This paper has been been published at the
Applied Networking Research Workshop 2018 (ANRW '18).

The published version is available at
https://dl.acm.org/citation.cfm?id=3232764.

Metrics for access network selection

Theresa Enghardt
TU Berlin
theresa@inet.tu-berlin.de

Philipp S. Tiesel
TU Berlin
philipp@inet.tu-berlin.de

Anja Feldmann
Max Planck Institute for Informatics
anja@mpi-inf.mpg.de

ABSTRACT

Today, most mobile devices can connect to the Internet via multiple access networks. To make an informed choice of which network to use, a host requires accurate and up to date performance metrics. However, so far such network characteristics are typically not readily available and can be highly volatile.

We explore what performance metrics are available on a host by monitoring and aggregating them within our Socket Intents prototype. These metrics then feed into our access network selection policies to improve page load time in a testbed emulating various network characteristics.

We find that minimum and median Smoothed Round Trip Time and maximum observed download rate are useful metrics in the sense that they enable us to speed up web page load times by up to 65% compared to using a single interface.

CCS CONCEPTS

• **Networks → Network performance analysis**; *Network dynamics*; *Network monitoring*; Wireless local area networks;

KEYWORDS

Access networks, Performance Monitoring, Network Properties, Multi-Access Connectivity

1 INTRODUCTION

In today's Internet, a host often has multiple access networks available to choose from. The configured default is not always the best option, as access networks not only differ in, e.g., latency, bandwidth, or loss, but also across time. Individual performance metrics can be estimated by querying the current state of network interfaces or transport protocol stacks. Yet, these metrics have to be collected and aggregated to get a comprehensive understanding of access network performance, To our knowledge, so far such metrics are not easily available to any host.

In this paper, we explore which performance metrics can be gathered and how to use them for access network selection. Next we show how Socket Intents policies based on these metrics can improve web performance.

The Socket Intents Prototype, first presented in [12], is a client-side software that enables access network selection on a host. An application can use the Socket Intents API to specify what it knows about its upcoming traffic. This information then serves as input to the access network selection policy, along with current performance metrics. Socket Intents is based on the observation that knowing about performance metrics alone is not enough. An access network selection policy also needs to know what to optimize for, as application have different requirements for different kinds of traffic. For instance, when downloading small objects, load time is dominated by Round Trip Time, while when downloading larger objects, bandwidth has a higher impact. Based on both the size of a transfer and the current performance metrics, our access network selection policy chooses the more suitable access network.

We demonstrate the feasibility of our approach in a testbed, in which we emulate a range of typical network characteristics. For these characteristics, we show how accurately our prototype captures the performance metrics, and how our access network selection policies can speed up page load time. Such policies can be made available through novel programming interfaces such as standardized by the Transport Services Working Group [5].

2 BACKGROUND

We briefly discuss existing tools to query performance metrics and systems to handle multiple access networks. Then we present our Socket Intents approach.

2.1 Querying Performance metrics

To understand network performance from a host perspective, there are many command line tools to read network interface counters including ethtool [8]. Furthermore, the iproute2 utilities [7] contain tools such as socketstat, which can query the current status of connections from the kernel. However,

ANRW '18, July 16, 2018, Montreal, QC, Canada

ACM ISBN 978-1-4503-5585-8/18/07…$15.00
https://doi.org/10.1145/3232755.3232764

none of these tools aggregates performance metrics in a central place on the host, so they can serve as input for access network selection. Performance metrics can also be collected within the network [6, 9, 17]. But such metrics are typically not made available to hosts. Furthermore, if they were made available to the host by the ISP, this would introduce the problem of whether the data can be trusted.

2.2 Multiple Access Networks

Most related work on using multiple access networks focuses on WiFi off- or onloading [1, 11], where a network operator controls multiple access networks and can then shift traffic between them. Our work instead empowers the end host to use its own access network selection policy according to performance metrics. Commercial systems have started exploring this [2], but they take fewer metrics into account.

When paths over multiple access networks are combined using MPTCP, performance differences also play a role, as examined in [3, 4, 10]. While an MPTCP scheduler handles performance differences implicitly, we explicitly provide performance metrics to an access network selection policy, which can then use either of the available networks, or combine their capacities using MPTCP.

2.3 Socket Intents Prototype

Our Socket Intents prototype[1], introduced in [12] and presented in detail in [16], provides access network selection to applications that use our Socket Intents API. Applications specify their *Intents*, pieces of information regarding what the application knows about its own communication. Intents can be characteristics, expectations, or preferences. In this paper we focus on the "Size to be Received", which is the number of bytes that the application expects to receive as the result of an outgoing message.

Once an application has specified the Intents for its traffic through the Socket Intents API, the access network selection policy chooses one of the local interfaces and addresses, thus, one of the available access networks, based on metrics and Intents[2]. On the chosen interface, the policy can decide to open a new connection, or, if there are already open connections over this interface, decide to reuse one of them.

2.4 Multi Access Manager

Within the Socket Intents prototype, the component that gathers performance metrics and hosts the access network selection policy is called the Multi Access Manager (MAM)[3]. When started it gathers a list of currently configured network interfaces and their IP addresses. It then starts gathering performance metrics, see Section 3.2, and makes them available to the access network selection policy, see Section 4.

[1]The source code is available at https://github.com/fg-inet/socket-intents

[2]Our policy can also select between multiple endpoints, e.g., multiple IP addresses resolved from the same host name.

[3]The MAM is a daemon running on the host in user space. It is written in C and runs on Linux and Mac OS X.

3 GATHERING PERFORMANCE METRICS

For selecting an appropriate access network for an upcoming communication unit the host needs performance estimates for each interface. Such estimates can be derived from performance metrics gathered on the host at different layers.

We focus on passive measurements, as we want to minimize overhead and not increase congestion on the wireless channel and access link or use up a user's data volume quota on cellular networks. This implies that we only get performance metrics for interfaces we use. Also, the gathered metrics are most useful to estimate performance for paths to destinations that the host communicates with. Furthermore, metrics are influenced by the traffic patterns, e.g., the amount of data and the number of connections.

3.1 Metrics Description

In the following, we list candidate metrics, categorized by the layer on which we can observe them on the host.

Physical Layer Metrics. As the first hop often dominates Round Trip Time (RTT) and bandwidth, and is often wireless, we consider the following metrics relevant for access network selection.

Signal Strength is often used to select the best Access Point or base station. Lower signal strength, relative to the same noise floor, is correlated with higher bit error rates and lower modulation rates. Low signal strength can lead to higher RTTs, but above a certain threshold, other metrics have a higher influence, see [14].

Modulation Rate determines how many bytes are transmitted within a symbol on the wireless channel. A high modulation rate leads to a higher possible bitrate, given sufficient signal strength.

Channel Utilization indicates the percentage of time during which there is a transmission on the wireless medium. A high channel utilization indicates a congested wireless network.

Network and Transport Layer Metrics. As some performance bottlenecks within the first few hops cannot be observed on the physical layer, we also explore higher layer metrics. Some of these metrics may be correlated with physical layer properties or with each other.

Available bandwidth is the number of bytes per second that can be sent or received over a given interface.

Round Trip Time (RTT) is the time from sending a packet to receiving the response. Round Trip Time

is typically observed in the transport protocol stack, e.g., per TCP connection.

Round Trip Time Variation is the disparity of RTT values either over time or among connections. A high RTT variation often indicates congestion [15].

Packet Loss is the percentage of sent packets not received on the other end, e.g., due to being dropped along the path. Packet loss is not directly observable from the end hosts. However, protocols that provide reliable transport, e.g., TCP, SCTP, and QUIC, do keep track of which data was actually received by the other end in order to retransmit lost data. Therefore, this information can be used to infer packet loss.

3.2 Implementation

Our Multi Access Manager (MAM) periodically queries metrics[4], aggregates, and logs them. We choose this approach rather than gathering the metrics in an event-based manner, e.g., using BPF, because it makes performance metrics easier to log and compare. Yet, periodically querying metrics may increase energy consumption, as periodically querying a driver prevents it from going into power save mode. MAM stores performance metrics for a certain time[5], and then times them out, i.e., resets them to their default value[6].

Physical Layer. As physical layer metrics are available from WiFi drivers, MAM queries them using Netlink. Monitoring similar metrics on a cellular interface is more complicated since it is usually not readily available from the driver.

WiFi Signal Strength of the last received data frame.

WiFi Modulation Rate of the most recently sent data frame and the last received frame.

WiFi Channel Utilization as parsed from QBSS information elements in beacon frames[7].

Network Layer. We query this information from the network interface driver.

Maximum download rate: Assuming that the bandwidth bottleneck for a path is within the access network, we approximate available bandwidth by monitoring the maximum observed download rate on the interface. To calculate this rate, we query the interface counters from the network driver, compute the difference between successively read counters, and divide the difference by time since the last reading.

Transport Layer. As these metrics are readily available, we query properties of current TCP connections from the Linux kernel via Netlink.

Smoothed Round Trip Times (SRTTs): If we assume that the highest latency link long the path is in the access network, we can treat the RTT observed on a TCP connection as an estimate for the RTT over this access network. Thus, we aggregate Smoothed RTTs for all connections[8] on the same prefix. As a lower bound, we compute the minimum. As a typical value of what a new connection can expect, we compute the median of the SRTT values. If the highest RTT link is not in the access network, SRTTs may vary across destination, so it may make sense to differentiate per destination.

Round Trip Time Variation: We compute RTT variation in two ways: By calculating the variation of the SRTT values we observe over the same prefix, and by looking at the SRTT variation values for each individual connection and calculating the median.

Packet Loss: Recent TCP implementations keep track of retransmitted packets and packets deemed lost. We query these counters and divide them by the total number of packets sent to approximate upstream packet loss. Note that these loss counters are only estimates, as there is no indication whether retransmits are caused by link errors, congestion, or re-ordering. On the receiver side, TCP implementations do not keep track of lost and re-ordered bytes. If they did, we could only estimate the amount of lost and re-ordered bytes[9], but have no certainty about how many packets were lost.

4 ACCESS NETWORK SELECTION

To take advantage of the above performance metrics our Socket Intents Prototype includes the following access network selection policies to improve web page load time. Each policy can choose among all available interfaces for each communication unit, e.g., each object to be downloaded.

[4]MAM queries metrics every n milliseconds. A callback interval of 100 ms has empirically proven to show good results. Longer query intervals result in lower performance overhead on the system, but also the metrics are less current.

[5]5 minutes works for us.

[6]We set the default values to 0. In this case, metrics will not be taken into account by the access network selection policy.

[7]There is no way to get QBSS from Netlink, so we set up a monitor interface that sniffs on the same channel as the WiFi interface that sends and receives our data. This monitor interface then filters out Beacon frames using BPF and parses the QBSS information elements.

[8]We query current TCP connections that are not in SYN-SENT, SYN-RECV, TIME-WAIT or CLOSE state, as in these states, the SRTT is set to zero.

[9]Distinguishing between lost and re-ordered bytes/packets is only possible in retrospect, i.e., 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 re-transmission of a lost packet.

4.1 Threshold Policy

Our Threshold Policy uses the fact that for smaller objects, latency dominates load time, while for large objects, bandwidth is more crucial. Thus, our Threshold Policy schedules small objects over the network with lower latency and distributes larger objects according to available download capacity.

To determine the threshold of when an object is considered small, the policy first predicts whether latency or bandwidth has a higher influence on the object load, i.e., which of these components account for a larger share in estimated load time. The latency component of the load time is approximated by the minimum SRTT if an existing connection is available for reuse, or twice this value if a new connection has to be set up. For computing the bandwidth component, the policy divides the object size by the estimated bandwidth capacity[10]. If the latency component is greater than the bandwidth component, the Threshold policy considers the object to be small and schedules it over the lower latency network. Otherwise, the policy estimates the load time over all available access networks. For connection reuse, it adds the median SRTT and the bandwidth component. Here, it uses the median SRTT because it assumes a longer object load and, thus, the connection will be affected by congestion and crosstraffic in a similar way as the other connections. For new connections, the policy computes the number of slow start rounds that the object will need until either the available capacity has been reached or the object has been fully retrieved. Finally, it compares the approximated load times and chooses the interface with the shorter predicted load time.

4.2 Threshold Policy With Penalty

Although the Threshold Policy takes other traffic that it has seen into account for its predictions, we find that it does not sufficiently consider external influences, such as crosstraffic. Thus, the Threshold Policy with Penalty additionally penalizes interfaces with high current external interference. After computing the download estimates over both access networks in a similar way as the Threshold Policy, it increases the load time estimate based on the penalty metrics. As WiFi crosstraffic accounts for a lot of external interference, we use WiFi utilization as penalty by increasing the load time estimate on a WiFi interface. This has the advantage that a WiFi with high crosstraffic will be utilized less, but the disadvantage that WiFi utilization cannot be compared with other access network technologies.

Initially, we experimented with SRTT variations to derive penalty, as a high variation of RTTs indicates congestion. However, we found that none of our SRTT variation metrics allows our policy to reliably distinguish between access network with and without crosstraffic. The mean of all SRTT variations currently observed on an interface suffered from higher RTT variation on links with higher base RTT even when the congestion level was similar, thus, underestimating congestion on the lower RTT link. The mean SRTT variations normalized by the minimum SRTT was too similar and gave the higher RTT link an unfair advantage. Therefore, we focus on WiFi utilization.

[10]The capacity is calculated by dividing the recently observed maximum download rate on the access network by the number of active connections on the same network, as seen by the policy.

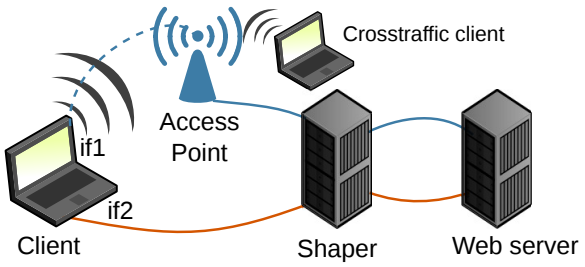


Figure 1: Testbed setup.

4.3 Biased Choice Policy

The above policies have the two limitations. First, load time estimation may not be accurate and, second, statistics may be out of date. Therefore, our Biased Choice Policy distributes objects across access networks in a probabilistic manner. The policy calculates a probability with which to use an access network based on the predicted load time. Lower predicted load times will result in a higher bias, e.g., if the predicted completion time on one of the access networks is 5 times higher than on the other, then this network is 5 times less likely to be chosen. We also combine the Biases Choice Policy with the Penalty Policy by reducing the bias for interfaces with high crosstraffic, i.e., current WiFi utilization.

5 EVALUATION

The objective of our evaluation is twofold: (1) We demonstrate feasibility of metrics-based access network selection and (2) we study the impact of our policies.

5.1 Setup

In our evaluation we want to observe metrics across a wide range of access network characteristics. Therefore, we set up a testbed where we control traffic, crosstraffic, and traffic shaping. As our metrics gathering relies on traffic being present on the network, we use web traffic. To this end, we set up a web server with static web pages containing a number of images of different sizes. Our focus is access network selection from the host, so we set up a notebook as the client, which can reach the web server through multiple access networks via a traffic shaper, see Figure 1.

We want to study actual WiFi characteristics, so the client connects to one of the access networks via a WiFi Access

Table 1: Shaping levels for feasibility study

Property	Levels
WiFi RTT:	20, 50, 100, 200, or 400 ms.
WiFi bandwidth up:	0.1, 0.25, 0.5, 1, 2, 4, or 6 Mibit/s.
WiFi bandwidth down:	0.25, 0.5, 2, 4, 8, 12 or 20 Mibit/s.
WiFi loss:	none, 0.1, 0.25, or 0.5%.
WiFi crosstraffic:	none, 50, 90, 150, or 200 Mbit/s.
Wired RTT:	60, 120, 180, 350, or 700 ms.
Wired bandwidth up:	0.05, 0.3, 0.7, 1.8, 3, or 5 Mibit/s.
Wired bandwidth down:	0.1, 0.5, 1, 2.5, 5, 15, or 20 Mibit/s.

Point[11]. To emulate different uplink characteristics, our traffic shaper imposes latency, rate limiting, and packet loss on traffic between the web server and each of the access networks[12]. Furthermore, we study how crosstraffic on the WiFi influences our metrics. Accordingly, we set up another client that is closer to the AP[13] and that sends crosstraffic in one continuous UDP flow during the entire page load. To generate realistic web traffic, the client runs the Firefox browser instrumented using Selenium. Modern web browsers are highly complex, so we implemented Socket Intents support within a client-side HTTP proxy. Our policies use the "Size to be Received" Intent. To obtain this information, the HTTP proxy first fetches the initial 4KB of each object and, then, reads the Content-Length header to determine its full size. Based on the size and on the observed performance metrics, the policy then chooses the access network on which the proxy should fetch this object. Once all objects of a web page have been retrieved, the browser logs the page load time[14].

5.2 Feasibility study

To demonstrate the feasibility of estimating access network characteristics based on metrics observed on the host, we compare our metrics with the actual shaped values. We use a wide range of typical characteristics of both cellular and WiFi networks[13], see Table 1. As the observed maximum download rate depends on the amount of traffic seen, our workload needs to saturate the capacity of the bottleneck link to get accurate results. To explore what traffic is needed to fill the link, our client fetches 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.

[11]We use 802.11n on channel 36 on the 5 GHz spectrum, where there are no other BSSIDs.

[12]Our traffic shaping is automated using Augmented Traffic Control (ATC), see https://github.com/facebook/augmented-traffic-control. We had to configure ATC to shape rather than police traffic, and in addition we manually set appropriate queue sizes to avoid buffer bloat.

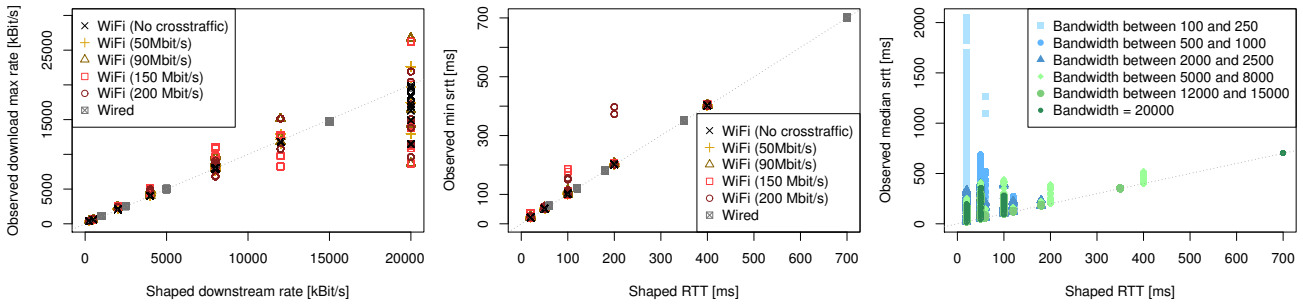
[13]Both AP and the crosstraffic client are located in the same office, while our web client is in a different office across the hall on the same floor.

[14]We record the total duration of a page load according to the HAR file. These page load times include processing and rendering times, but as we compare loads of the same page and we do not use JavaScript, rendering should impose a similar overhead each time.

Figure 2 shows our three main metrics, maximum download rate, minimum SRTT, and median SRTT, compared to the shaper settings with which they were measured. Figure 2a shows the estimated maximum download rates that the MAM observes on the WiFi and wired links during the last page load in a run. Over the WiFi with no crosstraffic and over the wired access network, the estimated maximum download rate corresponds to the shaped download rate in all cases except for 20000 kBit/s. In the latter case, the bottleneck link only gets saturated during the last page load, which is why we see the estimated maximum download rate increase up to the full 20000 kBit/s. With crosstraffic on the WiFi, the estimated maximum download rate 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 download rate 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 download rate than what is shaped on the bottleneck link upstream of the AP. Despite these irregularities, we conclude that the observed maximum download rate gives MAM an idea about the bottleneck download rate in our access network. We can account for outliers by making the maximum download rate more robust, i.e., by only taking values into account that are frequently observed.

Figure 2b shows the minimum Smoothed Round Trip Times (SRTTs) seen by MAM for different shaped RTTs[15] 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 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 2c shows the median of the SRTTs that MAM observes for the current TCP connections for different shaped download bandwidths. 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 bandwidth. However, for 2000 kbit/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 an interface as an estimate of the RTT that a download would experience if it was scheduled on this interface. The observed WiFi metrics are in line with our expectations. As we cannot monitor downstream packet loss, we exclude it from our feasibility study.

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



(a) Observed maximum download rate **(b) Observed minimum SRTT** **(c) Observed median SRTT**

Figure 2: Comparison of metrics observed by MAM against shaped parameters

Table 2: Policy scenarios: Network parameters.

	WiFi			Wired		
	RTT	Down	Up	RTT	Down	Up
	ms	MBit/s	MBit/s	ms	MBit/s	MBit/s
Both good	20	20.0	6.0	60	15.0	5.0
Both bad	200	2.0	0.5	350	1.0	0.7
Asymmetric	50	2.0	0.5	350	15.0	5.0

5.3 Load time speedups

Given that our metrics provide reasonable performance estimates, we explore whether using them as input to our access network selection policies can speed up page load time. To do so, we fetch the same page multiple times, first over each access network individually and, then, using the policies presented in Section 4.

We emulate three scenarios, see Table 2: Both access network have "good" performance, to check whether we can still get a performance benefit, both access network have "bad" performance, to see whether access network selection can help in challenging environments, and an asymmetric case in which neither network is "good" nor "bad", but one provides a high bandwidth while the other has low latency. Our client downloads static web pages with objects of different sizes.

In Figure 3 we can see the median page load times of three pages. Using only the wired interface results in the highest load times except in the asymmetric case for the two larger workloads, since in these cases the higher bandwidth over the wired interface is more useful than the lower RTT over the WiFi. Using the Threshold Policy, which distributes traffic across available access networks, improves load times by around 30% in most cases, e.g., with asymmetric network characteristics or a large web page. Adding a penalty to the highly utilized WiFi further reduces load time, with speedups of up to 65% for the 32 objects of 100 KB in the scenario where both access network have good performance. In other cases, the Threshold Policy does not speed up page load time, but chooses the better of the two available access networks. On the other hand, the Biased Choice Policy does not always improve page load times, it even sometimes performs worse than the better of the two interfaces, even when penalizing

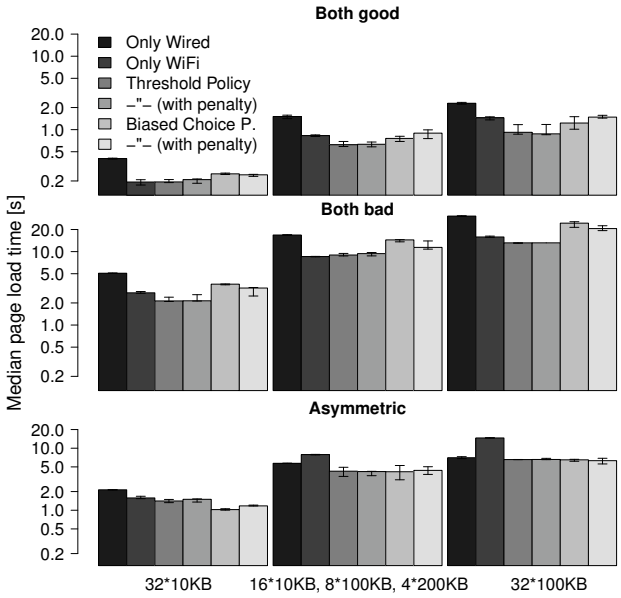


Figure 3: Page load times.

a highly utilized WiFi. With crosstraffic, page load times over WiFi increase, but the Threshold Policy still results in a speedup compared to the better of the two interfaces. Consequently, we see that speedups of up to 65% are possible by combining two access networks in a smart way.

6 CONCLUSION

Our Socket Intents prototype gathers performance metrics for its access network selection policies. We can accurately capture maximum download rate, minimum and median SRTT, and WiFi utilization in our testbed. We take advantage of this knowledge in our Threshold Policy with penalty, which can speed up page load by up to 65%.

In the future we will gather metrics with more realistic crosstraffic and over more access technologies, such as cellular networks. We also want to compare our access network selection policies with MPTCP and use more realistic workload, i.e., actual web pages, or other applications such as video streaming.

ACKNOWLEDGMENTS

This work has been supported by Leibniz Prize project funds of DFG - German Research Foundation: Gottfried Wilhelm Leibniz-Preis 2011 (FKZ FE 570/4-1).

Thanks to Puneeth Nanjundaswamy for contributing code to the initial version of the download rate monitoring, to Sören Becker for contributing the code of the initial version of the RTT monitoring, and to Marcin Bosk for contributing the code of the initial version of the WiFi utilization monitoring.

REFERENCES

[1] Adnan Aijaz, Hamid Aghvami, and Mojdeh Amani. 2013. A survey on mobile data offloading: technical and business perspectives. *IEEE Transactions on Wireless Communications* 20, 2 (2013), 104–112.

[2] Apple Inc. 2016. About Wi-Fi Assist. https://support.apple.com/en-us/HT205296

[3] Yung-Chih Chen, Yeon-sup Lim, Richard J Gibbens, Erich M Nahum, Ramin Khalili, and Don Towsley. 2013. A measurement-based study of multipath tcp performance over wireless networks. In *ACM IMC*. ACM, 455–468.

[4] Shuo Deng, Ravi Netravali, Anirudh Sivaraman, and Hari Balakrishnan. 2014. Wifi, lte, or both?: Measuring multi-homed wireless internet performance. In *ACM IMC*. ACM, 181–194.

[5] IETF TAPS Working Group. 2018. Transport Services (TAPS) Charter. https://datatracker.ietf.org/group/taps/charter/, accessed in June 2018.

[6] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. 2014. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Comm. Surveys and Tutorials* 16, 4 (2014), 2037–2064.

[7] Alexey Kuznetsov and Stephan Hemminger. 2018. iproute2 utilities for controlling TCP/IP networking and traffic in Linux. https://wiki.linuxfoundation.org/networking/iproute2, accessed in April 2018.

[8] David Miller. 2018. ethtool - utility for controlling network drivers and hardware. http://www.kernel.org/pub/software/network/ethtool/, accessed in April 2018.

[9] Kevin Phemius and Mathieu Bouet. 2013. Monitoring latency with openflow. In *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 122–125.

[10] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. 2012. How hard can it be? designing and implementing a deployable multipath TCP. In *USENIX NSDI*. USENIX Association, 29–29.

[11] Claudio Rossi, Narseo Vallina-Rodriguez, Vijay Erramilli, Yan Grunenberger, Laszlo Gyarmati, Nikolaos Laoutaris, Rade Stanojevic, Konstantina Papagiannaki, and Pablo Rodriguez. 2013. 3GOL: Power-boosting ADSL using 3G OnLoading. In *ACM CoNEXT*. ACM, 187–198.

[12] Philipp S. Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. 2013. Socket Intents: Leveraging Application Awareness for Multi-access Connectivity. In *ACM CoNEXT*. ACM, New York, NY, USA, 295–300. https://doi.org/10.1145/2535372.2535405

[13] Joel Sommers and Paul Barford. 2012. Cell vs. WiFi: On the Performance of Metro Area Mobile Connections. In *ACM IMC*. ACM.

[14] Kaixin Sui, Mengyu Zhou, Dapeng Liu, Minghua Ma, Dan Pei, Youjian Zhao, Zimu Li, and Thomas Moscibroda. 2016. Characterizing and improving wifi latency in large-scale operational networks. In *ACM MobiSys*. ACM, 347–360.

[15] Srikanth Sundaresan, Nick Feamster, and Renata Teixeira. 2016. Home network or access link? locating last-mile downstream throughput bottlenecks. In *PAM*. Springer, 111–123.

[16] Philipp S. Tiesel, Theresa Enghardt, Mirko Palmer, and Anja Feldmann. 2018. Socket Intents: OS Support for Using Multiple Access Networks and its Benefits for Web Browsing. arXiv:arXiv:1804.08484

[17] Niels LM Van Adrichem, Christian Doerr, and Fernando A Kuipers. 2014. Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 1–8.