← predictLoadTime (RTT_min_n, c_long_n, reuse_n, use_TLS, size)
            // long term estimate based on c_long_n of last 1 minute
        t_verylong_n ← predictLoadTime (RTT_min_n, c_verylong_n, reuse_n, use_TLS, size)
            // best case based on c_verylong_n of last 10 minutes
    net_cand ← network with shortest t_mid
    if Optimist Policy then                                // See Alg. 6.5
        return optimistPolicy (net_cand, t_segment, t_buffer, t_long_cand, t_verylong_cand)
    else if Pessimist Policy then                          // See Alg. 6.6
        return pessimistPolicy (net_cand, t_segment, t_buffer, t_short_cand, t_mid_cand,
            t_long_cand)
```

other downstream capacity estimates into account: The Optimist Policy considers switching to an alternative network, net_{alt}, with a better longer-term estimate, see Section 6.4.2. The Pessimist Policy determines whether the short term estimate on the net_{cand} is too long, in which case it considers switching to a net_{alt} which it deems "safer", see Section 6.4.3.

6.4.2 Optimist Policy: Considering an Alternative Based on Best Case

When an access network has not been recently used for a large transfer, its recent downstream capacity estimate may be outdated, i.e., below the actual available downstream capacity. Therefore, after the Optimist Policy has determined a net_{cand} to use based on the t_{mid} load time estimate, it considers switching to a net_{alt} according to Algorithm 6.5. Hereby, it considers the "best case" load time using the maximum downstream capacity estimates of the last 600 seconds (10 minutes), i.e., the t_{verylong}, for the load times on all available networks. If the net_{cand} also yields the better "best case" load time, the Optimist Policy stays with this network. Otherwise, if a different net_{alt} has a shorter "best case" load time, the Optimist Policy switches to this network in the following cases: If playout has not started yet, it tries out the alternative network because there is no risk of intermediate video stalling. If

Algorithm 6.5: Optimist Policy: Consider to switch networks.

```
1  Function optimistPolicy (net_cand, t_segment, t_buffer, t_long, t_verylong):
2      net_alt ← network with shortest t_verylong
       // Compare to net_cand with shortest t_mid
3      if t_buffer = 0 and net_alt not used for last segment then
4          return net_alt                          // Playout not started yet -- safe
5      if net_alt not used for last 3 segments then
6          if t_long_alt < 2/3 · t_buffer then
7              return net_alt                                              // Safe
8          else if t_long_cand > 2/3 · t_buffer and t_long_alt < t_long_cand then
9              return net_alt                          // Not safe, but better
10         else if net_alt not used last 10 segments then
11             return net_alt                  // Not used recently, try it
12     return net_cand                          // If we have not switched
```

the alternative network has not been used for at least three segments[5] and yields an acceptable t_{long} (using the downstream capacity estimate of the last 60 seconds), i.e., the load time on this network is below $\frac{2}{3}$ of the buffer level[6], it deems this network "safe" to try. Otherwise, if the t_{long} on the net_{cand} exceeds $\frac{2}{3}$ of the buffer level as well, neither network is "safe" to use, but the Optimist Policy picks the net_{alt} if it has a shorter t_{long}. If the net_{alt} was not picked for more than 10 segments[7], the Optimist Policy selects this network to give it a chance, as its t_{long} may be outdated like its t_{mid}. Finally, if the Optimist Policy has not decided to switch to the net_{alt}, it selects the net_{cand} to use.

6.4.3 Pessimist Policy: Considering an Alternative Based on Worst Case

If the load time of a transfer exceeds the buffer level, this will likely lead to stalling of the playout. If the load time exceeds the segment duration, this may result in a decrease of the buffer level within the player and may mean that the selected network is unable to sustain the current representation, i.e., quality level. Based on these two concerns, the Pessimist Policy first considers how likely these issues are to occur on the current net_{cand}, and then considers switching to an net_{alt} if it deems these issues less likely on this network.

To achieve this, the Pessimist Policy looks at the t_{worst} using the t_{short} of the last second, see Algorithm 6.6. If the t_{worst} on the net_{cand} is longer than either the buffer level t_{buffer} or the segment duration $t_{segment}$, it switches to an net_{alt} in the following cases: If the t_{worst} on the net_{alt} is shorter, the Pessimist Policy deems this net_{alt}

[5] We choose this parameter based on the segment duration of 4 seconds, which we use in our evaluation, such that the duration of three segments exceeds the t_{mid} of 10 seconds.

[6] We compare the load time estimate to $\frac{2}{3}$ of the buffer level to have a safety margin.

[7] We choose this parameter based on the segment duration of 4 seconds, so the load times for 10 segments exceed our *longtermEstimate* of 60 seconds by a factor of 1.5.

Algorithm 6.6: Pessimist Policy: Consider to switch networks.

```
1  Function pessimistPolicy (net_cand, t_segment, t_buffer, t_short, t_mid, t_long):
       // net_cand is network with shortest t_mid
2      t_worst ← t_short_cand                          // If not available, use t_mid_cand
3      if t_worst > t_buffer or t_segment then                          // Be concerned
4          net_alt ← Network with shortest t_short
5          if t_short_alt < t_buffer then
6              return net_alt                                           // safer to use
7          else if net_cand used for last segment and t_worst > 4/3 · t_buffer then
               // t_worst likely accurate: Use net_alt if faster for either estimate
8              if t_short_alt < t_short_cand then
9                  return net_alt
10             else if t_long_alt < t_long_cand then
11                 return net_alt
12     return net_cand                    // We are not concerned or found no net_alt
```

safer to use. The Pessimist Policy considers both estimates because the t_{short} may be out of date on a network that was not recently used. If, on the contrary, the net_{cand} was used for the last segment, the Pessimist Policy considers the t_{worst} more likely to be accurate. If, then, the t_{worst} is considerably longer than t_{buffer}, i.e., $\frac{4}{3}$ times longer, the Pessimist Policy is more ready to switch: It switches if the net_{alt} provides either a better t_{short} or t_{long}[8]. If neither of the estimates is better for the net_{alt}, the Pessimist Policy stays with the net_{cand}.

6.5 Selective MPTCP Policy for Video

In addition to the Optimist Policy and Pessimist Policy, we experiment with an IANS policy that selectively enables MPTCP for some transfers according to Algorithm 6.7. As MPTCP provides the most benefits for large transfers, the Selective MPTCP Policy only enables MPTCP when the category is set to BULKTRANSFER. Moreover, the Selective MPTCP Policy only enables MPTCP when sufficient downstream capacity is available on all networks because we observe that otherwise, the overhead of MPTCP connections and saturating the congestion window may overwhelm a network with insufficient downstream capacity. Determining how much downstream capacity is sufficient is non-trivial. As a first estimate, we compare the available downstream capacity, i.e., the c_{mid}, to the transfer size, size, of the bulk transfer. For our tests with HAS, we compute the estimated size of the next segment, size, from the next representation bitrate and the segment duration, similar to the Optimist Policy and Pessimist Policy. Then, we enable MPTCP if the downstream capacity is above a fraction of the size, depending on the current buffer level. Hereby, we select the network with the shortest latency, net_{short},

[8] The net_{cand} is initially selected based on having the shortest t_{mid}, thus, the net_{alt} cannot have a shorter t_{mid}.

Algorithm 6.7: Selective MPTCP Policy decision algorithm.

Input: Transfer with $\mathsf{category}$, $\mathsf{bitrate}_{\mathsf{segment}}$, $\mathsf{t}_{\mathsf{segment}}$, $\mathsf{t}_{\mathsf{buffer}}$, $\mathsf{use}_{\mathsf{TLS}}$
Networks $n \in \mathcal{N}$ with $\mathsf{RTT}_{\mathsf{min}}$, reuse, $\mathsf{c}_{\mathsf{short}}$, $\mathsf{c}_{\mathsf{mid}}$, $\mathsf{c}_{\mathsf{long}}$, $\mathsf{c}_{\mathsf{verylong}}$
Output: Network to use for transfer

$\mathsf{net}_{\mathsf{short}} \leftarrow$ network with shortest $\mathsf{RTT}_{\mathsf{min}}$
if $\mathsf{category} = \mathsf{QUERY}$ **then**
    return $\mathsf{net}_{\mathsf{short}}$ // No MPTCP
else if $\mathsf{category} = \mathsf{BULKTRANSFER}$ **then**
    $\mathsf{c}_{\mathsf{min}} \leftarrow$ lowest $\mathsf{c}_{\mathsf{mid}}$ // Last 10 seconds
    $\mathsf{size} \leftarrow \mathsf{bitrate}_{\mathsf{segment}} \cdot \mathsf{t}_{\mathsf{segment}}$
    if $\mathsf{t}_{\mathsf{buffer}} > 10$ and $\mathsf{c}_{\mathsf{min}} > \mathsf{size}/8$ **then** // High $\mathsf{t}_{\mathsf{buffer}}$: Be risky
        return all networks using MPTCP with first subflow on $\mathsf{net}_{\mathsf{short}}$
    else if $\mathsf{t}_{\mathsf{buffer}} \leq 10$ and $\mathsf{c}_{\mathsf{min}} > \mathsf{size}/4$ **then** // Low $\mathsf{t}_{\mathsf{buffer}}$: Be prudent
        return all networks using MPTCP with first subflow on $\mathsf{net}_{\mathsf{short}}$
    else // Insufficient capacity for MPTCP
        return network with highest $\mathsf{c}_{\mathsf{mid}}$ // No MPTCP

for the primary subflow, which is consistent with the default scheduler for MPTCP on Linux, the minRTT scheduler. Otherwise, we use only the network with the highest downstream capacity without any MPTCP to avoid overloading the lower downstream capacity network. While the Selective MPTCP Policy focuses on avoiding network overload due to low downstream capacity, future versions could also take asymmetric latency on the networks into account, as MPTCP has been shown to have issues with such scenarios. We do not consider this case in the initial version of this policy, as we see asymmetric latencies as an issue which should be addressed by the MPTCP scheduler.

7

Socket Intents Prototype

We implement Informed Access Network Selection (IANS), including Socket Intents, network performance characteristics, and access network selection policies, within the Socket Intents prototype. Our prototype consists of about 17k lines of C code and its source code is publicly available under BSD License[1].

First, we describe the architecture of the prototype. Then, we cover the Application Programming Interfaces (APIs) that the prototype provides for applications to specify their Socket Intents for a new connection or transfer. Furthermore, we detail how the prototype collects network performance characteristics and how it realizes access network selection policies. Finally, we present how we implement Socket Intents support within a client-side Web proxy and a video player which supports HTTP Adaptive Streaming (HAS).

7.1 Prototype Architecture

The Socket Intents prototype consists of two main components, see Figure 7.1: The Socket Intents library, which enables applications to express their Socket Intents, and the Multi Access Manager (MAM), which collects network performance characteristics and hosts the access network selection policy.

[1] https://github.com/fg-inet/socket-intents/

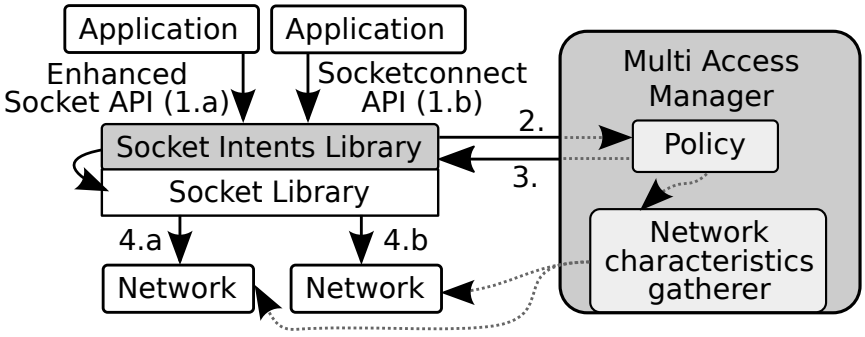


Figure 7.1: Architecture of the Socket Intents prototype.

To allow applications to express their Socket Intents on a per-connection and per-transfer basis, the Socket Intents library implements multiple APIs: Applications can either initiate a new connection through the Enhanced Socket API (1.a) or a new transfer through the Socketconnect API (1.b). Then, the Socket Intents library, which implements both Socket Intents APIs, queries the access network selection policy within the MAM (2). As the MAM continuously gathers network performance characteristics, the access network selection policy can access these network performance characteristics as well as the Intents. Based on all available information, the access network selection policy decides which access network(s) to use, i.e., which local address to bind the new connection to or which existing connection to re-use. Then, the access network selection policy communicates this decision back to the Socket Intents library (3). Finally, the Socket Intents library applies the decision (4a/b), e.g., by binding the new connection to the chosen local address.

To make the prototype easily extensible and portable across Operating Systems (OSes), we implement it in user space. As we want all applications to be able to benefit from IANS without having to replicate access network selection within each application, we implement access network selection outside of the context of a single application, but within an access network selection policy. Hereby, we place the access network selection policy as a central instance on the host, which can be queried by the Socket Intents library on behalf of any application. As access network selection can be useful for different use cases and in different contexts, our prototype is not specific to a single access network selection policy. Instead, we design access network selection policies as exchangeable modules which are running within the context of the MAM. The MAM can dynamically load a suitable access network selection policy based on the current context of the end-user device. Because access network selection policies need up-to-date network performance characteristics based on continuous observation of existing traffic, the MAM is a constantly running component, while access network selection policies are exchangeable. Therefore, the MAM continuously gathers network performance characteristics of the available access networks and makes this information available to the currently loaded access network selection policy.

7.2 Socket Intents APIs

The Socket Intents prototype implements multiple APIs for applications to specify their Socket Intents for a new connection or transfer. We design our Socket Intents APIs on top of the Socket API because the Socket API is the de-facto standard networking API on most OSes and because we want our prototype to be portable across OSes. In this thesis, we present two API variants[2]: An API that allows to

[2]The Socket Intents prototype implements a total of four APIs. In addition to the two variants presented in this thesis, there is an API based on getaddrinfo and an asynchronous variant of our Socketconnect API based on `select`. In the getaddrinfo API, an application obtains information about what IP addresses to use for a new connection from a modified `getaddrinfo`. In the asynchronous Socketconnect API, the application requests a socket to use for a new transfer similar to the synchronous Socketconnect API presented in this thesis. However, the asynchronous API

express Socket Intents for individual connections, see Section 7.2.1, and an API that allows to specify Socket Intents for individual transfers, see Section 7.2.2.

7.2.1 Socket Intents Per Connection: Enhanced Socket API

To allow applications which are using the Socket API to benefit from IANS, we design an enhanced version of the Socket API. In this API, an application can set one or multiple Socket Intents for each new connection, i.e., each socket that it creates. This allows the prototype to choose an appropriate access network for the new connection.

For each created socket, the application can set its Socket Intents using `setsockopt`. Hereby, we define a new socket option level for Intents with new socket options. For each socket, Intents are stored within a connection context, whereas all calls that belong to the same connection reference the same context. Our rationale for adding an explicit reference to the connection context to every call is that we need it to link Socket API calls to Domain Name System (DNS) resolution, as `getaddrinfo`, by itself, does not include a reference to a socket or any other reference to a connection. The reason for including DNS resolution in our API and linking it to the other calls is that we have to resolve domain names over the same access network as the connection will use, recall Section 6.1.

7.2.2 Socket Intents Per Transfer: Socketconnect API

To allow applications to specify their Socket Intents for individual transfers, the Socket Intents prototype implements the Socketconnect API. Here, we integrate DNS resolution and connection establishment within a single call, `socketconnect`. Before a transfer, the application calls socketconnect with a domain name, a destination port, and a list of Socket Intents. The API returns a socket which is connected to a remote host resolved from this domain name. The connection can either be newly established or be reused from the pool of available connections. The application can then use this socket for its transfer and once it is done, the application releases this socket to return it to the pool of available connections using `socketrelease` or closes the connection using `socketclose`.

We choose to use sockets as an abstraction for connections to ease the integration of our API into existing applications. The application calls socketconnect either with an existing socket, which represents the implicit pool of all connections to the same destination or with "-1" to explicitly request a newly established connection. The API then informs the application whether the returned socket corresponds to a new connection or a reused connection. This allows the application to carry out

uses a non-blocking `connect` call and the application has to check whether the socket is ready for read/write operations using a modified `select`. As we do not use these APIs for this thesis, we do not provide further details here.

any necessary initialization for a new connection, e.g., to perform a Transport Layer Security (TLS) handshake, which it does not need for an existing connection.

Beneath the API, the Socket Intents library manages the pool of available connections per application, remote domain name, and service. When the application calls socketconnect, the library checks whether there are currently unused open sockets in the pool. It then sends this information to the access network selection policy, along with the Socket Intents of the new transfer. The access network selection policy then either chooses to reuse one of these connections, i.e., an existing socket, or it provides information to open a new connection, i.e., a new socket. As connections to the same remote host and port can be used interchangeably for stateless application layer protocols such as HyperText Transfer Protocol (HTTP)[3], the prototype groups all such connections into socket sets. Sockets in the same set can be bound to different local addresses, thus, use different access networks.

7.3 Implementing Access Network Selection

To select an access network for a new connection or transfer, the Socket Intents prototype calls an access network selection policy, which is hosted within the MAM. The MAM runs continuously and collects information on which access networks are currently available and collects their network performance characteristics. If provides this information to the access network selection policy.

7.3.1 Multi Access Manager

The MAM is the component of the Socket Intents prototype that hosts the access network selection policy and provides it with network performance characteristics. After startup, the MAM collects information on what access networks are currently available. Hereby, it queries the OS regarding the locally available network interfaces. This results in a list of local interfaces, i.e., a list of access networks, assuming that every network interface is connected to a different access network. Furthermore, the MAM queries what IP addresses are assigned to these interfaces to see which interfaces are available for use. As IP addresses on the same interface may change, e.g., with temporary IPv6 addresses, for each interface, the MAM stores not only individual IP addresses, but aggregates them by network prefixes. Moreover, the MAM needs to distinguish access network by IP prefix when collecting network performance characteristics, where it matches statistics for Transmission Control Protocol (TCP) connections to the corresponding interface. In addition, the MAM reads a configuration file which specifies which access network selection policy to load. Furthermore, the configuration file specifies which of the configured IP prefixes, i.e., access networks to take into account. For each considered access network the configuration file may

[3]The current prototype supports HTTP/1.1. However, Socket Intents support can also be implemented in, e.g., an HTTP/2 library or a Quick UDP Internet Connection (QUIC) implementation

specify DNS configuration and additional per-network information, e.g., which network to use by default. The MAM then initializes the access network selection policy with this information. To be flexible, e.g., to use different access network selection policies without having to restart the MAM, thus, losing the state of the current network performance characteristics, the MAM is able to change access network selection policies while it is running.

After initialization, the MAM starts gathering network performance characteristics on all access networks, i.e., all interfaces and network prefixes. Then, the MAM runs as a service available to all applications on the host via a Unix domain socket. Through this socket, the Socket Intents library sends the Intents for new connections or transfers and receives the decision from the access network selection policy in return. To answer such requests asynchronously, the MAM runs an event loop using libevent. We design the MAM to not keep state per request to be able to scale, e.g., when answering requests from many applications. However, some access network selection policies do keep state internally.

7.3.2 Collecting Network Performance Characteristics

To be able to provide up-to-date information on the current network performance characteristics to the access network selection policies, recall Chapter 5, the MAM collects and stores such information by periodically querying different parts of the networking stack. The query interval is configurable. We test different callback intervals, e.g., 1 s, 200 ms, 100 ms, or 50 ms. While a long interval risks using outdated values in the access network selection policy, a short interval results in more computational overhead due to more frequent callbacks. As our experiments require policy decisions on the timescale of tens to hundreds of microseconds, we use a callback interval of 100 ms.

For Smoothed Round Trip Times (SRTTs) and SRTT variations, the MAM queries the TCP stack through the Netlink API for all current TCP connections. It only includes TCP connections that are not in SYN-SENT, SYN-RECV, TIME-WAIT or CLOSE state, as connections in these states do not offer SRTT samples. To aggregate SRTTs for a specific access network, the MAM then filters the connections by the network prefix configured on this access network. To get data rates, the MAM reads network interface counters. It stores them and aggregates them, e.g., by computing the maximum observed upstream and downstream data rate. Hereby, the MAM considers samples during a configurable time period. Here, an interval of 10 minutes has shown to yield good results. While longer intervals make it more likely that any traffic was present at the time, less recent results may also have become outdated in the meantime because available downstream capacity changed. For wireless network performance characteristics, the MAM periodically queries the WiFi driver once per callback interval. Here, a higher query interval may increase energy consumption if it prevents a driver from going into power save mode. We only get nonzero network performance characteristics for access networks on which the host sees traffic. We use a default value of zero for network performance characteristics that are unavailable.

In this case, these values are not taken into account by our access network selection policies.

7.3.3 Access Network Selection Policy Implementation

The Socket Intents prototype implements access network selection policies, see Section 6.1. As there may not be a single optimal access network selection policy for every scenario, we implement access network selection policies as exchangeable modules which are loaded by the MAM according to its configuration file. An access network selection policy module implements a set of callback functions, which are called to select the access network for each new connection or transfer. Additionally, the access network selection policy implements callbacks for initialization and teardown. Upon initialization, the access network selection policy gets a list of access networks from the MAM augmented with information from the configuration file. The access network selection policy stores state per access network, e.g., whether an access network is the default network. For teardown, e.g., if MAM is terminated or if the access network selection policy module is exchanged, the access network selection policy then has to clean up any internal state.

After being initialized, the access network selection policy is ready to select an access network for connections or transfers. In each case, the access network selection policy gets a callback for an application request, which provides the domain name of the remote host, the remote port, and the Socket Intents for the new connection or transfer. Additionally, for per-transfer access network selection, the request may include a list of connections available for reuse. In addition to the input from the application, the access network selection policy has network performance characteristics available for all available access networks on which the host has recently seen traffic. Based on this information the access network selection policy selects an access network, see Chapter 6. Hereby, the access network selection policy chooses a local IP address for binding the new connection, which is configured on the interface connected to the chosen access network. Alternatively, the access network selection policy can reuse an already established connection over the access network by choosing one of the available sockets. After selecting an access network, the access network selection policy resolves the domain name over this network. It resolves the names asynchronously and keeps separate DNS configurations for different access networks using libevent's asynchronous getaddrinfo.

7.4 Applications Supporting Socket Intents

To benefit from IANS using the Socket Intents prototype, an application must use one of the Socket Intents APIs, e.g., the APIs from Section 7.2. For Web browsing, we wrote a Web proxy which uses the Socketconnect API to set the Size to be Received Intent for each Web resource. For HTTP Adaptive Streaming (HAS), we modified a video player to use the Socketconnect API for each segment.

7.4.1 Web Proxy

To enable IANS for Web browsing, we set the Size to be Received Socket Intent for each individual transfer, i.e., each resource. Because browsers are complex, fast-moving systems and we want to be flexible to use different browsers with our prototype, we implemented Socket Intents support in a Web proxy, thus, enabling any browser to use Socket Intents.

We wrote an HTTP proxy which supports the Socketconnect API and provides the Size to be Received for each resource to be loaded. To acquire the size of an individual resource, the proxy performs a two-step download: It first fetches the first 1000 Bytes[4] while setting the Content-Range header. If the resource size is 1000 Bytes or smaller, the proxy already receives the full resource with this first request. Otherwise, it receives the first part and knows the size of the rest of the resource from the Content-Length header[5]. Thus, the proxy can set the Size to be Received Socket Intent for the IANS policy to select a suitable access network for loading the rest of the resource. Using the Socketconnect API allows the proxy to reuse TCP connections wherever possible, i.e., when loading another resource from the same remote host. This enables the proxy to benefit from IANS on a per-request basis without delaying loading small resources.

As most Web pages are HTTPS-based and do not allow unencrypted HTTP[6], our proxy supports connecting to remote Web servers via TCP as well as TLS. TLS support is realized using the openssl library. We store TLS state per local socket within the proxy.

7.4.2 Video Player

To enable IANS for video streaming such as HAS, we implement Socket Intents support within a video player using Dynamic Adaptive Streaming over HTTP (DASH). We modify the GPAC player[7] version 0.7.2-DEV. We choose GPAC because it is written in C, which enables it to seamlessly use our APIs and because it provides better documentation than other players.

[4] The size of the initial chunk is configurable and presents a tradeoff between putting more load on a short latency network to load initial chunks for all resources and loading more small resources with the initial chunk. We first test an initial sizes of 4000 Bytes and find that it results in too much load on the low capacity network in our experiments. We do not observe this problem for 1000 Bytes, while 1000 Bytes of content fits within a single packet if the HTTP headers are around 500 Bytes or less, which we observe for the Web pages we test first.

[5] Some Web servers do not set a Content-Length for some resources. In such cases, an IANS policy does not know the size, but may, e.g., selects the shorter latency network by default.

[6] In April 2019, 78% of all page loads using Firefox use HTTPS, see `https://letsencrypt.org/stats/#percent-pageloads`.

[7] GPAC is an open source cross-platform multimedia framework maintained by Telecom Paris Tech, see `http://www.gpac.io/`. GPAC implements a video player called MP4Client on Linux and Osmo4 on other platforms. Addtionally, GPAC contains a segmenter called MP4Box, which converts media into different formats, a tool called DashCast to create DASH conforming streams, and other tools.

GPAC includes an Adaptive Bit-Rate algorithm (ABR), which select the representation of the next segment to be loaded, based on the download rate of the previous segment and/or the current status of the video playout buffer. In particular, GPAC implements multiple different ABRs such as BBA-0 [63] and BOLA [64].

Within the GPAC player, we implemented support for the Socketconnect APIs to enable it to use IANS on a per transfer basis, i.e., for each video segment. In particular, we add the socketconnect call to the networking module and then provide the resulting socket, i.e., the connection over the chosen access network, to an internal data structure in GPAC called the download manager. For each video to be played, the player has to first load a manifest file and initial segments, which are small. For these transfers, we set the Traffic Category to Query so that the access network selection policy prioritizes for short latency. When loading the actual video segments, we set the Traffic Category to Bulk, to differentiate them from the manifest files and the initial segments. The video segments are typically at least hundreds of kilobytes in size, and, therefore, their load times are often dominated by high downstream capacity instead of short latency. However, there are different representations and the sizes of the segment depends on the chosen representation. We provide additional information to the access network selection policy by setting the Bitrate Received Socket Intent to the bitrate of the next segment as selected by the ABR. Moreover, we use the status of the playout buffer of the video player to set the Duration Socket Intent. This is the maximum duration that a segment load should take while avoiding stalling the video playout. We have all of this information readily available within the video player.

8

Impact of Informed Access Network Selection on Application Performance

We evaluate the benefits of Informed Access Network Selection (IANS) for Web browsing and HTTP Adaptive Streaming (HAS) using the Socket Intents prototype. First, we outline our methodology, including network scenarios, workload, course of experiments, and performance metrics for both Web and HAS. Then, we present the results of our feasibility study, which shows how accurately our prototype estimates network performance characteristics in our setup. Next, we evaluate the benefits of IANS for Web performance, both focusing on the asymmetric network scenario and using a systematic evaluation of scenarios with different downstream capacity and latency. Finally, we show how IANS improves HAS.

8.1 Network Scenarios

Here, we outline our methodology for studying the benefits of IANS for Web performance and HAS. To enable a systematic evaluation we use a testbed with mirrored Web pages and video workload where we have full control over the network performance characteristics. In addition, for Web, we evaluate the achievable speedups "in the wild" by accessing Web pages via the Internet. For HAS, we do not use servers "in the wild", as a HAS workload is usually hosted on a single server, while Web pages are spread across multiple servers. As we host our HAS workload on a single server similar to a setup "in the wild", while also emulating variable capacity on a bottleneck link in our testbed, loading the video content from a server "in the wild" promises no further insights.

Figure 8.1 shows our testbed, which represents a scenario where the access network is the performance bottleneck. We have full control over almost all components, namely a client, a traffic shaper, a Web server, and two network links, a wired and an 802.11 wireless link. To include the effects of actual WiFi we realize the wireless link via a WiFi Access Point (AP). Yet, to limit side effects we use a stationary AP on an otherwise unoccupied 5 GHz channel.

The client is connected to the traffic shaper via two access networks: To network 1 via a 1 Gbit/s Ethernet wired link with less than 1 ms delay, and to network 2 via the 802.11 wireless link. The wired link adds minimal delay and no congestion,

as it directly connects the client to the shaper. The wireless link adds delays and congestion related to WiFi. With these two access networks, the client can either load the Web pages mirrored on the Web server via each of the networks individually or it can bundle the downstream capacity of both networks using IANS or Multipath TCP (MPTCP).

We focus on a scenario with two access networks because it represents the realistic case of an end-user device having both WiFi and cellular available to use. Here, our goal is to explore the impact of different network performance characteristics in cases where it makes sense to use different networks for different transfers. Therefore, we do not consider trivial cases in which IANS would use the third network for all transfers, if it outperforms the other two networks, or cases in which IANS would not use the third network at all, if it does not provide better network performance characteristics than the other two networks. Moreover, the combination of more than two networks would render our experiment setup much more complex due to more combinations of parameters. However, our IANS policies support selecting between more than two networks, so our results can be generalized to scenarios with more than two networks for cases in which at most two networks are actually utilized.

To emulate different network scenarios in our testbed, we realize delay and downstream capacity restrictions via a shaper. Note, the shaper clearly dominates the network performance characteristics of network 1 while network 2 also sees the effects of WiFi. Cross-traffic can be imposed on the WiFi network and is realized by using another client which sends traffic on the same channel. We impose both a constant load of cross-traffic which fully utilizes the WiFi using an iperf User Datagram Protocol (UDP) flow and a self-similar load of Transmission Control Protocol (TCP) flows to fully utilize the shaper link using Harpoon [75][1].

We emulate different network scenarios for Web browsing and HAS, see Table 8.1. For Web, we follow a factorial design approach across different values for latency and

[1]We configured Harpoon to generate TCP flows with an average total throughput similar to the shaped downstream capacity, whereby the file sizes follow a Pareto distribution with alpha=1.2 and shape=1500 bytes and the inter-connection times follow an exponential distribution with a mean of one second.

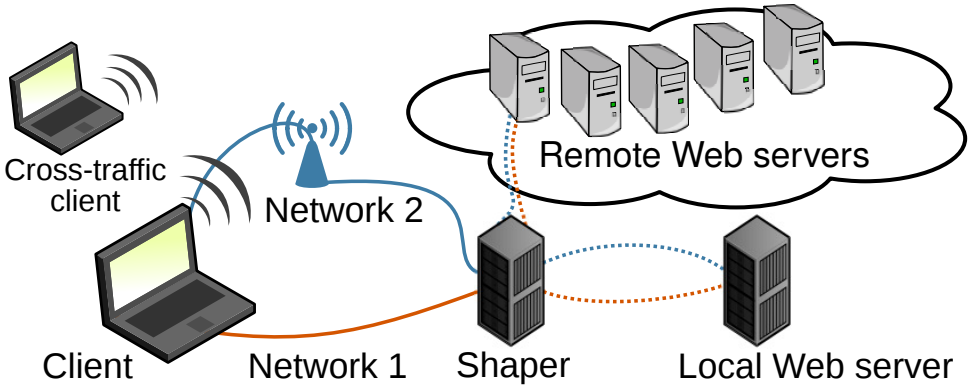


Figure 8.1: Testbed setup.

downstream capacity, see Table 8.2a. Our rationale for this approach is to cover a wide range of network performance characteristics which influence load times: While latency dominates load times for small resources, downstream capacity dominates load times for large resources. As small and large resources may also be influenced differently by concurrent flows of different sizes, we study the effect of both UDP and TCP cross-traffic on the WiFi. Note that in actual networks, downstream capacity may exceed the values used in our experiments. However, in such cases, the network which provides high downstream capacity is usually the best choice for all transfers, resulting in trivial IANS policy decisions. As examining such cases does not promise any interesting insights, we do not include such cases in our experiments.

To evaluate the achievable Web performance speedups "in the wild" the client can reach the Internet via the traffic shaper. Our vantage point is located in a well-connected university network. Thus, when using the above shaping parameters the access networks should remain the bottleneck. To confirm this we repeatedly (5 times) downloaded our chosen Web pages without any traffic shaping and confirm that the median/75th quantile latencies are lower at 17/26 ms than most shaper latencies and the receive times for large resources ($>$ 320 KB, the 99th quantile of the resource sizes) are small with a median of 41 ms. Thus, the network conditions imposed by the traffic shaper have a significant impact on the performance of "in the wild" Web page downloads. To minimize side-effects due to Domain Name System (DNS), we run a resolver on the traffic shaper. For each domain name, this resolver returns the same set of IP addresses in the same order. It caches the results for the maximum experiment duration of 20 hours. Thus, within each experiment, subsequent page loads are likely to be directed to the same Content Delivery Network (CDN) node. To further limit the impact of Internet performance variations on any particular access network selection policy, we randomize the order of page loads with different access network selection policies.

For HAS, we study both scenarios with variable capacity and scenarios with cross-traffic, see Table 8.2b. Here, we focus on varying the downstream capacity because video segment load times are typically dominated by capacity. Our two scenarios emulate different types of downstream capacity limitations: While a variable downstream capacity realized by the traffic shaper emulates limitations due to, e.g., low Received Signal Strength (RSS) in a wireless network, cross-traffic limiting the available downstream capacity emulates the presence of other flows on the bottleneck link of the network. For the variable capacity scenarios, we set the median downstream capacity based on factors of the lowest representation of one of our video workloads, 218 kBit/s, for both networks. Since a downstream capacity of 218 kBit/s is insufficient for loading the lowest representation due to overhead, we scale up the downstream capacity by factors of 1.5, 2, 2.5, 4, 5, and 10 to allow loading different representations. While on network 1 we set the downstream capacity to be constant, we let the downstream capacity on network 2 vary around the median using different variation patterns with a different coefficient of variation (c_v). In particular, we vary the downstream capacity according to six different variation patterns seen by HAS sessions using 3G networks in mobile scenarios [76]. Note, our capacities are scaled

Table 8.1: Emulated network scenarios.

Property	Levels
Downstream capacity:	2, 5, 10, or 20 Mbit/s.
Additional latency:	10, 50, or 100 ms.
WiFi crosstraffic:	None, constant UDP, variable TCP.

(a) Web browsing.

Property	Levels (variable capacity scenarios)	Levels (cross-traffic scenarios)
Median downstream capacity:	218, 327, 436, 545, 872, 1090, or 2180 kBit/s.	2 or 5 MBit/s.
c_v for downstream capacities:	0, 0.3, 0.34, 0.45, 0.49, 0.6, or 0.7.	0.
Additional latency:	80 ms.	10 or 100 ms.
WiFi crosstraffic:	None.	1, 2, 3, 4, or 8 TCP sessions.

(b) HAS.

up from the original data set[2] according to the median downstream capacity. As latency we use 80 ms, as this is the latency seen in the mobile scenarios in which the downstream capacity traces were taken [76]. For the cross-traffic scenarios, we keep the downstream capacity on network 1 constant at 2 Mbit/s, which is sufficient for loading a high representation of the video, e.g., with a sufficient screen resolution. Network 2 provides an even higher downstream capacity of 5 Mbit/s, which enables to load an even higher quality representation of the video, but we also introduce TCP cross-traffic to network 2. In particular, we study the impact of 1, 2, 3, 4, or 8 concurrent TCP sessions, each of which load files of varying sizes using Harpoon. To see if latency influences our results, here, we study both scenarios with 10 ms and 100 ms of additional latency on both networks.

8.2 Workload

We evaluate Web and video performance using different workloads hosted on the Web server in our testbed, and, for Web, using remote Web servers.

8.2.1 Web

Our goal is to evaluate IANS vs. traditional access network selection policies across a diverse set of Web pages popular among actual users while not falling into typical pitfalls, e.g., initial redirects [60]. While we base our workload on the Alexa top list, we limit bias [59] by selecting 102 highly ranked Web sites from nine categories[3]. This resulted in a list of domain names which typically redirect to some landing

[2] In the original data set, median capacities are between 88 and 219 kBit/s, depending on the scenario. As such, the median original capacities are often below the lowest representation bitrate of our video and insufficient for loading our video workload. Thus, we scale the capacities in the data set up to a factor of the lowest representation of our video workload.

[3] Our categories are: Search engines, News pages, shopping, social media, sports, arts and entertainment, business, games, and science.

Table 8.2: Representation bitrates and resolutions.

Avg. bitrate (kbps)			Resolution		
RB	**BBB**	**V**	**RB**	**BBB**	**V**
201	218	210	480x360	480x360	480x360
395	378	433	480x360	480x360	854x480
500	509	574	854x480	854x480	854x480
892	783	811	854x480	1280x720	1280x720
1498	1474	1422	1280x720	1280x720	1280x720
1992	2087	1861	1280x720	1920x1080	1440x1080
2996	3936	3523	1920x1080	1920x1080	1440x1080

page. Since such initial redirects inflate load times [60] we eliminated redirects by manually loading each page once and then adding the resulting Uniform Resource Locator (URL) to our hit list. For some sites, mainly social media sites, we find that the resulting URL does not reflect actual user experience as the landing page we saw consisted of only a few objects. For those, we picked an alternative publicly accessible subpage of the same site, e.g., a publicly accessible social media profile.

Another difficulty we encountered is that the content of Web pages changes frequently, even across consecutive page loads. Thus, we mirror all 102 Web pages in our workload[4] to the Web server in our testbed[5], see Section 8.1. This setup allows us to fetch the same version for all these pages across experiments. When accessing Web pages "in the wild", we cannot enforce that the same version of a page is retrieved. Thus, to ensure comparability of the results we only include those Web page loads in our evaluation where the resulting resource count and page size differ by less than 1% within the same run.

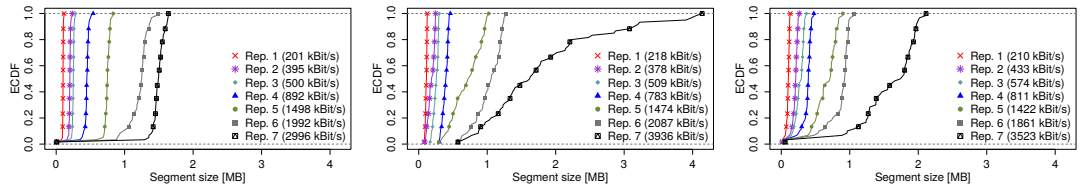
We opt for hosting Web pages on a dedicated server instead of using an emulator, such as Mahimahi [77], which replays Web page loads on different emulated network conditions on a single host. Our reasons are twofold: First, our testbed enables a wider range of scenarios, e.g., realistic access network conditions using an actual WiFi Access Point and cross-traffic. Second, we can compare loads of the mirrored version to Web page loads "in the wild" given similar network conditions.

8.2.2 HAS

As different workload properties, such as different file size distributions, may influence the effect of IANS on HAS performance, our study includes different workloads. For comparability to prior work, we utilize a well-known HAS data set [78], from which we select three videos of different genres: "Red Bull Playstreets" (RB) as a sports video, "Big Buck Bunny" (BBB) as an animation movie, and "Valkaama" (V) as

[4] We mirrored pages on January 21, 2019. We load each page once, store all resources locally, and copy them to our Web server. For each origin, i.e., each host that contributes content to the original page, we create a virtual host on our server.

[5] Our Web server is equipped with 8 cores and 16 GB of RAM, similar to a 2xlarge Amazon EC2 instance and runs an Apache Web server.



(a) "Red Bull Playstreets" (RB). (b) "Big Buck Bunny" (BBB). (c) "Valkaama" (V).

Figure 8.2: ECDFs of video segment sizes for different videos in our workload.

a live action movie. We choose videos of different genres as, even while using the same video encoding algorithm, the amount of data to be transmitted varies due to differences in content.

While these videos have overall durations of around 10 or around 90 minutes, we limit our experiment workload to a fixed subsequence within each video, which simplifies comparing results for different videos. In particular, we choose the first four minutes of each video, as this duration lies within the application range of the P.1203 model [66] (between 1 and 5 minutes), which we use to estimate the Quality of Experience (QoE). Each video is split into video segments of a fixed length. While the data set includes segment lengths of 1, 2, 4, 6, 10, and 15 seconds, we choose a segment length of 4 seconds, as it presents a good tradeoff between long and short segment lengths [78]. While shorter segment lengths introduce a higher overhead, longer segment lengths lead to fewer opportunities to adapt to changing network conditions.

Each video is available in different representations, i.e., different screen resolutions and target bitrates. From the 20 representations provided in the dataset for each video, we select a subset, as too many available representations may lead to frequent representation switches, thus, to QoE degradations.

We use seven representations for each workload, see Table 8.2, which correspond to typical resolutions and bitrates used by commercial HAS providers [79]. Our lowest chosen representation has a target bitrate of around 200 kBit/s with a screen resolution of 480x360 pixels. Lower representations correspond to a very small screen resolution, i.e., 320x240 pixels, compared to the display size of our client, which is 1920x1080. Loading a representation with such a small resolution leads to poor QoE even if no stalling events occur. The highest representation we choose corresponds to a high representation in the original workload which provides the same resolution as the screen resolution of our client, 1920x1080. Loading this representation is likely to provide a high QoE [65] and is still possible for network scenarios with high downstream capacity, see Section 8.1.

We depict the video segment size distributions for the different videos and chosen quality representations in Figure 8.2. For RB, the segment size distribution for each representation remains rather constant with only minor deviations, see Figure 8.2a. In contrast, segment sizes vary significantly for the other video workloads, particularly for high quality representations, see Figure 8.2b and Figure 8.2c. Here, within the same representation, some segments are much larger than others. As it is common to

estimate the segment size based on the average encoding bitrate, such a high spread in segment sizes may lead to inaccurate estimates.

8.3 Performance Metrics

Web: To evaluate Web performance, we capture Page Load Time (PLT), which is the time between the navigation start event and the onLoad event, as made available via the Navigation Timings Application Programming Interface (API). Furthermore, we capture Above-The-Fold time (ATF), the time taken to load all user-visible content, using the plugin introduced by da Hora et al. [57]. Finally we compute the Mean Opinion Score (MOS) using Byte Index until ATF based on the WQL model [80]: $MOS = -0.4731 * ln(ByteIndex_{ATF}) + 7.0813$. The Byte Index is computed based on the resources contained in the HTTP Archive (HAR) file, which is exported after each page load. For each resource loaded until ATF, we compute the integral of resource size over resource load time, see Bocchi et al. [58]. To determine the resource size we use the Content-Length header field, or, if not present, the body size as reported in the HAR file.

HAS: To evaluate HAS performance, during playout of the video, we log initial playout delay, start and end timestamps for all segment loads, the representation level at which each segment is played out, the buffer status and download rate based on which the Adaptive Bit-Rate algorithm (ABR) has chosen to load this representation, and timestamps at which all frames were rendered. We compute the frequency and duration of stalling events of the video playout both based on the download timestamps of the segments and based on the render times of the frames. As the render times indicate both stalling events due to long segment load times and stalling events unrelated to network conditions, e.g., due to decoder delays in the player, in our evaluation we use the stalling events based on download timestamps, which only include stalling events due to long segment load times.

From the collected application layer metrics, we compute QoE estimates. To limit potential biases of the used QoE model, we use two different models: ITU-T P.1203[6] [66, 67] and the Cumulative QoE Model (CQM)[7] [81]. We compare the MOS values computed using P.1203 to the MOS values computed using CQM at the end of each video load and find we can draw the same conclusions based on both models. In particular, the relative differences of the median MOS values for two different IANS policies or scenarios are similar when using either P.1203 or CQM. This confirms that our results are robust to the used QoE model. However, we note that the absolute MOS values vary for the two models. Although we use the same audiovisual quality scores as input to both P.1203 and CQM, we observe that the MOS scores computed using P.1203 are generally higher than using CQM by between 0.3 and 0.5. For example, while in theory the MOS can range between 1 and 5, the highest MOS value we observe is 4.3

[6] We use the code provided at `https://github.com/itu-p1203/itu-p1203` in mode 0.

[7] We use the code provided at `https://github.com/TranHuyen1191/CQM` with Tran's Window quality model.

for P.1203 and 3.9 for CQM. In this case, the highest available representation with an audiovisual score of about 4.1 is played out continuously and no stalling occurs. Here, P.1203 slightly increases the final MOS score because there are no stalling events. In contrast, CQM considers recent minimum, maximum, and average quality scores and produces a slightly lower MOS.

8.4 Course of Experiments

Web: For our evaluation, we repeatedly load Web pages using different access network selection policies. We instrument Firefox 63.0.4 via the Marionette browser automation interface. To prevent skewed results due to browser caching, we set up a fresh browser profile for each page load. To allow the browser to leverage IANS, it uses a local Web proxy[8] as discussed in Section 7.4.1. We compare the following cases: Loading a page using only a single network, using both networks with MPTCP with the primary subflow on the shorter latency network[9], and using Socket Intents with its Threshold Policy, see Section 6.3. We repeat each page load 5 times to eliminate measurement artifacts.

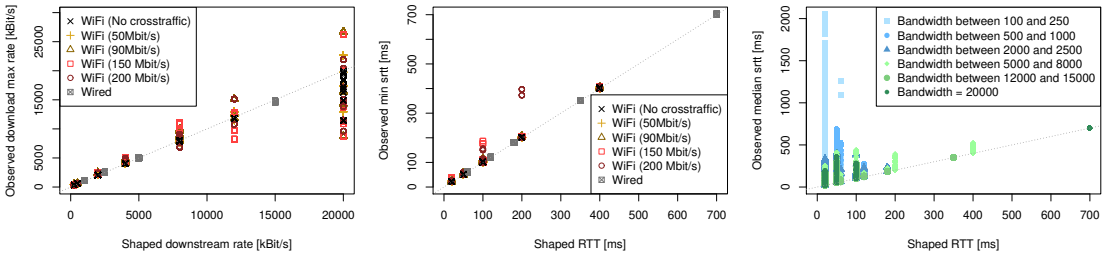
HAS: We load the video in our workload using the IANS-enabled video player, see Section 7.4.2.

As ABRs, we select BBA-0 [63] and BOLA [64]. We run experiments with different ABRs to make our results less dependent on any particular ABR. While our workload data set [78] is widely used, it does not include any audio. Thus, we emulate audio by periodically loading a file of 100 KB in parallel to the video segments, which corresponds to audio at a bitrate of 192 kBit/s with a duration of 4 seconds. We compare the following access network selection policies to load the video segments: Loading the video using only a single network, using MPTCP for all transfers, and using three IANS policies: The Optimist Policy, see Section 6.4.2, the Pessimist Policy, see Section 6.4.3, and the Selective MPTCP Policy, see Section 6.5. For the audio segments, our IANS policies choose the network which is not currently used for the video segments. This has the side effect of getting performance estimates for these networks.

We run each experiment for a fixed duration because this allows us to directly compare different experiment runs with each other: For the variable capacity scenarios, downstream capacity varies over time according to the same pattern during each video load, so each video load experiences the same changes in network conditions at the same point in time during the experiment. Note that a fixed experiment duration

[8] We evaluate the overhead of the proxy by comparing PLTs with and without the proxy. We load Web pages on network 1 without traffic shaping. PLTs with proxy increases by a median of 6.3%. There are two main reasons for this increase. First, the browser supports Transport Layer Security (TLS) False Start, but OpenSSL, which the proxy uses, does not. Thus, the browser has one fewer round trip for each TLS handshake than the proxy. Second, the Socket Intents-enabled proxy uses a two-step download for querying the Size to be Received.

[9] We use the minRTT scheduler for MPTCP, which is the default on Linux.



(a) Downstream capacity. (b) Minimum SRTT. (c) Median SRTT.

Figure 8.3: Estimated network performance characteristics against shaping

implies that we may load less content for experiments in which stalling occurs. We fix our experiment duration at 240 seconds as this time is sufficient to load enough video segments for our IANS policies to show effects, while the length of the played out video content is within the application range of the P.1203 model [66]. To limit possible biases due to comparing MOS values computed based on video content of different durations, we only include video loads with between 120 and 240 seconds of content in our results. We choose this duration as, when looking at the MOS values over time computed using CQM, the relative differences between two video loads after the first 120 seconds of content are usually similar to the differences after loading the full 240 seconds. Furthermore, we look at MOS values computed using P.1203 based on shorter content durations, i.e., for the first 120, 150, 180, and 210 seconds of content for each video load. For the cross-traffic scenarios, the results look identical to MOS values based on 240 seconds of content. For the variable capacity scenarios, we can draw the same conclusions based on MOS values for shorter content durations, except for the capacity decrease scenario, in which shorter content durations do not capture the decrease in capacity, and except for outliers due to artifacts in the P.1203 model. We repeat our experiment 5 times for each combination of scenario, policy, and ABR. In our evaluation, we compute the median MOS with confidence intervals of the median for each combination of scenario and policy, i.e., for up to 15 MOS values.

8.5 Network Characteristics Feasibility Study

Before we evaluate the performance benefits of IANS for Web and HAS, we conduct a feasibility study to check the accuracy of the network performance characteristics that are available to our IANS policies. Hereby, we set up a wide range of different shaping levels within our testbed and check the network performance estimates available to IANS against them.

To demonstrate the feasibility of estimating network performance characteristics within our prototype, we compare our measured network performance characteristics with the actual shaped values in our testbed, see Section 8.1. We use a wide range of typical network performance characteristics of both cellular and WiFi net-

Table 8.3: Shaping levels for feasibility study

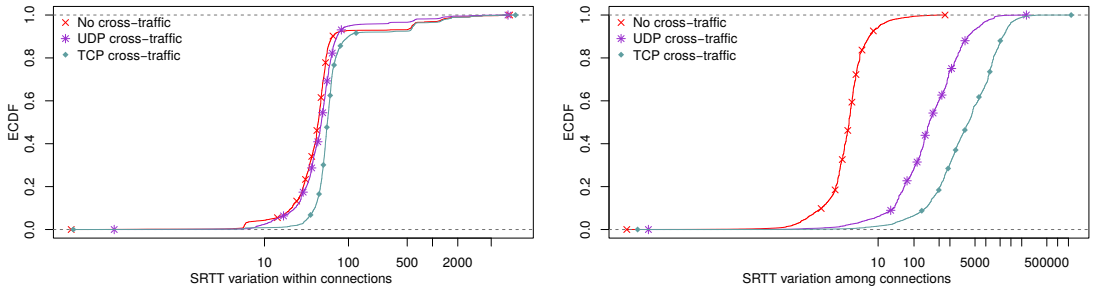
Property	Levels
Network 1 RTT:	60, 120, 180, 350, or 700 ms.
Network 1 upstream capacity:	0.05, 0.3, 0.7, 1.8, 3, or 5 Mbit/s.
Network 1 downstream capacity:	0.1, 0.5, 1, 2.5, 5, 15, or 20 Mbit/s.
Network 2 RTT:	20, 50, 100, 200, or 400 ms.
Network 2 upstream capacity:	0.1, 0.25, 0.5, 1, 2, 4, or 6 Mbit/s.
Network 2 downstream capacity:	0.25, 0.5, 2, 4, 8, 12 or 20 Mbit/s.
Network 2 packet loss:	none, 0.1, 0.25, or 0.5%.
Network 2 cross-traffic:	none, constant UDP flow of 50, 90, 150, or 200 Mbit/s.

works [1], see Table 8.3. As the observed maximum downstream capacity depends on the amount of traffic seen, our workload needs to saturate the downstream capacity of the bottleneck link to get accurate results. To explore what traffic is needed to fill the link, our client fetches static Web pages of increasing sizes: 32 objects of 1 KB, 32 objects of 10 KB, 8 objects of 100 KB, 2 objects of 1 MB, and 32 objects of 100 KB.

Figure 8.3 shows our three main network performance characteristics, maximum downstream capacity, minimum Smoothed Round Trip Time (SRTT), and median SRTT, compared to the shaper settings against which they were measured. Figure 8.3a shows the estimated maximum downstream capacity that the Multi Access Manager (MAM) observes on the WiFi and wired links during the last page load in a run, i.e., once the bottleneck link has been filled. Over the WiFi with no crosstraffic and over the wired access network, the estimated maximum downstream capacity corresponds to the shaped downstream capacity in all cases except for 20 MBit/s. In the latter case, the bottleneck link only gets saturated during the last page load, which is why we see the estimated maximum downstream capacity increases during page load as the bottleneck link gets saturated. At the end of the page load, the estimated maximum downstream capacity has reached the full 20 MBit/s. With cross-traffic on the WiFi, the estimated maximum downstream capacity varies around the shaped rate due to random contention. When the page load collides with the crosstraffic, the WiFi driver at the client will discard any affected frame, thus, the observed downstream capacity drops, as it only includes correctly received packets. Afterwards, the WiFi Access Point will retransmit any lost frames on Layer 2, resulting in a temporarily higher downstream capacity than what is shaped on the bottleneck link upstream of the AP. To smooth out these irregularities, we introduce a moving average across the last 10 samples and use the maximum of the moving averages as our maximum downstream capacity estimate.

Figure 8.3b shows the minimum SRTTs seen by MAM for different shaped Round Trip Times (RTTs)[10] for all page loads. In most cases the minimum SRTT corresponds to the shaped RTT. In the presence of high crosstraffic with either 150 Mbit/s or 200 Mbit/s constant UDP traffic on the wireless channel, the MAM sees a few outliers in cases where all TCP connections are affected by the crosstraffic. However, we note

[10]We shape every RTT as symmetrical latencies on the up- and downlink.



(a) Variation within connections

(b) Variation among connections

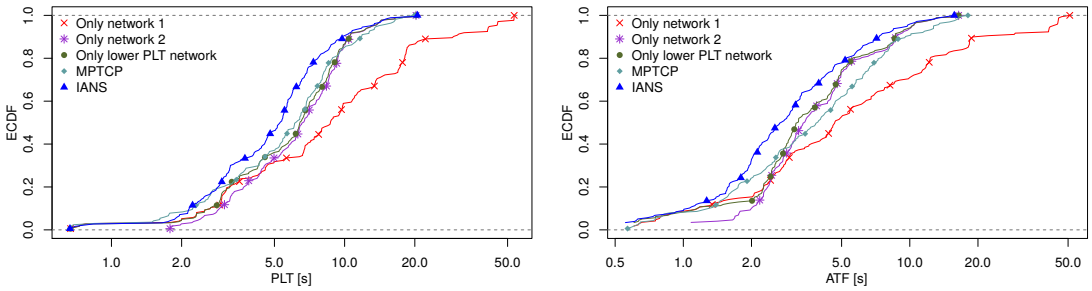
Figure 8.4: SRTT variations with cross-traffic.

that in most cases, the minimum SRTT observed by MAM is a good estimate for the minimum SRTT that can be achieved on the path. Figure 8.3c shows the median of the SRTTs that MAM observes for the current TCP connections for different shaped downstream capacity limitations. As congestion on the path has a high impact on the SRTTs of the connections, the observed values are considerably higher when a large workload is fetched on a network with low downstream capacity. However, for 2 Mbit/s and higher, the observed median SRTTs are close to the shaped RTTs, even with crosstraffic on the WiFi. We conclude that we can use the median SRTT observed over a network as an estimate of the RTT that a download will experience if it is scheduled on this network.

Figure 8.4 shows the SRTT variations for scenarios without cross-traffic, for constant UDP cross-traffic on the wireless link of network 2, and for variable TCP cross-traffic on the bottleneck of network 2. Figure 8.4a shows SRTT variations within single connections for a network with a shaped RTT of 10 ms and a downstream capacity of 2 Mbit/s during all Web page loads in our Web evaluation. While the median of SRTT variations is 43 ms without any cross-traffic, this median increases only sligthly with UDP cross-traffic on the WiFi, i.e., to 47 ms. With variable TCP cross-traffic, the median increases to 56 ms. Figure 8.4b shows SRTT variations among all concurrent connections for a network with a shaped RTT of 100 ms and a downstream capacity of 20 Mbit/s. While this variation is low without any cross-traffic, i.e., with a median of 1.5 ms, with constant UDP cross-traffic on WiFi it increases to 267 ms and with variable TCP cross-traffic on the bottleneck it increases to 3327 ms.

As SRTT variations within connections increase only slightly more than the shaped RTT even for low capacities, it is too weak a signal for us to detect cross-traffic. However, SRTT variation among connection varies by a multiple of the shaped RTT. This signal is better suited to detect cross-traffic.

The observed channel utilization is in line with our expectations, i.e., it increases to 60-70% with cross-traffic on WiFi (plot not shown). As the prototype cannot estimate downstream packet loss, but only upstream packet loss, which is not relevant to our performance evaluation, we exclude loss from our feasibility study.



(a) Empirical Cumulative Distribution Function (ECDF) of Page Load Times (PLT)

(b) ECDF of Above-The-Fold Times (ATF)

Figure 8.5: Asymmetric scenario (network 1: 10 ms, 2 Mbit/s, network 2: 100ms, 20 Mbit/s)

8.6 IANS Benefits For Web Browsing

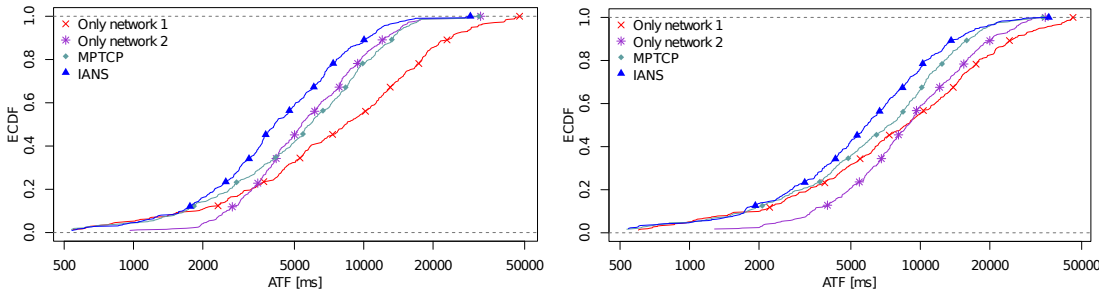
Next, we report on the benefits of IANS in a single scenario and in our systematic study, in which we highlight under which network conditions it yields which benefits.

8.6.1 Asymmetric Network Scenario: In-depth Discussion

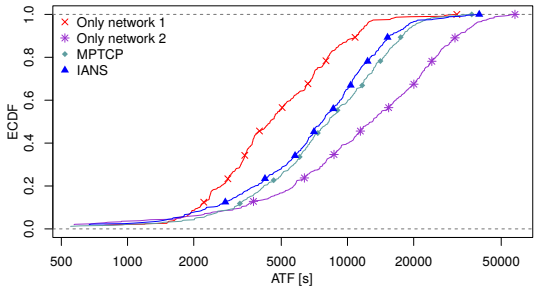
We start our exploration by focusing on the asymmetric network scenario. Our motivation for starting with this case is that here IANS should yield the biggest benefits for Web performance. In particular, network 1 has a short latency of only 10 ms but a limited downstream capacity of 2 Mbit/s. Network 2 has a downstream capacity of 20 Mbit/s but a latency of 100 ms.

Figure 8.5 shows the Web performance of all access network selection policies according to two metrics, PLTs and ATFs, as ECDF across all 102 Web pages. IANS outperforms all other access network selection policies for ATF and in almost all cases for PLT. Only MPTCP can lead to shorter PLTs, see Figure 8.5a. When investigating the cause, we find that IANS, as intended, benefits from the asymmetric network performance characteristics: Small resources benefit from the short latencies of network 1, while large resources benefit from the high downstream capacity of network 2. Since MPTCP distributes all resource loads across both networks, it saturates the short latency network first. This leads to congestion on this network which, in turn, inflates the load times of smaller resources. Thus, ATFs and PLTs increase for MPTCP. IANS avoids unnecessary congestion on the short latency network since it mainly uses the high downstream capacity network for retrieving large resources. Furthermore, IANS yields a statistically significant improvement in MOS (plot not shown) with a mean MOS difference of 0.11 compared to using the “better” (lower PLT) of the two single networks, and of 0.07 compared to using MPTCP.

Using a single network (either network 1 or network 2) leads to the worst PLTs and ATFs with a huge spread ranging from 0.7 seconds to more than 53 seconds. Using



(a) With constant UDP cross-traffic on network 2. (b) With variable TCP cross-traffic on network 2 and its bottleneck.

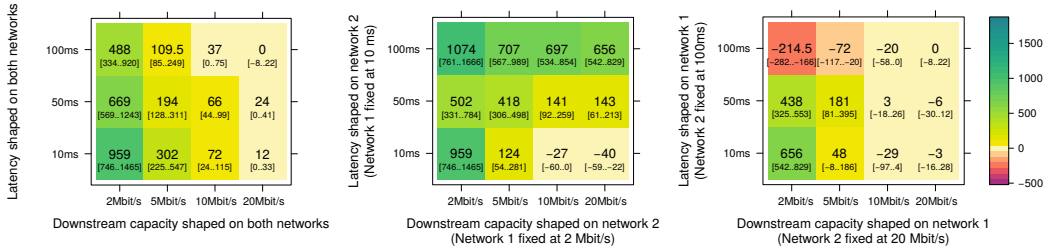


(c) With variable TCP cross-traffic on network 2 and its bottleneck, if network 2 is shaped at 10 ms and 2 Mbit/s.

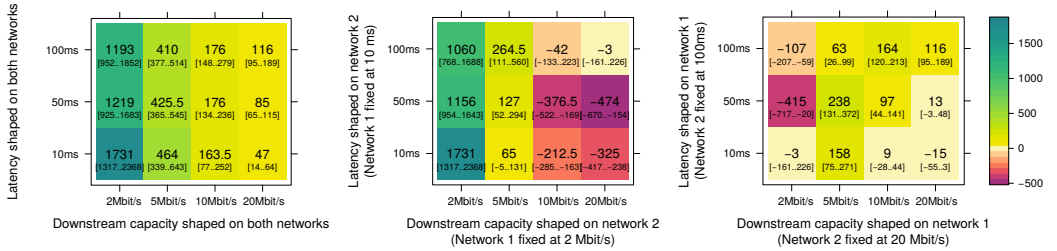
Figure 8.6: ECDF of Above-The-Fold Times for asymmetric scenario (network 1: 10 ms, 2 Mbit/s, network 2: 100ms, 20 Mbit/s)

the better of the two networks avoids some of the outliers; none of the page loads takes longer than 20.4 seconds. Thus, in this scenario, there is no single "best" network to use, as intended in our setup, see Section 8.1.

Adding cross-traffic results in slightly longer load times but does not change the general observations, see Figure 8.6. IANS still outperforms using either single network or using MPTCP and speedups vs. using a single access network are comparable to the scenario without cross-traffic. Both for constant UDP cross-traffic, see Figure 8.6a, and for variable TCP cross-traffic, see Figure 8.6b, ATFs when using only network 2 become slightly longer because the cross-traffic on network 2 results in longer load times for some resources. Hereby, we see a higher impact for variable TCP cross-traffic, because we place this traffic on both the WiFi network and the bottleneck link. If we instead add variable cross-traffic to a network with short latency and low downstream capacity, IANS still outperforms MPTCP, see Figure 8.6c. However, as both IANS and MPTCP are sensitive to congestion on the lower latency network, using only the network which does not have cross-traffic, becomes the better option. Therefore, a future version of IANS should detect high congestion on a network and then avoid loading resources over this network.

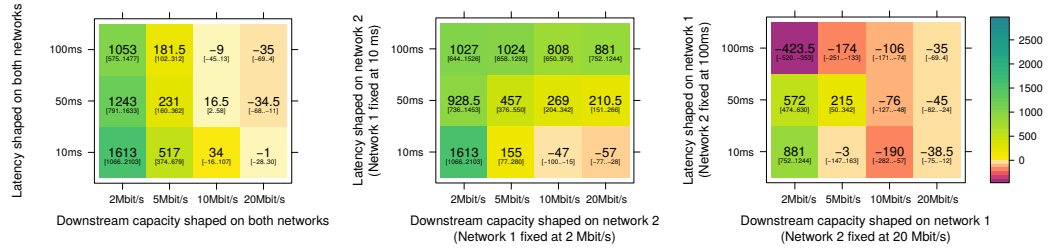


(a) IANS: Symmetric (b) IANS: Asymmetric, network 1 fixed (c) IANS: Asymmetric, network 2 fixed

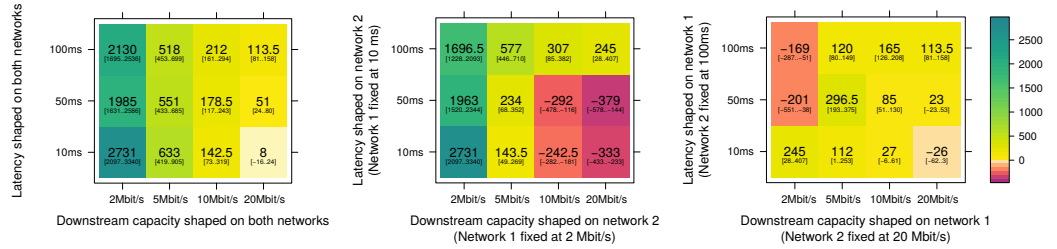


(d) MPTCP: Symmetric (e) MPTCP: Asymmetric, network 1 fixed (f) MPTCP: Asymmetric, network 2 fixed

Figure 8.7: ATF improvements vs. using the single "better" network (median [ms] plus confidence intervals).

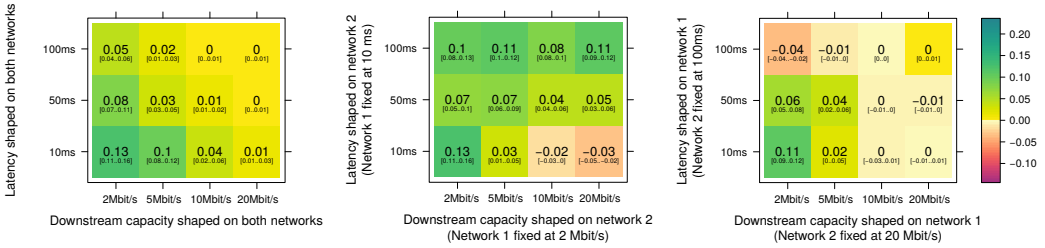


(a) IANS: Symmetric (b) IANS: Asymmetric, network 1 fixed (c) IANS: Asymmetric, network 2 fixed

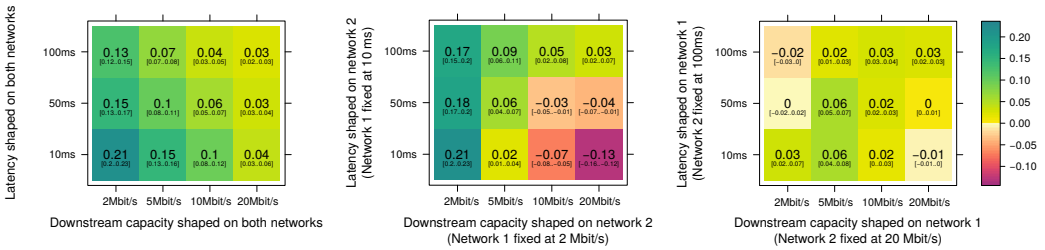


(d) MPTCP: Symmetric (e) MPTCP: Asymmetric, network 1 fixed (f) MPTCP: Asymmetric, network 2 fixed

Figure 8.8: PLT improvements vs. using the single "better" network (median [ms] plus confidence intervals).



(a) IANS: Symmetric (b) IANS: Asymmetric, network 1 fixed (c) IANS: Asymmetric, network 2 fixed



(d) MPTCP: Symmetric (e) MPTCP: Asymmetric, network 1 fixed (f) MPTCP: Asymmetric, network 2 fixed

Figure 8.9: MOS improvements vs. using the single "better" network (median [ms] plus confidence intervals).

8.6.2 Systematic Study of Scenarios

Next, we look at the results of the systematic study of 33 network scenarios[11] with latencies of 10, 50, or 100 ms and downstream capacities of 2, 5, 10, or 20 Mbit/s for both access networks. Rather than using 33 ECDFs to visualize the results, we show, using heatmaps, the median ATF improvement of IANS and MPTCP vs. the "best" single network (according to PLT), see Figure 8.7. Note, the results for PLT and MOS show similar results, see Figure 8.8 and 8.9. Each subplot focuses on a different scenario, namely, Figure 8.7a and 8.7d on access networks with symmetric network conditions, Figure 8.7b and 8.7e on access networks with asymmetric conditions where network 1's characteristics do not change and Figure 8.7c and 8.7f on access networks with asymmetric conditions where network 2's characteristics do not change. The color schema (same for all plots) goes from green for large performance improvements over light yellow for no significant differences to violet for large performance penalties. Each heatmap entry contains the median ATF improvement in ms and the corresponding confidence intervals.

Overall, green which corresponds to significant improvements dominates the results for IANS and MPTCP. However, red and violet are more common for MPTCP. Yellow dominates if the downstream capacity is large for both networks. Note, that the size of the confidence intervals can be large, i.e., hundreds of milliseconds, as load times fluctuate between page loads. Some of these effects are due to influences unrelated to networking, such as in-browser processing, and different orders in which the browser

[11] Each scenario corresponds to a combination of latencies and downstream capacities on both networks in our Web browsing experiments, see Table 8.2a.

may load the resources. Other effects are related to the multitude of different Web pages and that the Web page load times range from less than a second to more than 20 seconds.

IANS shows the largest speedups in asymmetric scenarios, see Figure 8.7b and 8.7c, similar to the one discussed Section 8.6.1. For MPTCP, this is not always the case, see Figure 8.7e and 8.7f. The reason is that IANS (for the 10/100 ms latency case) in contrast to MPTCP can indeed load small/large resource over the short latency/high downstream capacity network while MPTCP tends to use both and, thus, may suffer from head-of-line blocking and congestion on the low downstream capacity network.

Even for less asymmetric scenarios (10/50 ms, 50/100 ms latency) IANS still improves ATFs over MPTCP, see Figure 8.7b and 8.7e. MPTCP saturates network 1 (the lower latency one) first, which, again, leads to congestion on network 1 and inefficient use of the high downstream capacity of network 2. MPTCP still degrades ATF if both networks have the same short latency (10ms), but asymmetric downstream capacities (2/10 Mbit/s or 2/20 Mbit/s), see Figure 8.7e. Since network 2 is the WiFi network, it adds delay and, thus, MPTCP will again saturate the lower latency one rather than taking advantage of the higher downstream capacity one.

For symmetric scenarios with low downstream capacities IANS again shows significant ATF improvements, see Figure 8.7a. Here, both networks should be used, so IANS's capability of distributing resource loads across the networks improves resource load times and, thus, PLT and ATF. Speedups increase as latencies decrease due to less per-connection overhead. For purely symmetric scenarios, MPTCP outperforms IANS, see Figure 8.7d, as MPTCP is able to use the bundled network downstream capacity at a finer granularity by using subflows. Thus, if a page's resources are skewed, i.e., have one large resource, MPTCP can efficiently use both networks while IANS cannot.

In scenarios with high downstream capacity, e.g., Figure 8.7a and 8.7d neither IANS nor MPTCP yields major benefits as a single access network already provides sufficient network resources. Still, MPTCP is slightly better than IANS since it is able to reuse all TCP connections. However, future versions of IANS could detect such network conditions and then default to MPTCP. With cross-traffic on network 2, using MPTCP remains the best option in most cases. However, using only network 1, which is not impacted by cross-traffic, is even better in case network 1 has sufficient downstream capacity. Therefore, enabling MPTCP only selectively using IANS may have an advantage over using MPTCP in all scenarios.

One scenario, see Figure 8.7c and Figure 8.7f upper left corner, stands out, since both IANS and MPTCP show rather large performance penalties. Here, both access networks have a long latency (100 ms) and the downstream capacities are asymmetric (2/20 Mbit/s). Per se, using network 2 only is the best option. However, both IANS and MPTCP use network 1 as well, as its latency is lower than network 2's due to the WiFi overhead. By using network 1 IANS incurs extra cost due to TCP connection overhead and both IANS and MPTCP degrade performance due to the imposed congestion on network 1. Moreover, IANS cannot reuse existing TCP connections

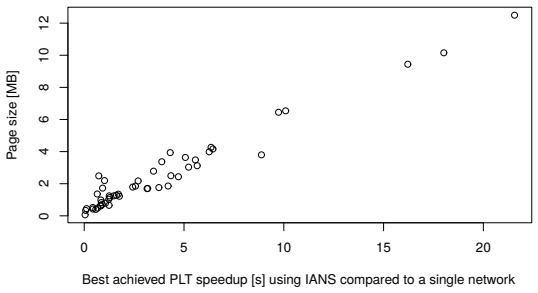


Figure 8.10: Symmetric scenario with 10ms, 2Mbit/s.

on a different network. Thus, in the future, IANS should account for connection establishment overhead and congestion even more.

Summary of systematic study: Our most important findings are: 1.) IANS improves Web performance the most for asymmetric scenarios and for symmetric scenarios with low downstream capacity. 2.) For asymmetric scenarios IANS often outperforms MPTCP. Indeed, MPTCP may introduce a performance penalty even compared to using a single network only. 3.) In symmetric scenarios, using MPTCP helps the most. Thus, IANS should take advantage of MPTCP for such cases.

8.6.3 Performance Benefits For Different Web Pages

Next, we take a closer look at Web page properties and check to which extent they correlate with the observed speedups. The most obvious property is Web page size[12]. Thus, Figure 8.10 shows a scatter plot of the Web page size vs. the best achieved PLT speedup[13] for one of the symmetric scenarios (10 ms and 2 Mbit/s). Note, there is a strong correlation. Almost all points fall on the diagonal. Thus, the Pearson correlation coefficient with 0.978 is also close to the perfect one. For MOS we see some improvements as well but not a strict correlation, which is not surprising given that MOS uses a logarithmic scale. Still, IANS, e.g., improves a "bad" MOS' (< 2.5) to an "acceptable" MOS (> 2.5) for 10% of the Web pages.

We find similar strong correlations between Web page size and PLT for almost all symmetric scenarios, see Table 8.4. Note, correlation decreases with larger downstream network capacities. This is not surprising since as we pointed out above, the best options for such scenarios is using a single network. When considering Web pages of different categories, see Section 8.2.1, we find that "Gaming" Web pages are often larger, so they show more significant speedups than, e.g., smaller Web pages from the "Search Engine" category.

[12]The Web page size is the sum of all resource sizes from one page load. In our workload, Web page sizes range from 64 KB to 12.5 MB.

[13]The plot for ATF is similar but contains one outlier due to a failure of the plugin which exports ATF.

Table 8.4: Pearson coefficient between page size and best achieved PLT speedup: Symmetric scenarios

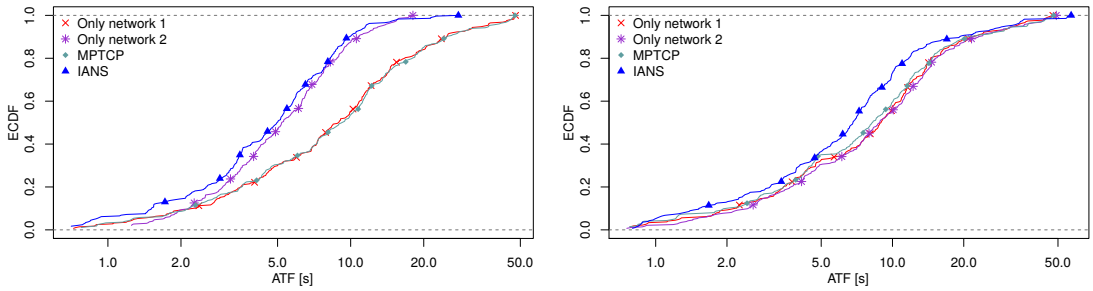
Latency \ Capacity	2 Mbit/s	5 Mbit/s	10 Mbit/s	20 Mbit/s
10ms	0.978	0.943	0.923	0.764
50 ms	0.953	0.920	0.847	0.367
100 ms	0.974	0.956	0.589	0.232

We do not find strong correlations for other Web page properties including resource count[14], and median resource size[15]. The largest Pearson correlation coefficients for resource count/median resource size are 0.517/0.541 for the symmetric scenario (10 ms and 2 Mbit/s).

For asymmetric scenarios, we see correlations between page size and PLT when the downstream capacity is low for both networks (Pearson coefficients greater than 0.8). Otherwise, there is no strong correlation with page size as the benefits of downstream capacity bundling decreases. There is also no strong correlation with resource count or median resource size.

Still, IANS shows improved performance, recall Section 8.6.1. The reason is that IANS provides multiple opportunities for speedups: On the one hand, IANS provides benefits for large Web pages and those with many large resources. On the other hand, IANS realizes speedups for small pages with a few "large" resources[16], which benefit from the high downstream capacity network, and many "small" resources[17], which benefit from the short latency network. Moreover, distributing resources avoids congestion on the lower latency network. This is where IANS shows an advantage over MPTCP.

IANS cannot speed up Web page loads where the page exclusively consists of small resources since using a single short latency access network while reusing the TCP connections is the best option. Moreover, IANS does not provide much benefit if the Web page consists of many "small" and few large resources, e.g., a single large resource. In such cases, scheduling the large resource on the large downstream capacity network may not always be beneficial due to its delay and the corresponding connection establishment overhead.



(a) Asymmetric scenario.

(b) Symmetric scenario.

Figure 8.11: "In the wild": ECDFs of Above-The-Fold Times (ATF)

8.6.4 Performance In The Wild

To confirm that the performance benefits do not just apply to controlled testbed settings with mirrored Web pages, we load the same pages from their servers "in the wild", recall Section 8.1. Figure 8.11a shows the ECDFs for the ATFs[18] for the asymmetric network scenario (network 1: 10 ms and 2 Mbit, network 2: 100 ms and 20 Mbit) first discussed in Section 8.6.1. As before IANS outperforms either of the two single networks as well as MPTCP. Indeed, MPTCP is much worse and close to using only the low downstream capacity network for most Web pages. The reason is that most Web servers "in the wild" do not support MPTCP: Only three out the of 102 Web pages in our workload partially support MPTCP—we see at least one successfully established subflow with an MPTCP option. Note, even for these not all involved Web servers did support MPTCP. Without server-sided MPTCP support, MPTCP establishes the primary subflow over the lower latency network, i.e., network 1, and suffers from its low downstream capacity. Thus, the fact that MPTCP needs server-sided support limits the performance benefits that it can achieve in the wild. Since IANS does not need any server-sided support IANS can provide similar benefits in the wild as in the testbed.

Figure 8.11b shows the ECDF for the ATF for a symmetric network scenario (10 ms and 2 Mbit/s). Again MPTCP does not provide any benefits compared to using a single network as most servers do not yet support MPTCP. Here, even though MPTCP outperforms IANS in the testbed, see Section 8.6.2, IANS in the wild currently outperforms MPTCP (due to its limited deployment) as well as only using a single network.

In summary, our experiments using Web servers "in the wild" confirm IANS's performance benefits. Furthermore, they highlight that MPTCP, due to its limited

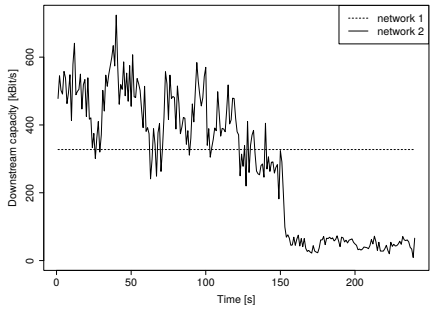
[14] Resource count is the number of successful HyperText Transfer Protocol (HTTP) requests (status code other than 4xx or 5xx) as observed from the HAR file exported during page load. In our workload, resource count ranges from 10 to 314, with a median of 73.

[15] In our workload, the median of all resource sizes loaded for a single page ranges from 300 B to 79 KB.

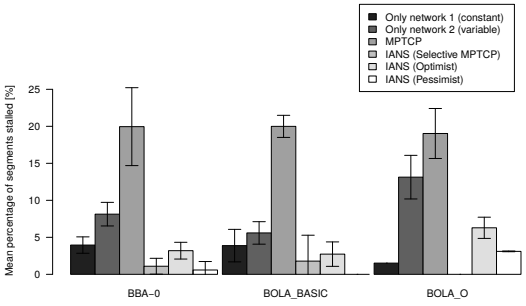
[16] We define "large" as greater than the 95th quantile of all resource sizes, which is 133 KB.

[17] We define "small" resources as those smaller than overall median resource size, which is 7.7 KB.

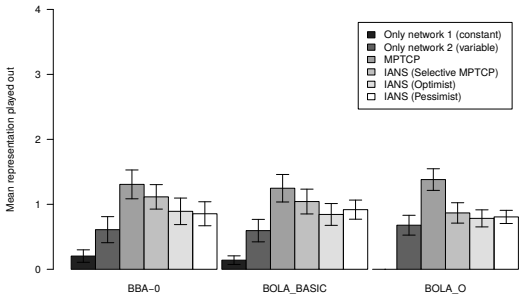
[18] PLT and MOS again show similar effects.



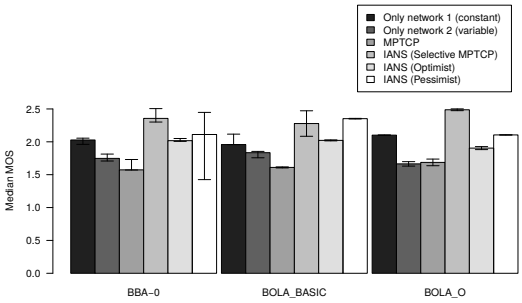
(a) Shaped downstream capacity.



(b) Percentage of video segments stalled (mean with confidence intervals).



(c) Loaded Representations (mean with confidence intervals).



(d) QoE (median MOS with confidence intervals).

Figure 8.12: Capacity decrease scenario (median downstream capacity 327 kBit/s, downstream capacity on network 2 varies with $c_v = 0.7$).

deployment, often cannot provide the expected performance benefits in practice and may even degrade performance.

8.7 IANS Benefits For Video Streaming

Next, we evaluate the benefits of IANS for video performance in a scenario with decreasing capacity and in our systematic study. First, we compare the benefits of different IANS policies, all of which use the same Socket Intents, depending on the downstream capacity and downstream capacity variation pattern. Then, we complement this study by a discussion of scenarios with constant downstream capacity, but with variable cross-traffic.

8.7.1 Capacity Decrease Scenario: In-Depth Discussion

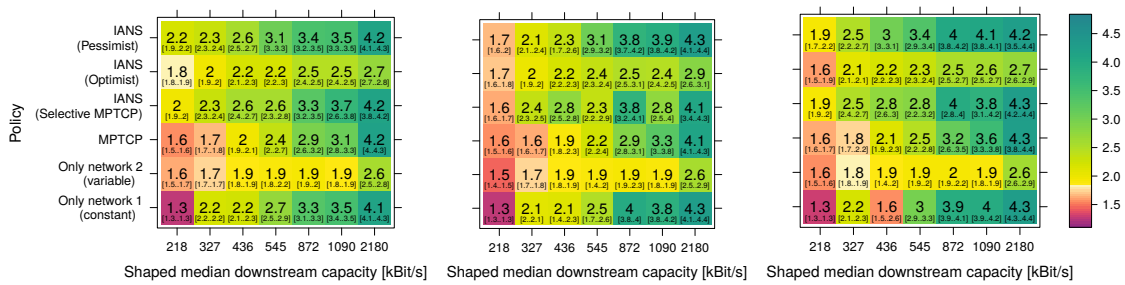
First, we focus on a scenario with decreasing downstream capacity for network 2 and constant downstream capacity for network 1, using the "Big Buck Bunny" workload. Our motivation for starting with this case is that IANS should yield high benefits as it is downstream capacity aware and should, thus, always use the higher downstream

capacity network. Here, we focus on how the Optimist Policy and Pessimist Policy adapt to the downstream capacity changes and whether the Selective MPTCP Policy can provide any benefits compared to using MPTCP for all segments. Figure 8.12 shows details of our experiment setup and our results.

In our experiment, we shape downstream capacities according to Figure 8.12a, i.e., network 1 provides a constant downstream capacity of 327 kBit/s throughout the experiment, while the downstream capacity on network 2 varies around the same median capacity according to the "metro" trace [76]. As 327 kBit/s is 1.5 times the bitrate of the lowest representation of our animation video, network 1 provides enough downstream capacity to load the lowest representation at all times. Network 2 initially provides more downstream capacity, i.e., between 350 and 600 kBit/s, which allows loading video segments using a higher representation. However, after around 150 seconds, the downstream capacity of network 2 decreases to between 30 and 60 kBit/s, so using network 2 leads to stalling even for the lowest representation. Here, the end-user device which created the original trace entered a tunnel, leading to a drastic decrease in signal strength. Due to these drastic changes in downstream capacity, the coefficient of variation (c_v) for this scenario is high, i.e., 0.7.

Figure 8.12b shows the percentage of stalled segments, i.e., segments with long load times which resulted in stalling of the playout. Here, IANS recognizes the decreased downstream capacity on network 2 and, therefore, uses only network 1 after downstream capacity decreases. Stalling may still occur in cases where the downstream capacity decreases at the same time at which the IANS policy decides which network to use for a transfer, therefore, IANS cannot detect the downstream capacity change for this transfer yet. In contrast, MPTCP continues to use both network 1 and network 2 for all transfers, which leads to stalling for all segments loaded after the capacity decrease. This leads to a mean stalling percentage of about 20% of segments for MPTCP because it is able to load about 10 more segments after the capacity decrease, all of which stall the playout. In contrast, network 2 is only able to load 3 more segments during the remainder of the experiment due to very long load times, which results in a lower overall percentage of stalled segments around 5 to 15%. Note, there is an interaction between access network selection policy and ABR. Using different ABRs leads to different stalling percentages. However, for all ABRs used in our experiments, we observe similar overall QoE and stalling percentages per scenario and access network selection policy.

While stalling is important for QoE, which representation of the video is retrieved also plays a role. Figure 8.12c shows the mean of the loaded representations in the Capacity Decrease scenario for all ABRs and access network selection policies. Here, using only network 1 leads to loading the lowest representation, 0, for most segments, as network 1 provides an insufficient downstream capacity to load a higher representation. As MPTCP can aggregate the downstream capacity of network 1 and network 2, using MPTCP leads to loading a higher representation for all ABRs. IANS yields a higher mean representation than any single network, but a lower mean representation than MPTCP because they use a single network for many segment loads.



(a) RB. (b) BBB. (c) V.

Figure 8.13: Median MOS with confidence intervals for scaled Capacity Decrease scenarios (c_v =0.7).

Combining the stalling events and the representations, as well as other factors such as the frequency of representation switches, Figure 8.12d shows the QoE as Mean Opinion Score (MOS) computed using the ITU-T P.1203 model for all access network selection policies and ABRs. Here, the Pessimist Policy and the Selective MPTCP Policy outperform the other access network selection policies by providing an "acceptable" MOS of more than 2, as opposed to a "bad" MOS of below 2. The QoE is higher because both the Pessimist Policy and the Selective MPTCP Policy reduce stalling events by recognizing the decreased downstream capacity on network 2 and, therefore, by only using network 1 after downstream capacity decreases. In contrast, MPTCP continues to use both network 1 and network 2 for all transfers, which leads to stalling, recall Figure 8.12b and, therefore, a bad MOS. We note that, for BBA-0, the Pessimist Policy provides bad QoE in one case due to too frequent switches between representations[19]. While we see this effect here only for BBA-0, we see the same effect for other ABRs in other scenarios. The Optimist Policy achieves a QoE similar to using only network 1: While it allows higher representations of the video to be loaded, it still sees several stalling events. We find that these stalling events occur because, after, the downstream capacity on network 2 decreases, the Optimist Policy attempts to use network 2 for every fourth segment because of its "best-case" load time estimate due to the high downstream capacity the Optimist Policy has seen in the past.

Summary of single scenario: IANS can detect a persistent decrease in downstream capacity and use a network with a more stable downstream capacity, thus, reduce stalling and improve QoE. In particular, the Pessimist Policy and the Selective MPTCP Policy provide good results for the Capacity Decrease scenario.

8.7.2 Systematic Study of Variable Capacity Scenarios

Next, we present the results of our systematic study of scenarios with variable capacity, in which we keep the downstream capacity on network 1 constant and vary the downstream capacity on network 2 during each run, recall Table 8.2b. In total, our

[19]The P.1203 model penalizes the MOS if the ABR never remains with the same representation for 30 seconds or more during the duration of the experiment (240 seconds).

study consists of 42 variable capacity scenarios, whereby each scenario corresponds to a combination of median downstream capacity and capacity variation pattern. For each scenario, we show the QoE achieved by different access network selection policies using heatmaps, see Figure 8.13 for the Capacity Decrease scenario scaled to different median downstream capacities and Figure 8.14 for all other scenarios. While the Capacity Decrease scenarios and the Train scenarios have a high c_v of 0.6 and 0.7, the Ferry scenarios and the Car scenarios have a medium c_v of around 0.5, and the Bus scenarios as well as the Tram scenarios have the lowest c_v of around 0.3.

Each subplot refers to all scenarios with the same capacity variation pattern, whereby we scale the median downstream capacities. Within each subplot, each column corresponds to a single scenario and shows the achieved QoE for different access network selection policies. Hereby, IANS policies are displayed at the top and results for using only a single network or using MPTCP for all transfers are displayed below. Each heatmap entry shows the QoE as MOS values computed using the ITU-T P.1203 model. Since the different ABRs often yield a similar QoE, we show the median MOS for all ABRs for the same scenario and access network selection policy. Furthermore, each heatmap entry contains the median and the corresponding confidence interval. Note, the size of the confidence interval can be large, i.e., with MOS differences of more than 1, because for some video loads, the P.1203 model penalizes the computed MOS due to frequent representation switches[20]. The color schema (same for all plots) ranges from violet and red for MOS values below 2 over light yellow for MOS values between 2 and 2.5 to green for MOS values of 2.5 or more.

Overall, in Figure 8.13, green dominates the results for IANS and for scenarios with high downstream capacities, whereas red and violet are more common for using a single network and for low downstream capacities. In Figure 8.14, green also dominates for MPTCP.

For the RB video in the Capacity Decrease scenarios, shown in Figure 8.13a, IANS yields MOS improvements for scenarios with low downstream capacities, i.e., of 545 kBit/s or less. For the scenario with a median capacity of 218 kBit/s, both the Pessimist Policy and the Selective MPTCP Policy provide a median MOS of 2 or more for RB, while a single network and MPTCP yield a MOS of 1.6 or less. While MPTCP hurts performance due to stalling, IANS is able to improve the MOS because it is aware of downstream capacity changes, e.g., the decrease in downstream capacity on network 2. Therefore, IANS loads video segments over the higher downstream capacity network 1, which reduces stalling events or avoids stalls entirely, as seen previously in Section 8.7.1.

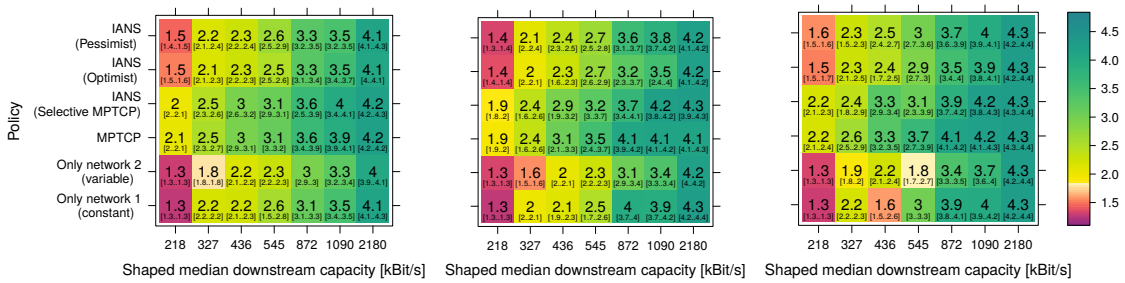
For the BBB video, see Figure 8.13b, IANS leads to worse performance than RB for the lowest downstream capacity scenario with a median downstream capacity of 218 kBit/s. This is because BBB's lowest representation has a slightly higher bitrate than RB, so there is insufficient capacity to load even the lowest representation

[20] In particular, we find that P.1203 heavily penalizes playouts in which the played out representation changes more frequently than every 30 seconds even for cases in which two playouts are otherwise identical, i.e., they include the same number of stalling events and similar played out representations.

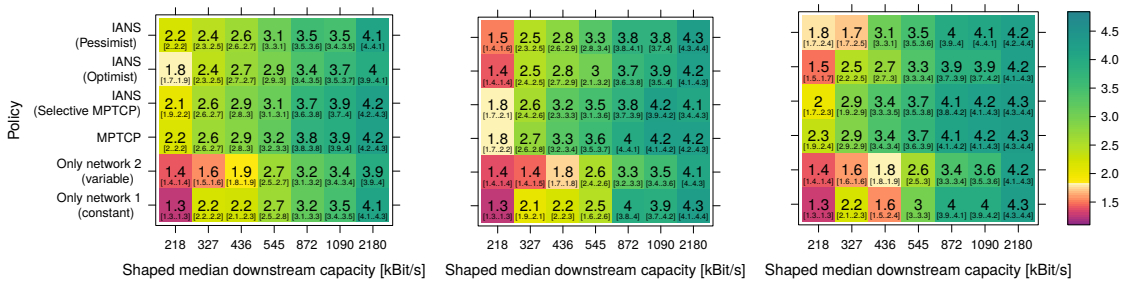
without stalling. For scenarios with downstream capacities between 327 kBit/s and 545 kBit/s, similar to RB, both the Pessimist Policy and the Selective MPTCP Policy are able to select a network with sufficient downstream capacity and, therefore, improve the MOS compared to using a single network or MPTCP. Surprisingly, for BBB, the Selective MPTCP Policy performs worse for 545 and 1090 kBit/s than for 436 and 872 kBit/s because stalling events occur for the former cases. Here, the Selective MPTCP Policy enabled MPTCP based on the assumption that sufficient capacity is available for the selected representation bitrate, however, the loaded video segment was unusually large. This occurs only for BBB, as BBB has the highest spread in video segment sizes, recall Figure 8.2b. Future work may fine-tune the Selective MPTCP Policy for such workloads. Results for V are similar to RB, see Figure 8.13c, except for a low MOS score for the 436 kBit/s scenario and network 1. Here, we see an interaction between the shaped capacity and the representations' segment sizes for this particular video, which causes the ABR to switch representations frequently, which causes P.1203 to penalize MOS.

For the "Train" scenarios, shown in Figure 8.14a, both the Selective MPTCP Policy and using MPTCP outperform using a single network, e.g., they improve a MOS of 2.2 using either single network to a MOS of 3. Here, the Optimist Policy and Pessimist Policy are able to only slightly improve QoE, as they can only use the better single network. For the RB workload, using the better single network does not result in switching to a higher quality representation, which would be required to provide a better QoE as seen using MPTCP. However, for the BBB workload, see Figure 8.14b, and for the V workload, see Figure 8.14c, we do see an increased MOS for the Optimist Policy and Pessimist Policy. For these workloads, selecting the better single network prevents stalling and allows switching to a higher bitrate representation.

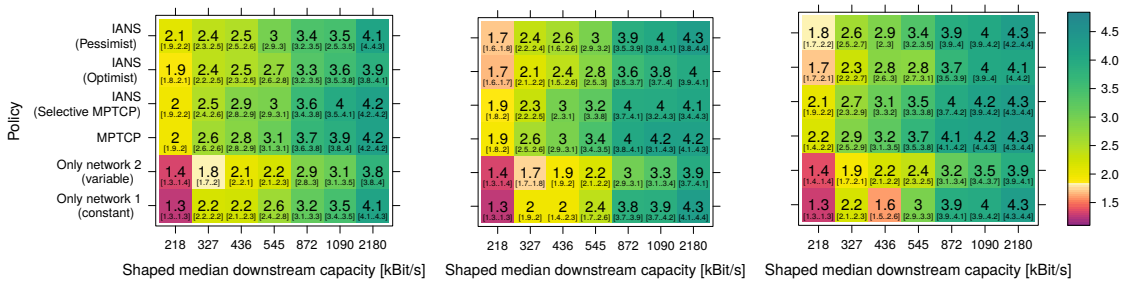
For the "Ferry" scenarios, see Figure 8.14d, ever for RB, all IANS policies outperform using either single network, e.g., improving a MOS of 2.2 or 1.9 on network 1 or network 2 to a MOS of 2.7 or 2.6 using the Optimist Policy and Pessimist Policy or even 2.9 using the Selective MPTCP Policy. Here, MPTCP yields similar MOS improvements as the Selective MPTCP Policy, as there is always sufficient capacity available to use MPTCP. For BBB, see Figure 8.14e, we again see low MOS values for the Optimist Policy and the Pessimist Policy for the 218 kBit/s scenario, as these policies can only select between the two single networks, which both provide insufficient downstream capacity to load even the lowest representation. For scenarios with capacities between 327 kBit/s and 545 kBit/s, IANS yields even higher MOS improvements for BBB than for RB. At 872 kBit/s and above, the Optimist Policy and Pessimist Policy yield a similar performance as using network 1, and only the Selective MPTCP Policy or MPTCP are able to improve performance. Finally, V generally yields similar results as RB and BBB, see Figure 8.14f. Again, we see a few cases of interactions between the workload representation and the shaped capacity, similar to the Capacity Decrease scenario. Results for the "Car", "Bus", and "Tram" scenarios, see Figure 8.14g through Figure 8.14o, are similar to the "Ferry" scenario.



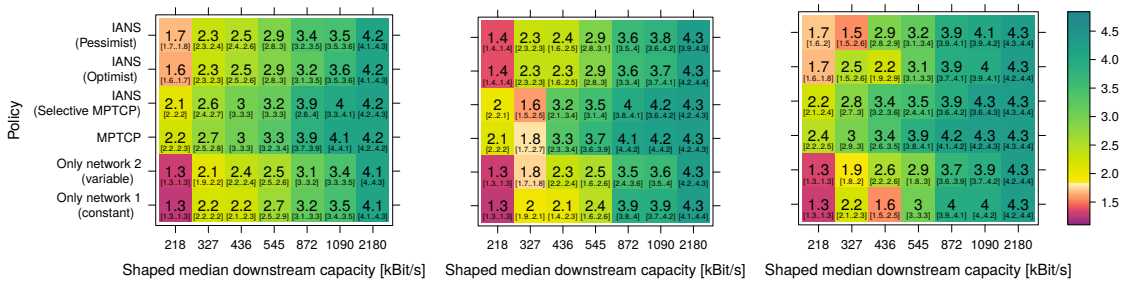
(a) RB, Train (c_v =0.6). (b) BBB, Train (c_v =0.6). (c) V, Train (c_v =0.6).



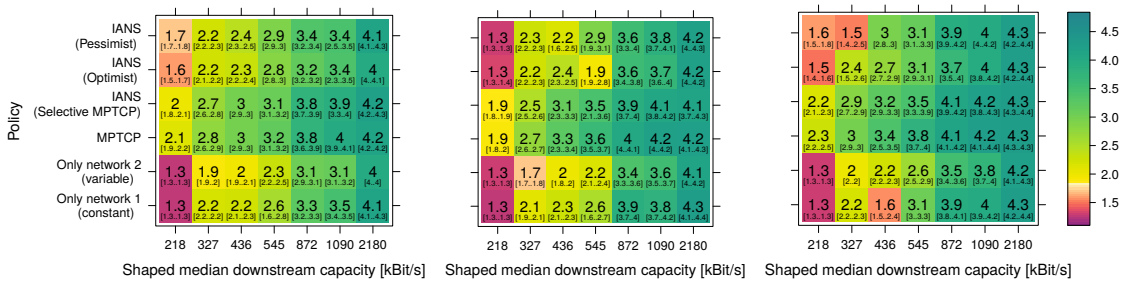
(d) RB, Ferry (c_v =0.49). (e) BBB, Ferry (c_v =0.49). (f) V, Ferry (c_v =0.49).



(g) RB, Car (c_v =0.45). (h) BBB, Car (c_v =0.45). (i) V, Car (c_v =0.45).

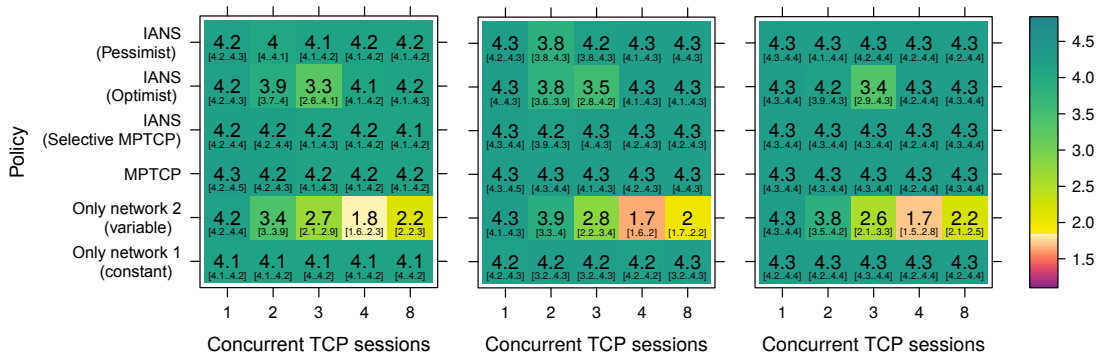


(j) RB, Bus (c_v =0.34). (k) BBB, Bus (c_v =0.34). (l) V, Bus (c_v =0.34).

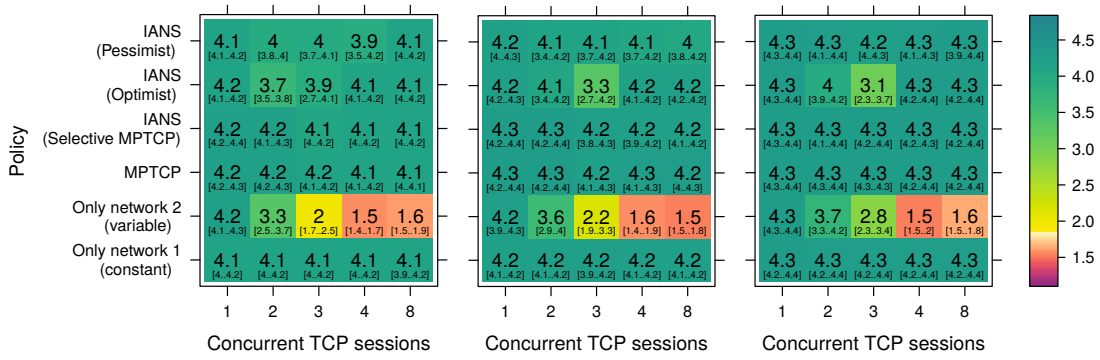


(m) RB, Tram (c_v =0.3). (n) BBB, Tram (c_v =0.3). (o) V, Tram (c_v =0.3).

Figure 8.14: Median MOS with confidence intervals for variable capacity scenarios other than Capacity Decrease



(a) RB, Latency = 10 ms. (b) BBB, Latency = 10 ms. (c) V, Latency = 10 ms.



(d) RB, Latency = 100 ms. (e) BBB, Latency = 100 ms. (f) V, Latency = 100 ms.

Figure 8.15: Median MOS with confidence intervals for scenarios with variable TCP cross-traffic.

Summary of systematic study: Our most important findings are:

1. IANS improves QoE in cases with low available downstream capacity by avoiding stalling and enabling to load higher video representations. In such scenarios, the Pessimist Policy often improves QoE compared to using only a single network and outperforms the Optimist Policy in most cases.

2. While MPTCP improves QoE in most variable capacity scenarios, it hurts performance for the Capacity Decrease scenario.

3. In such a scenario, the Selective MPTCP Policy improves QoE by avoiding using the low downstream capacity network. This shows the potential of using IANS to selectively enable MPTCP only for cases in which it benefits performance.

Going forward, IANS should combine the Pessimist Policy with the Selective MPTCP Policy by allowing it to enable MPTCP.

8.7.3 Cross-Traffic Scenarios

After varying the shaped downstream capacity on network 2, we now focus on scenarios in which we keep the shaped downstream capacity constant, but introduce variable

TCP cross-traffic, recall Table 8.1. Here, network 1 provides a constant downstream capacity of 2 Mbit/s without any cross-traffic, while downstream capacity on network 2 is constant at 5 Mbit/s with cross-traffic on the bottleneck link. We vary the latency on both networks and show results for scenarios with short latencies (10 ms) and with long latencies (100 ms).

Figure 8.15 shows a heatmap of the QoE for different cross-traffic scenarios and ABRs for all video workloads. For the RB video, Figure 8.15a shows the results for scenarios with short latencies and Figure 8.15d for the scenarios with long latencies. Within each subplot, we show scenarios with 1, 2, 3, 4, and 8 concurrent TCP sessions, i.e., harpoon sessions loading files of varying sizes. For these scenarios, each heatmap entry shows the achieved median MOS with confidence intervals for a specific scenario and access network selection policy. The color scheme is the same as in Section 8.7.2.

Here, IANS yields a high MOS even with concurrent cross-traffic of up to 8 TCP sessions, while the cross-traffic decreases the MOS for network 2. This confirms that IANS can detect the decreased downstream capacity due to cross-traffic and, therefore, uses network 1 or MPTCP. For some cases, the Optimist Policy sees slightly decreased performance, as attempts to use network 2 occasionally. Results for BBB are similar, see Figure 8.15b and Figure 8.15e. However, for BBB, we see an interaction between the shaped downstream capacity on network 1 and the BOLA_BASIC ABR. Here, BOLA_BASIC continuously switches representations, which leads to a decreased MOS on network 1 for some cases. Results for V are similar to BBB and RB, see Figure 8.15c and 8.15f.

Summary for cross-traffic scenarios: For cases with high current cross-traffic on a network, IANS avoids using this network, thus, it avoids MOS degradations.

9 Conclusion

In this thesis, we study access network selection informed by application needs and network performance characteristics to improve application performance. In this chapter, we summarize our findings, discuss lessons learned, and outline future work.

9.1 Summary

Performance bottlenecks leading to adverse network conditions, such as long latency or low downstream capacity, may degrade application performance and lead to an unsatisfactory Quality of Experience (QoE). Due to such performance bottlenecks, a single access network may not be able to satisfy application needs. However, end-user devices often have a choice between multiple access networks. We argue that using these networks efficiently can help overcome some of the performance bottlenecks, and, thus, improve application performance. Therefore, we introduce Informed Access Network Selection (IANS) to select the most suitable access network(s) based on application needs and network performance characteristics. Then, we show how IANS can improve the performance of Web browsing and HTTP Adaptive Streaming (HAS).

First, we investigate how to accurately assess application performance for Web browsing. We find several Web performance pitfalls, e.g., regarding initial redirects for load times and data sources for resource sizes, and derive guidelines regarding Web performance measurement.

To achieve IANS, we enable applications to express what to optimize for when selecting an access network. To this end, we design Socket Intents as hints about what an application knows, expects, or wants to achieve. In contrast to Quality of Service (QoS) requirements, Socket Intents are considered in a best-effort manner and help to use the available resources in the most efficient way. Given that the actually available information may depend on the application and its context, we specify different Intents: For example, applications can set the Traffic Category to indicate what network performance characteristics to optimize for, e.g., to optimize small transfers for short latency and large transfers for high downstream capacity. Alternatively, applications can provide more detailed properties such as the Size to be Received for a transfer, so IANS can decide what network performance characteristics to optimize for. To match application needs to the most suitable access networks, we collect

estimates of the current network performance characteristics, such as latency and downstream capacity, based on existing traffic.

Combining both Socket Intents and network performance characteristics, we design IANS policies to improve the performance of Web browsing and HAS. For Web browsing, we define the Threshold Policy, which distributes Web resource loads based on the Size to be Received as well as current latency and available downstream capacity estimates. For HAS, we define three different IANS policies: The Optimist Policy, the Pessimist Policy, and the Selective MPTCP Policy. While the Optimist Policy and the Pessimist Policy select the most suitable access network according to downstream capacity estimates on different time scales, the Selective MPTCP Policy enables Multipath TCP (MPTCP) only for transfers with the Traffic Category set to Bulk, i.e., the application wants to optimize for high downstream capacity, as MPTCP adds overhead for other transfers while it risks not providing any benefit. Furthermore, MPTCP is only used if sufficient downstream capacity is available, i.e., if MPTCP is unlikely to suffer from extensive head-of-line blocking, to prevent performance degradations.

We implement IANS within the Socket Intents prototype. For application needs, we design enhanced networking Application Programming Interfaces (APIs) through which an application can specify its Socket Intents. To determine network performance characteristics, we collect and aggregate estimates within the prototype. We enable the following applications to benefit from IANS: For Web browsing, we write a Web proxy setting the Size to be Received for each transfer, so we can load each resource over the most suitable access network selected by IANS. For HAS, we modify a video player to set the Bitrate Received of the representation for the next segment and the Duration of the current buffer level, so IANS can choose a network with short predicted load time while preventing stalls due to an empty buffer.

To evaluate the benefits of IANS for Web browsing and HAS, we use both a controlled testbed and servers “in the wild”. For Web browsing, we find that IANS yields shorter load times than a single network or MPTCP in particular for scenarios with asymmetric network performance characteristics, in which one access network provides short latency but low downstream capacity and the other network provides high capacity but long latency. In such cases, IANS even outperforms MPTCP because MPTCP can experience performance problems caused by self-induced congestion on the low downstream capacity network. Using Web servers “in the wild”, we confirm that IANS shows the same speedups as in our testbed, as it does not require server-sided support. In contrast, MPTCP is not able to provide the same benefits an in the testbed because MPTCP deployment is still limited. For HAS, we find that IANS improves QoE for scenarios with low downstream capacities. Here, IANS helps overcome limitations of MPTCP by enabling MPTCP only if sufficient downstream capacity is available.

9.2 Discussion and Lessons Learned

Application Knowledge Benefits Access Network Selection. From our evaluation, we learn that IANS improves application performance, particularly in scenarios with low downstream capacities or under asymmetric network conditions. In such scenarios, IANS provides a major advantage compared to application-agnostic MPTCP, e.g., by reducing relevant Web performance metrics such as Above-The-Fold time (ATF) by between 500 and 1000 ms in the median. Moreover, IANS can complement MPTCP because it works on a different granularity, i.e., it distributes transfers instead of subflows of a single transfer.

Web Browsing: Providing Application Knowledge is Challenging. For Web browsing, our application specifies the Size to be Received prior to loading each resource. We find that having such knowledge available is challenging. In practice, this information could be included in the metadata, e.g., in the Web page that references the resource. For dynamic resources, the Size to be Received could be injected into the page by the Web server dynamically, however, this increases server load. For our evaluation, to make the Size to be Received available for unmodified Web pages, we use a two-step download. While this introduces a performance overhead, it does not require modifying the Web page. Here, there is a tradeoff between the extent of the modifications to an application and the achievable performance optimization for the Size to be Received. Other Socket Intents may be more readily available, for example, the Size to be Sent for uploaded files.

What Socket Intents to Set: Traffic Properties or Network Performance Characteristics. We specify Socket Intents to either set properties of its transfers, such as the Size to be Received, or to set what network performance characteristics to optimize for, e.g., for short latency or high downstream capacity. While predicting the properties of future transfers may be challenging, e.g., due to dynamically generated resources with changing sizes, such properties may provide useful hints to IANS policies, as they enable calculations such as load time predictions. In contrast, setting what network performance characteristics to optimize for may be easier for some applications, but it embeds assumptions in the application code. This may be a problem in case an application is not updated when an assumption no longer holds. However, we note that some applications already embed such assumptions in the implementation, e.g., by always preferring WiFi under the assumption that it provides better performance than cellular. Finally, setting an Intent of course only makes sense if there are IANS policies available that can make good decisions based on it.

Implementation Experience When Integrating Socket Intents Into Applications. When implementing support for the Socket Intents APIs into applications, we find that while this is feasible with limited effort for some applications, it is more challenging for others. Relevant applications often implement their own connection abstraction on top of the Socket API. Therefore, instead of, e.g., simply replacing existing Socket API calls with the calls of our enhanced Socket API, we have to integrate our API calls into the existing connection abstraction. For instance, when

modifying the video player presented in Section 7.4.2, we have to provide the socket returned by the Socketconnect API to a data structure within the application. While integrating our synchronous Socketconnect API into the video player is feasible because the player uses threads, it is not feasible to integrate this API into applications using asynchronous networking via callbacks. To support such applications, future work should add a callback-based API. Alternatively, it may be beneficial to implement Socket Intents support into a general purpose HyperText Transfer Protocol (HTTP) library that can be used by multiple applications that use such a library, e.g., mobile apps. However, highly complex applications such as Web browsers are often monolithic and use their own internal HTTP library, so we would have to modify each browser individually. While it would be feasible to implement Socket Intents into a browser, we opt for a Web proxy for this thesis for multiple reasons. First, using a Web proxy allows us to support multiple different browsers. Second, browser release cycles are short, so any third party modifications may quickly become out of date and, therefore, unrepresentative of the Web performance experienced by actual users. However, if support for an enhanced networking API were included in the main branch of the browser and, thus, included in new releases, the best option would be to implement Socket Intents directly in the browser.

API Design: Limitations of Sockets. When designing APIs for applications to express Socket Intents for their connections or transfers, we use sockets as an abstraction. We find that this works well, e.g., when IANS policies distribute HTTP/1.1 transfers across Transmission Control Protocol (TCP) connections on different access networks. However, one drawback of using sockets is that the application has to manage additional information for each socket, e.g., a Transport Layer Security (TLS) context. For more recent versions of HTTP, such as HTTP/2 and HTTP/3, which uses Quick UDP Internet Connection (QUIC), IANS can still distribute transfers, e.g., as new streams on a single TCP or QUIC connection on each access network. However, it is infeasible to use sockets as an abstraction for streams, as streams are usually implemented on top of sockets. Therefore, an enhanced networking API has to present a different abstraction to applications to support HTTP/2 or QUIC. The TAPS API [56], provides such an abstraction, i.e., a generic connection over which an application can send and receive messages. Underneath, this connection may correspond to a TCP connection encrypted by TLS, but also to one or multiple QUIC streams onto which messages can be distributed. This not only relieves the application of a major share of the burden to manage connections but also enables per-transfer IANS for HTTP/2 and QUIC.

9.3 Future Work

Changing Bottlenecks: Shift Focus to Path Selection. Access network performance is continuously improving, e.g., with the evolution of uplinks and of 4G cellular networks and with the upcoming deployment of 5G. Communication systems using these technologies may provide short latencies and high downstream capacities in the

access network. Therefore, performance bottlenecks may shift from the access network to the backbone, as observed by Baranasuriya et al. [5]. Even if bottlenecks are no longer in the access networks, still, IANS can select between different paths with different network performance characteristics to the same destination. Over these different paths, different remote hosts may serve the same content, e.g., two different Content Delivery Network (CDN) replica. In such cases, there is additional potential for optimization using IANS. However, shifting our focus from access network selection to path selection makes it more difficult to use performance measurements obtained for one path to estimate the performance for a different path over the same access network. We conclude that future work should differentiate network performance characteristics not by the access network, but by, e.g., the remote host or subnet. Once collected, such estimates should be made available in a unified way, such that multiple different path selection technologies, e.g., a TAPS implementation or a connection manager, can benefit from this information.

Advanced Access Network Selection Policies. To select an access network or path for a new application or transfer, performance estimates may not be available or they may be outdated. Therefore, to use the access network or path with the shortest latency, IANS should open multiple concurrent connections over the available access networks or paths. Then, it should use the connection that connects first, similar to the "Happy Eyeballs"-inspired approach taken by NEAT [55] and TAPS [82]. More generally, an IANS policy could explicitly build an adaptive control loop by learning about the actual performance experienced by transfers, e.g., load times. Based on this feedback, the IANS policy could adapt and optimize its decisions.

Exploring Access Network Selection for More Applications. In addition to Web browsing and HAS, more applications may benefit from IANS. For example, future work should consider other HTTP-based applications than Web browsing, which have diverse requirements: An audio streaming application may interactively stream content and prefetch other content at the same time. In parallel, other applications may, e.g., synchronize files from a mobile end-user device to a server, perform updates, or issue interactive queries for the user to look up specific information. These transfers may have different Timeliness requirements [83] and the Size to be Received may be more readily available for such applications than for Web browsing.

Access Network Selection in the Real World. To satisfy the needs of actual users, access network selection needs to take other objectives than performance into account. For example, using a cellular interface may be limited by a data plan and the readiness to use multiple access networks may be influenced by the current battery status. Here, future work should include the Cost Preference or Energy Efficiency Socket Intents. To match these Socket Intents to networks, end-user devices should gather information on the current battery status or data plan information. Furthermore, different entities may be interested in defining access network selection policies, e.g., the user, operating system, network provider, or service provider. These different entities may prioritize different objectives. For example, while a user may prioritize application performance, a network provider may prioritize distributing network load. Therefore, there should be a framework to express and reconcile

different access network selection policies and objectives. Designing such a framework is not trivial, e.g., as it may have to make complex decisions transparent to the user or to other entities.

9.4 Outlook: Better APIs for a Better Internet

Providing an enhanced networking API such as the TAPS API [56] has benefits in addition to IANS. Such an API can simplify applications by relieving them of the burden of advanced connection management. Furthermore, if protocol details are handled within the connection abstraction underneath the API, it becomes easier to deploy new protocols. For example, in addition to automatically preferring IPv6 over IPv4, applications may be able to use new transport protocols such as QUIC without being modified. Therefore, enhanced networking APIs have the potential to improve the state of the art of the Internet.

Glossary

ABR Adaptive Bit-Rate algorithm: Algorithm used by a HAS player to adapt the loaded media representation to the conditions of the network and the player. 26, 42–44, 70, 71, 84, 91–93, 106–109, 112

AP Access Point. 7, 8, 11, 15, 20–22, 56, 85, 94

API Application Programming Interface. v, 4, 7, 28–32, 34, 35, 44, 47, 77–84, 91, 114–116, 118

ATF Above-The-Fold time: The time taken to load all user-visible content of a Web page. 5, 34, 91, 96–98, 100–102, 104, 115, 131

BSS Besic Service Set: Connectivity provided by a single WiFi Access Point. 11

c_v coefficient of variation. 87, 88, 105–108, 110, 131

CDN Content Delivery Network. 27, 42, 60, 87, 117

CPE Customer Premises Equipment. 12

CQM A model that predicts QoE over time during video playout [81].. 91–93

DASH Dynamic Adaptive Streaming over HTTP. 42, 83

DNS Domain Name System. 36, 37, 40, 60, 61, 79, 81, 82, 87

DOM Document Object Model. 34, 35

DSCP Differentiated Services Code Point. 46

DSL Digital Subscriber Line. 12–14

EBSS Extended Basic Service Set: Connectivity provided by a WiFi network including all its Access Points. 8, 11

ECDF Empirical Cumulative Distribution Function. 96–98, 104, 131

HAR HTTP Archive: Format to store HTTP requests and responses. 34–41, 91, 104

HAS HTTP Adaptive Streaming. v, 3–5, 33, 41–44, 47, 59, 61, 70, 71, 74, 77, 82, 83, 85–89, 91–93, 113, 114, 117

HTTP HyperText Transfer Protocol. 17, 26–28, 31, 35, 37–42, 69, 80, 83, 104, 116, 117

HTTPS HTTP secured by TLS. 37, 83

IANS Informed Access Network Selection. v, xvi, 3–5, 22, 27, 33, 41, 43–47, 49, 59, 61–63, 69–71, 74, 77–79, 82–89, 91–93, 96–109, 111–118

IETF Internet Engineering Task Force. 20, 32, 45

IFOM IP flow mobility for Proxy Mobile IPv6. 20–22

ISP Internet Service Provider. 8, 12–15, 19

LAN Local Area Network. 7, 8, 11–14, 20

MAM Multi Access Manager: Component of the Socket Intents prototype that hosts the policies and gathers network characteristics. 77, 78, 80–82, 94, 95

mHTTP Multi-Source Multipath HTTP. 27

MOS Mean Opinion Score: User satisfaction on a scale from 1 to 5. v, 91–93, 96, 99, 100, 102, 104, 105, 107–112, 131, 132

MPD Media Presentation Description. 42

MP-H2 Multipath HTTP/2. 27

MPQUIC Multipath QUIC. 27

MPTCP Multipath TCP. v, 2, 4, 5, 7, 23–27, 30, 31, 52, 59, 60, 74, 75, 86, 92, 96–104, 106–109, 111, 112, 114, 115, 131

NAT Network Address Translation. 17, 24, 25, 51

OS Operating System. 7, 24, 25, 27–32, 78, 80

OTT Over-The-Top: Solution without network integration. 21, 22

PLT Page Load Time. 34, 37, 91, 92, 96, 99, 101–104, 131, 133

QoE Quality of Experience. v, 1, 5, 20–22, 41, 44, 70, 90, 91, 105–109, 111, 113, 114

QoS Quality of Service. 32, 46, 113

QUIC Quick UDP Internet Connection: A UDP-Based multiplexed and secure transport protocol [37, 38]. 26–29, 40, 53, 55, 56, 80, 116, 118

RSS Received Signal Strength. 10, 21, 22, 50, 56, 71, 87

RTP Real-Time Transport Protocol. 27, 41

RTT Round Trip Time. 17, 22, 26, 27, 53, 54, 56, 94, 95

SCTP Stream Control Transmission Protocol. 26, 55

SNR Signal-to-Noise-Ratio. 10, 12

SRTT Smoothed Round Trip Time. 4, 53, 54, 68, 81, 94, 95, 131

TAPS Transport Services: Working Group in the IETF that develops a common abstraction on top of different transport protocols [56]. 32, 116–118

TCP Transmission Control Protocol. 2, 17, 23–31, 37, 38, 40, 47, 52–56, 61, 65, 67, 68, 80, 81, 83, 86–88, 94, 95, 101, 103, 111, 112, 116, 131, 132

TCP/IP Protocol stack based on TCP and IP. 29

TLS Transport Layer Security. 28, 31, 35, 37, 38, 65, 67, 80, 83, 92, 116

TTFP Time To First Paint: The time until the first content of a Web page is rendered. 34, 36, 37

UDP User Datagram Protocol. 28, 29, 55, 86, 87, 94, 95

URL Uniform Resource Locator. 34, 37, 40, 42, 89

VoIP Voice over IP. 20–22

Bibliography

[1] Joel Sommers and Paul Barford. "Cell vs. WiFi: on the performance of metro area mobile connections". In: *Proceedings of the 2012 Internet Measurement Conference*. ACM. 2012, pp. 301–314 (cit. on pp. 1, 17, 94).

[2] Shuo Deng, Ravi Netravali, Anirudh Sivaraman, and Hari Balakrishnan. "Wifi, lte, or both?: Measuring multi-homed wireless internet performance". In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM. 2014, pp. 181–194 (cit. on pp. 1, 2, 17, 26).

[3] Srikanth Sundaresan, Nick Feamster, and Renata Teixeira. "Home network or access link? locating last-mile downstream throughput bottlenecks". In: *International Conference on Passive and Active Network Measurement*. Springer. 2016, pp. 111–123 (cit. on pp. 1, 11, 14, 67).

[4] Changhua Pei, Youjian Zhao, Guo Chen, Ruming Tang, Yuan Meng, Minghua Ma, Ken Ling, and Dan Pei. "WiFi can be the weakest link of round trip network latency in the wild". In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, pp. 1–9 (cit. on p. 1).

[5] Nimantha Baranasuriya, Vishnu Navda, Venkata N Padmanabhan, and Seth Gilbert. "QProbe: locating the bottleneck in cellular communication". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM. 2015, p. 33 (cit. on pp. 1, 15, 117).

[6] Sebastian Egger, Tobias Hossfeld, Raimund Schatz, and Markus Fiedler. "Waiting times in quality of experience for web based services". In: *2012 Fourth International Workshop on Quality of Multimedia Experience*. IEEE. 2012, pp. 86–96 (cit. on p. 1).

[7] Sandvine. *The Mobile Internet Phenomena Report*. `https://www.sandvine.com/hubfs/downloads/phenomena/2019-mobile-phenomena-report.pdf`. 2019 (cit. on p. 1).

[8] Sunghwan Ihm and Vivek S Pai. "Towards understanding modern web traffic". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM. 2011, pp. 295–312 (cit. on p. 1).

[9] Feng Qian, Subhabrata Sen, and Oliver Spatscheck. "Characterizing resource usage for mobile web browsing". In: *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM. 2014, pp. 218–231 (cit. on p. 1).

[10] Edward J Oughton and Zoraida Frias. *Exploring the Cost, Coverage and Rollout Implications of 5G in Britain*. `https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/577965/Exploring_the_Cost_Coverage_and_Rollout_Implications_of_5G_in_Britain_-_Oughton_and_Frias_report_for_the_NIC.pdf`. 2016 (cit. on p. 1).

[11] OECD. *Communications Outlook*. http://www.oecd.org/sti/broadband/communications-outlook.htm. 2013 (cit. on pp. 2, 14).

[12] OECD. *Digital Economy Outlook*. https://www.oecd.org/sti/oecd-digital-economy-outlook-2017-9789264276284-en.htm. 2017 (cit. on pp. 2, 14).

[13] European Commission. *Study on Broadband Coverage in Europe*. https://ec.europa.eu/digital-single-market/en/news/study-broadband-coverage-europe-2017. 2017 (cit. on pp. 2, 14).

[14] Akamai. *State of the Internet / Connectivity Report*. https://www.akamai.com/us/en/resources/our-thinking/state-of-the-internet-report/global-state-of-the-internet-connectivity-reports.jsp. 2017 (cit. on p. 2).

[15] M. Wasserman and P. Seite. *Current Practices for Multiple-Interface Hosts*. RFC 6419 (Informational). RFC. Fremont, CA, USA: RFC Editor, Nov. 2011. DOI: 10.17487/RFC6419. URL: https://www.rfc-editor.org/rfc/rfc6419.txt (cit. on pp. 2, 31).

[16] A. Ford et al. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6824 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Jan. 2013. DOI: 10.17487/RFC6824. URL: https://www.rfc-editor.org/rfc/rfc6824.txt (cit. on pp. 2, 23).

[17] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP". In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 2012, pp. 399–412 (cit. on pp. 2, 24, 25).

[18] Yung-Chih Chen, Yeon-sup Lim, Richard J Gibbens, Erich M Nahum, Ramin Khalili, and Don Towsley. "A measurement-based study of multipath tcp performance over wireless networks". In: *Proceedings of the 2013 conference on Internet measurement conference*. ACM. 2013, pp. 455–468 (cit. on pp. 2, 25).

[19] Bo Han, Feng Qian, Shuai Hao, and Lusheng Ji. "An anatomy of mobile web performance over multipath TCP". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM. 2015, p. 5 (cit. on pp. 2, 26).

[20] IEEE. "IEEE Standard for Definitions of Terms for Antennas". In: *IEEE Std 145-2013 (Revision of IEEE Std 145-1993)* (Mar. 2014), pp. 1–50. DOI: 10.1109/IEEESTD.2014.6758443 (cit. on p. 9).

[21] René-Jean Essiambre, Gerard Foschini, Peter Winzer, and Gerhard Kramer. "Capacity limits of fiber-optic communication systems". In: *Optical Fiber Communication Conference*. Optical Society of America. 2009, OThL1 (cit. on p. 13).

[22] Matthew Sargent and Mark Allman. "Performance within a fiber-to-the-home network". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 22–30 (cit. on p. 13).

[23] Péter Szilágyi and Csaba Vulkán. "LTE user plane congestion detection and analysis". In: *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE. 2015, pp. 1819–1824 (cit. on p. 15).

[24] Jeffrey Erman, Vijay Gopalakrishnan, Rittwik Jana, and Kadangode K Ramakrishnan. "Towards a spdy'ier mobile web?" In: *IEEE/ACM Transactions on Networking* 23.6 (2015), pp. 2010–2023 (cit. on p. 15).

[25] Florian Metzger, Albert Rafetseder, Peter Romirer-Maierhofer, and Kurt Tutschku. "Exploratory analysis of a GGSN's PDP context signaling load". In: *Journal of Computer Networks and Communications* 2014 (2014) (cit. on p. 15).

[26] Ookla. *Speedtest Market Reports*. `https://www.speedtest.net/reports/`, last accessed 21. May 2019. 2018 (cit. on p. 17).

[27] R. Braden (Ed.) *Requirements for Internet Hosts - Communication Layers*. RFC 1122 (Internet Standard). RFC. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864, 8029. Fremont, CA, USA: RFC Editor, Oct. 1989. DOI: `10.17487/RFC1122`. URL: `https://www.rfc-editor.org/rfc/rfc1122.txt` (cit. on p. 19).

[28] Adnan Aijaz, Hamid Aghvami, and Mojdeh Amani. "A survey on mobile data offloading: technical and business perspectives". In: *IEEE Wireless Communications* 20.2 (2013), pp. 104–112 (cit. on p. 20).

[29] Rachad Maallawi, Nazim Agoulmine, Benoit Radier, and Tayeb Ben Meriem. "A comprehensive survey on offload techniques and management in wireless access and core networks". In: *IEEE Communications Surveys & Tutorials* 17.3 (2015), pp. 1582–1604 (cit. on p. 20).

[30] Sven Wiethölter, Marc Emmelmann, Robert Andersson, and Adam Wolisz. "Performance evaluation of selection schemes for offloading traffic to IEEE 802.11 hotspots". In: *2012 IEEE International Conference on Communications (ICC)*. IEEE. 2012, pp. 5423–5428 (cit. on p. 21).

[31] Claudio Rossi, Narseo Vallina-Rodriguez, Vijay Erramilli, Yan Grunenberger, Laszlo Gyarmati, Nikolaos Laoutaris, Rade Stanojevic, Konstantina Papagiannaki, and Pablo Rodriguez. "3GOL: Power-boosting ADSL using 3G OnLoading". In: *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM. 2013, pp. 187–198 (cit. on p. 21).

[32] Shubhada Gadgil, Shashi Ranjan, Divya Joshi, Mahima Mehta, Nadeem Akhtar, and Abhay Karandikar. "Performance evaluation and viability of IFOM in heterogeneous LTE—WLAN network". In: *2015 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2015, pp. 1524–1529 (cit. on p. 21).

[33] Mikhail Gerasimenko, Nageen Himayat, Shu-ping Yeh, Shilpa Talwar, Sergey Andreev, and Yevgeni Koucheryavy. "Characterizing performance of load-aware network selection in multi-radio (WiFi/LTE) heterogeneous networks". In: *2013 IEEE Globecom Workshops (GC Wkshps)*. IEEE. 2013, pp. 397–402 (cit. on p. 22).

[34] Florian Wamser, Thomas Zinner, Phuoc Tran-Gia, and Jing Zhu. "Dynamic bandwidth allocation for multiple network connections: improving user QoE and network usage of YouTube in mobile broadband". In: *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*. ACM. 2014, pp. 57–62 (cit. on p. 22).

[35] Shuo Deng, Anirudh Sivaraman, and Hari Balakrishnan. "All your network are belong to us: A transport framework for mobile network selection". In: *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*. ACM. 2014, p. 19 (cit. on p. 22).

[36] C. Raiciu, M. Handley, and D. Wischik. *Coupled Congestion Control for Multipath Transport Protocols*. RFC 6356 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Oct. 2011. DOI: 10.17487/RFC6356. URL: https://www.rfc-editor.org/rfc/rfc6356.txt (cit. on p. 25).

[37] Jana Iyengar and Martin Thompson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet Draft draft-ietf-quic-transport-20. IETF, Apr. 2019 (cit. on pp. 26, 120).

[38] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. "The quic transport protocol: Design and internet-scale deployment". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM. 2017, pp. 183–196 (cit. on pp. 26, 120).

[39] Christoph Paasch, Simone Ferlin, Ozgu Alay, and Olivier Bonaventure. "Experimental evaluation of multipath TCP schedulers". In: *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*. ACM. 2014, pp. 27–32 (cit. on p. 26).

[40] Hang Shi, Yong Cui, Xin Wang, Yuming Hu, Minglong Dai, Fanzhao Wang, and Kai Zheng. "STMS: Improving MPTCP Throughput Under Heterogeneous Networks". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 2018, pp. 719–730 (cit. on p. 26).

[41] Yihua Ethan Guo, Ashkan Nikravesh, Z Morley Mao, Feng Qian, and Subhabrata Sen. "Accelerating multipath transport through balanced subflow completion". In: *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM. 2017, pp. 141–153 (cit. on p. 26).

[42] Yeon-sup Lim, Erich M Nahum, Don Towsley, and Richard J Gibbens. "ECF: An MPTCP path scheduler to manage heterogeneous paths". In: *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM. 2017, pp. 147–159 (cit. on p. 26).

[43] HyunJong Lee, Jason Flinn, and Basavaraj Tonshal. "RAVEN: Improving Interactive Latency for the Connected Car". In: *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM. 2018, pp. 557–572 (cit. on p. 26).

[44] Bo Han, Feng Qian, Lusheng Ji, and Vijay Gopalakrishnan. "MP-DASH: Adaptive video streaming over preference-aware multipath". In: *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM. 2016, pp. 129–143 (cit. on p. 26).

[45] R. Stewart (Ed.) *Stream Control Transmission Protocol*. RFC 4960 (Proposed Standard). RFC. Updated by RFCs 6096, 6335, 7053. Fremont, CA, USA: RFC Editor, Sept. 2007. DOI: 10.17487/RFC4960. URL: https://www.rfc-editor.org/rfc/rfc4960.txt (cit. on p. 26).

[46] Varun Singh, Saba Ahsan, and Jörg Ott. "MPRTP: multipath considerations for real-time media". In: *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM. 2013, pp. 190–201 (cit. on p. 27).

[47] Juhoon Kim, Yung-Chih Chen, Ramin Khalili, Don Towsley, and Anja Feldmann. "Multi-source multipath HTTP (mHTTP): A proposal". In: *ACM SIGMETRICS Performance Evaluation Review* 42.1 (2014), pp. 583–584 (cit. on p. 27).

[48] Ashkan Nikravesh, Yihua Guo, Xiao Zhu, Feng Qian, and Z Morley Mao. "MP-H2: A Client-only Multipath Solution for HTTP/2". In: *MobiCom. ACM* (2019) (cit. on p. 27).

[49] Quentin De Coninck and Olivier Bonaventure. "Multipath quic: Design and evaluation". In: *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*. ACM. 2017, pp. 160–166 (cit. on p. 27).

[50] D. Thaler (Ed.) et al. *Default Address Selection for Internet Protocol Version 6 (IPv6)*. RFC 6724 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Sept. 2012. DOI: 10.17487/RFC6724. URL: https://www.rfc-editor.org/rfc/rfc6724.txt (cit. on p. 30).

[51] Olivier Bonaventure. *Apple uses Multipath TCP*. http://blog.multipath-tcp.org/blog/html/2018/12/15/apple_and_multipath_tcp.html. 2018 (cit. on p. 31).

[52] Olivier Bonaventure. *Which servers use Multipath TCP?* http://blog.multipath-tcp.org/blog/html/2018/12/19/which_servers_use_multipath_tcp.html. 2018 (cit. on p. 31).

[53] Hasan Abbasi, Christian Poellabauer, Karsten Schwan, Gregory Losik, and Richard West. "A quality-of-service enhanced socket API in GNU/Linux". In: *Proceedings of the 4th Real-Time Linux Workshop, Boston, Massachusetts*. Citeseer. 2002 (cit. on p. 31).

[54] Brett D Higgins, Azarias Reda, Timur Alperovich, Jason Flinn, Thomas J Giuli, Brian Noble, and David Watson. "Intentional networking: opportunistic exploitation of mobile network diversity". In: *Proceedings of the sixteenth annual international conference on Mobile computing and networking*. ACM. 2010, pp. 73–84 (cit. on p. 32).

[55] Naeem Khademi, David Ros, Michael Welzl, Zdravko Bozakov, Anna Brunstrom, Gorry Fairhurst, Karl-Johan Grinnemo, David Hayes, Per Hurtig, Tom Jones, et al. "NEAT: a platform-and protocol-independent internet transport API". In: *IEEE Communications Magazine* 55.6 (2017), pp. 46–54 (cit. on pp. 32, 117).

[56] Brian Trammell, Michael Welzl, Theresa Enghardt, Gorry Fairhust, Mirja Kuehlewind, Colin Perkins, Philipp Tiesel, and Christopher Wood. *An Abstract Application Layer Interface to Transport Services (work in progress)*. Internet Draft draft-ietf-taps-interface-05. IETF, Nov. 2019 (cit. on pp. 32, 116, 118, 121).

[57] Diego Neves da Hora, Alemnew Sheferaw Asrese, Vassilis Christophides, Renata Teixeira, and Dario Rossi. "Narrowing the gap between QoS metrics and Web QoE using Above-the-fold metrics". In: *International Conference on Passive and Active Network Measurement*. Springer. 2018, pp. 31–43 (cit. on pp. 34, 91).

[58] Enrico Bocchi, Luca De Cicco, and Dario Rossi. "Measuring the quality of experience of web users". In: *ACM SIGCOMM Computer Communication Review* 46.4 (2016), pp. 8–13 (cit. on pp. 35, 40, 91).

[59] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D Strowes, and Narseo Vallina-Rodriguez. "A long way to the top: significance, structure, and stability of internet top lists". In: *Proceedings of the Internet Measurement Conference 2018*. ACM. 2018, pp. 478–493 (cit. on pp. 36, 88).

[60] Theresa Enghardt, Thomas Zinner, and Anja Feldmann. "Web Performance Pitfalls". In: *PAM*. Springer. 2019, pp. 286–303 (cit. on pp. 41, 88, 89).

[61] Thomas Stockhammer. "Dynamic adaptive streaming over HTTP–: standards and design principles". In: *Proceedings of the second annual ACM conference on Multimedia systems*. ACM. 2011, pp. 133–144 (cit. on p. 42).

[62] Jonathan Kua, Grenville Armitage, and Philip Branch. "A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP". In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1842–1866 (cit. on p. 43).

[63] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. "A buffer-based approach to rate adaptation: Evidence from a large video streaming service". In: *ACM SIGCOMM Computer Communication Review* 44.4 (2015), pp. 187–198 (cit. on pp. 43, 84, 92).

[64] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. "BOLA: Near-optimal bitrate adaptation for online videos". In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, pp. 1–9 (cit. on pp. 43, 84, 92).

[65] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. "A survey on quality of experience of HTTP adaptive streaming". In: *IEEE Communications Surveys & Tutorials* 17.1 (2015), pp. 469–492 (cit. on pp. 44, 90).

[66] Alexander Raake, Marie-Neige Garcia, Werner Robitza, Peter List, Steve Göring, and Bernhard Feiten. "A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1". In: *Ninth International Conference on Quality of Multimedia Experience (QoMEX)*. Erfurt: IEEE, May 2017. ISBN: 978-1-5386-4024-1. DOI: 10.1109/QoMEX.2017.7965631. URL: http://ieeexplore.ieee.org/document/7965631/ (cit. on pp. 44, 90, 91, 93).

[67] Werner Robitza, Steve Göring, Alexander Raake, David Lindegren, Gunnar Heikkilä, Jörgen Gustafsson, Peter List, Bernhard Feiten, Ulf Wüstenhagen, Marie-Neige Garcia, Kazuhisa Yamagishi, and Simon Broom. "HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P.1203 – Open Databases and Software". In: *9th ACM Multimedia Systems Conference*. Amsterdam, 2018. ISBN: 9781450351928. DOI: 10.1145/3204949.3208124 (cit. on pp. 44, 91).

[68] Michael Seufert, Nikolas Wehner, and Pedro Casas. "Studying the Impact of HAS QoE Factors on the Standardized QoE Model P. 1203". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, pp. 1636–1641 (cit. on p. 44).

[69] Philipp S. Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. "Socket Intents: Leveraging Application Awareness for Multi-access Connectivity". In: *ACM CoNEXT*. Santa Barbara, California, USA: ACM, 2013, pp. 295–300. ISBN: 978-1-4503-2101-3. DOI: 10.1145/2535372.2535405 (cit. on p. 45).

[70] Philipp Tiesel, Theresa Enghardt, and Anja Feldmann. *Socket Intents (work in progress)*. Internet Draft draft-tiesel-taps-socketintents-01. IETF, Oct. 2017 (cit. on p. 45).

[71] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. *An architecture for differentiated services*. Tech. rep. RFC 2475. IETF, 1998 (cit. on p. 46).

[72] Theresa Enghardt and Cyrill Krähenbühl. *A Vocabulary of Path Properties (work in progress)*. Internet Draft draft-enghardt-panrg-path-properties-03. IETF, Nov. 2019 (cit. on p. 52).

[73] Theresa Enghardt, Philipp S Tiesel, and Anja Feldmann. "Metrics for access network selection". In: *Proceedings of the Applied Networking Research Workshop*. ACM. 2018, pp. 67–73 (cit. on p. 52).

[74] Mirko Palmer. "Implementation and Evaluation of Multi-Access Policies for MPTCP Path Management in User-Space". MA thesis. TU Berlin, 2015 (cit. on p. 60).

[75] Joel Sommers, Hyungsuk Kim, Paul Barford, and Paul Barford. "Harpoon: a flow-level traffic generator for router and network tests". In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 32. 1. ACM. 2004, pp. 392–392 (cit. on p. 86).

[76] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. "Commute path bandwidth traces from 3G networks: analysis and applications". In: *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM. 2013, pp. 114–118 (cit. on pp. 87, 88, 105).

[77] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. "Mahimahi: Accurate Record-and-Replay for HTTP". In: *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. 2015, pp. 417–429 (cit. on p. 89).

[78] Stefan Lederer, Christopher Müller, and Christian Timmerer. "Dynamic adaptive streaming over HTTP dataset". In: *Proceedings of the 3rd Multimedia Systems Conference*. ACM. 2012, pp. 89–94 (cit. on pp. 89, 90, 92).

[79] YouTube Help. *Live encoder settings, bitrates, and resolutions*. `https://support.google.com/youtube/answer/2853702?hl=en`, last accessed 25. June 2019. 2019 (cit. on p. 90).

[80] Tobias Hoßfeld, Florian Metzger, and Dario Rossi. "Speed Index: Relating the Industrial Standard for User Perceived Web Performance to Web QoE". In: *IEEE International Conference on Quality of Multimedia Experience*. 2018 (cit. on p. 91).

[81] Huyen T. T. Tran, Nam Pham Ngoc, Tobias Hoßfeld, Michael Seufert, and Truong Cong Thang. *Cumulative Quality Modeling for HTTP Adaptive Streaming*. 2019 (cit. on pp. 91, 119).

[82] Anna Brunstrom, Tommy Pauly, Theresa Enghardt, Karl-Johan Grinnemo, Tom Jones, Philipp Tiesel, Colin Perkins, and Michael Welzl. *Implementing Interfaces to Transport Services (work in progress)*. Internet Draft draft-ietf-taps-impl-05. IETF, Nov. 2019 (cit. on p. 117).

[83] Robert Kiefer, Erik Nordström, and Michael J Freedman. "From feast to famine: managing mobile network resources across environments and preferences". In: *2014 Conference on Timely Results in Operating Systems (TRIOS 14)*. 2014 (cit. on p. 117).

List of Figures

List of Tables