<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper presented at *6th IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web based Service Architectures (PEDISWESA)*.

Citation for the original published paper:

Grinnemo, K-J., Brunström, A. (2015)
A First Study on Using MPTCP to Reduce Latency for Cloud Based Mobile Applications.
In: *6th IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web based Service Architectures (PEDISWESA)* IEEE Computer Society

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-36001>

A First Study on Using MPTCP to Reduce Latency for Cloud Based Mobile Applications

Karl-Johan Grinnemo and Anna Brunstrom

Department of Computer Science

Karlstad University, Karlstad, SWEDEN

Email: {karlgrin,annab}@kau.se

Abstract—Currently, Multipath TCP (MPTCP) – a modification to standard TCP that enables the concurrent use of several network paths in a single TCP connection – is being standardized by IETF. This paper provides a comprehensive evaluation of the use of MPTCP to reduce latency and thus improve the quality of experience or QoE for cloud-based applications. In particular, the paper considers the possible reductions in latency that could be obtained by using MPTCP and multiple network paths between a cloud service and a mobile end user. To obtain an appreciation of the expected latency performance for different types of cloud traffic, three applications are studied, Netflix, Google Maps, and Google Docs, representing typical applications generating high-, mid-, and low-intensity traffic. The results suggest that MPTCP could provide significant latency reductions for cloud applications, especially for applications such as Netflix and Google Maps. Moreover, the results suggest that MPTCP offers a reduced latency despite a few percent packet loss, and in spite of limited differences in the round-trip times of the network paths in an MPTCP connection. Still, larger differences in the round-trip times seem to significantly increase the application latency, especially for Netflix, Google Maps, and similar applications. Thus, to become an even better alternative to these applications, this paper suggests that the MPTCP packet scheduling policy should be changed: Apart from the round-trip times of the network paths in a connection, it should also consider the difference in round-trip time between the network paths.

I. INTRODUCTION

The growth of mobile usage and cloud services has been tremendous: 40 percent of the world's smartphone users access Internet and apps even before getting out of bed [1]. And, once out of bed, Internet and apps are used almost constantly. As content delivery over cellular networks is rapidly growing, it is becoming increasingly important to secure quality of service from the cloud service provider to the end user.

Coverage and speed are the biggest drivers of service satisfaction, and currently large sums of money are invested into the expansion of cellular networks and the deployment of LTE and 4G. Still, one cost-effective way of improving the mobile end-user experience has largely remained untapped – multihoming: Mobile devices such as smartphones and tablets are often equipped with both WiFi and 3G/4G interfaces, but they rarely use more than a single interface at a time. One of the reasons multihoming has earned so little attention is that existing network-layer solutions have been considered too immature. Another reason is that up to now there has in practice only been one transport protocol available that supports multihoming, the Stream Control Transmission Pro-

ocol (SCTP) [2]: a transport protocol that in large part has proven itself incompatible with existing middlebox solutions, uses socket API extensions that are not easily incorporated in existing TCP applications, and whose support for simultaneous use of several network paths has yet to become standardized.

Currently, a set of extensions to standard TCP, Multipath TCP (MPTCP) [3], is being standardized in the Internet Engineering Task Force (IETF) that address the deficiencies of SCTP, and make it possible for existing TCP applications to run their traffic over several simultaneous network paths, with less worries about firewalls, NATs and other middleboxes. Recent works by Raiciu et al. [4], Paasch et al. [5], and others have evaluated MPTCP multihoming over combinations of WiFi and 3G/4G links, and their results are indeed promising. Still, their works have largely concerned the throughput characteristics of MPTCP, and to a lesser extent the latency characteristics, an often more important factor for the quality of experience (QoE) of several cloud-based mobile applications.

The main contribution of this paper is that it provides a fairly comprehensive evaluation of the expected latency performance of typical cloud applications when accessed from a multihomed mobile device that employs MPTCP. The selected applications are Netflix, Google Maps and Google Docs, and were purposely selected to represent a high-, a mid-, and a low-intensity application. Apart from the impact of traffic and its characteristics on the latency performance, the effect of differences in delay and/or packet loss between network paths in an MPTCP connection are considered. The paper suggests that MPTCP could provide a reduced latency for high- and mid-intensity applications such as Netflix and Google Maps, despite a few percent packet loss, and despite some differences in the round-trip time (RTT) of the network paths in a connection. However, the paper also suggests that MPTCP provides little or no latency reduction for low-intensity traffic such as Google Docs, and that its latency performance significantly degrades with larger differences in the RTTs of the network paths.

The remainder of the paper is organized as follows. A brief overview of MPTCP is provided in Section II. The experiment setup and methodology are described in Section III, and the outcome of our experiment is presented and discussed in Sections IV and V. The paper concludes in Section VI.

Fig. 1. Experiment setup.

II. MPTCP OVERVIEW

As previously mentioned, MPTCP is a number of extensions to standard TCP that enable multipath connections, i.e., connections which provides for concurrent transmission over several network paths. It has been standardized by IETF in RFC 6824 [3], and a de facto reference implementation of MPTCP has been developed for the Linux kernel [6]. Apart from that, there are at least three known implementations of MPTCP of which the one used by Apple’s personal digital assistant service, Siri, is the most widespread and well-known.

An MPTCP connection is established by using the same three-way handshake as standard TCP does, but with an extra option in the SYN segment that indicates its support for MPTCP. The three-way handshake creates the first so-called TCP subflow over one interface. To use an additional interface, MPTCP carries out yet another three-way handshake procedure on that particular interface.

If a connection comprises several subflows, it is the MPTCP packet scheduler that selects on which subflow network path a segment should be sent, and, considering properties of the selected network path, which segment to send. The default policy employed by the packet scheduler in Linux MPTCP is to select network path on the basis of the shortest RTT, shortest-RTT-first: segments are first sent on the network path with the lowest smoothed RTT estimator (SRTT). Only when the congestion window of this network path has become filled, segments are sent over the network path with the next lowest SRTT.

MPTCP uses a congestion control algorithm that couples the standard TCP congestion control algorithms of different subflows together: the congestion on each subflow is measured, and traffic are reallocated from the most congested subflows to the less congested ones. The original coupled congestion control employed by MPTCP is the Linked Increases Algorithm (LIA). However, since LIA fails to fully balance the congestion among the subflows in an MPTCP connection, other coupled congestion control algorithms have been proposed, e.g., the Opportunistic Linked Increases Algorithm (OLIA), which rectifies the load balancing deficiency of LIA, and thus provides an improved congestion balancing.

III. EXPERIMENT SETUP

In our experiment, we aimed to model a cloud-to-mobile end-user session from a latency perspective. The experiment

setup is illustrated in Figure 1. The Server modeled a node in a service cloud which hosted cloud applications accessed by an end user, the Client. Both the Client and Server hosts ran native Ubuntu 13.10 (64 bit) with Multipath TCP version 0.88.7 compiled into the Linux kernel.

The two available network access paths, P1 and P2, were modeled by two FreeBSD 10.0 hosts, Dnet1 and Dnet2, which both ran the Dummynet network emulator [7]. The Dnet1 and Dnet2 hosts emulated 100-Mbps network paths, and were configured with 100-slot queues.

The bandwidth of the network paths was selected to model the practically available bandwidth in near-term 4G networks – currently operators such as Telia [8] in Sweden, Verizon [9] in the U.S., and many other mobile operators worldwide offer up to 80 Mbps downlink bandwidth, thus even a conservative prediction would suggest a 100 Mbps downlink speed within the next few years.

To obtain an appreciation of how the network path bandwidths impact the latency performance, some of the tests were also carried out with 10-Mbps emulated network paths. However, the results from these tests are in most cases very similar to the ones obtained with 100-Mbps network paths, and only mentioned in those cases they substantially differ from the 100-Mbps tests.

The RTTs emulated by Dnet1 and Dnet2 were chosen in the range of 10 - 400 ms. They were chosen to cover the feasible range of RTTs experienced in 4G networks. Particularly, a lower bound on the RTT of 10 ms was selected with the rationale of being a strict lower bound on the user-plane latency between the user equipment and the access gateway in an LTE network [10], and an upper bound of 400 ms was selected on the basis of being a liberal upper bound for interactive Web access.

As mentioned, traffic from three typical cloud applications, Netflix, Google Maps, and Google Docs, were studied. They were chosen to represent high-, mid-, and low-intensity cloud applications. We used the D-ITG [11] traffic generator to emulate traffic from these applications. To obtain relevant traffic profiles for D-ITG, we used Wireshark and captured TCP traffic from repeated real sessions on a Google Chrome Web browser. Since the captured TCP traffic to a large degree depended on the network conditions, e.g., available bandwidth and RTT, at the time of the capture, we could not directly create the D-ITG traffic profiles from the captured TCP packets. Instead, we used the TCP captures to reconstruct the application traffic, and created the traffic profiles on the basis of this traffic. Reconstructing the Netflix traffic was reasonably straightforward since that traffic was transmitted in clear over HTTP, however, the Google Maps and Google Docs traffic was sent encrypted (HTTPS), and thus the actual application traffic could only be approximated.

Figure 2 shows representative traffic time series plots from the three studied cloud applications. The Netflix traffic used in our experiment was captured while watching random parts from the first season of the series “24”. The capture took place some time after the initial buffering phase, and Figure 2(a)

Fig. 2. Traffic time series for studied cloud applications.

TABLE I
LINUX KERNEL SETTINGS.

Kernel Parameter	TCP	MPTCP
net.core.wmem_max		104857600
net.core.rmem_max		104857600
net.ipv4.tcp_congestion_control	cubic	coupled
net.ipv4.tcp_wmem	4096 87380 104857600	
net.ipv4.tcp_rmem	4096 87380 104857600	
net.mptcp.mptcp_enabled	0	1
net.mptcp.mptcp_debug	N/A	0
net.mptcp.mptcp_checksum	N/A	1
net.mptcp.mptcp_path_manager	N/A	fullmesh
net.mptcp.mptcp_syn_retries	N/A	3

illustrates how the Netflix traffic was sent in ON-OFF cycles with block sizes roughly between 500 KiB and 1.5 MiB, a behavior that fairly well corroborates with previous traffic studies [12]. The Google Maps traffic was captured while a person was visualizing routes between arbitrary destinations. As shown in Figure 2(b), the traffic comprised of non-periodic, and relatively large, traffic spikes. In fact, some times the traffic spikes reached levels on par with the Netflix traffic. Lastly, the Google Docs traffic was captured while a person was working on a text. As can be observed from Figure 2(c), the traffic bursts hover around 400 Bytes with some spikes of more than twice that size, a behavior that aligns well with the traffic pattern study of Augustin et al. [13].

In all tests, the latency a packet experienced from being sent by the ITGSend application on the Server until it was received by the ITGRecv application on the Client, i.e., the application-level packet latency, was measured. Each test lasted for three minutes and was repeated 30 times. The mean of the average packet latency in all test runs was used as metric for the latency performance.

In order to evaluate MPTCP, each test was run with both MPTCP and the default Linux TCP, i.e., CUBIC TCP. In the tests with the default Linux TCP, traffic was only sent on P1, while MPTCP utilized both paths.

MPTCP was configured with coupled congestion control (LIA), the MPTCP checksum enabled, and the full mesh path manager. The send and receive buffers were configured to

100 MiBytes, i.e., well in line with the recommended settings of RFC 6182 [14], and thus appropriately sized to prevent transmissions to be stalled due to buffer shortage. The relevant TCP and MPTCP kernel settings are listed in Table I.

IV. LATENCY OVER LOSS-FREE PATHS

As a first step, we evaluated the latency experienced during loss-free conditions, i.e., when neither P1 nor P2 had any explicit packet losses. The bar graphs in Figure 3 show the mean of the average packet latencies in the symmetrical-path tests, i.e., the tests with the same RTTs on both network paths; the error bars in the graphs illustrate the 95% confidence intervals. We observe a substantial reduction in packet latency with MPTCP for Netflix traffic. In absolute terms, the reduction increased with increased RTT, however, the relative reduction was fairly constant: It was more than 40% for all considered RTTs, and, it was around 50% for several RTTs. We also observe a reduction in packet latency with MPTCP for Google Maps traffic. Again, the absolute reduction increased with increasing RTT while there was no correlation between the relative reduction and the RTT. Instead, the relative reduction varied between 20% and 30%. As regards the Google Docs traffic, we did not observe any significant difference in packet latency with MPTCP. The reason to this was that the Google Docs traffic comprised sparsely generated packets, and was unable to utilize the extra network path provided by MPTCP.

The results from the asymmetrical-path tests, i.e., the tests in which P1 had a fixed RTT of 10 ms while P2 had an RTT that was larger than the one of P1, are shown in Figure 4. As follows, in the tests with an RTT on P2 of 20 ms or less, the path asymmetry had a minor impact on the MPTCP latency performance. However, a much longer RTT on P2 than on P1, significantly deteriorated the latency performance of MPTCP for Netflix and Google Maps traffic: In the Netflix tests, MPTCP went from providing a latency performance on par with standard TCP, when the RTT on P2 was 40 ms, to an increase of packet latency of more than 200%, when the RTT on P2 was 400 ms. The drop in latency performance was even worse for the Google Maps traffic. In these tests, MPTCP went from offering roughly the same latency as standard TCP, when

Fig. 3. Packet latency vs. RTT in loss-free, symmetrical-path tests, bandwidth: 100 Mbps.

Fig. 4. Packet latency vs. RTT in loss-free, asymmetrical-path tests, bandwidth: 100 Mbps.

Fig. 5. Packet latency vs. RTT in loss-free, asymmetrical-path tests, bandwidth: 10 Mbps.

the RTT on P2 was 40 ms, to an increase of packet latency of more than 800%, when the RTT on P2 was 400 ms. The result was an effect of the packet scheduler policy employed by MPTCP, shortest-RTT-first: In the Google Docs tests, almost all traffic could be scheduled for transmission over the path with the shortest RTT, P1, however, in the Netflix and Google Maps tests, more traffic was generated per transmission round than could be permitted transmission by the P1 congestion window. Thus, traffic had to be scheduled over P2 despite its much longer RTT.

As seen in Figure 5, similar results were also obtained in the corresponding tests with 10 Mbps network paths. Of course, the larger queueing delays in these tests resulted in longer packet latencies for both standard TCP and MPTCP. Also, since the queueing delays constituted a larger part of the experienced packet latency, the impact of path asymmetry for MPTCP was smaller in these tests than in the 100-Mbps tests.

(a) $RTT_1 = 10$ ms, $RTT_2 = 10$ ms.

(b) $RTT_1 = 40$ ms, $RTT_2 = 40$ ms.

(c) $RTT_1 = 100$ ms, $RTT_2 = 100$ ms.

Fig. 6. Packet latency vs. packet-loss rate in Netflix tests with the same RTT on both paths.

(a) $RTT_1 = 10$ ms, $RTT_2 = 10$ ms.

(b) $RTT_1 = 40$ ms, $RTT_2 = 40$ ms.

(c) $RTT_1 = 100$ ms, $RTT_2 = 100$ ms.

Fig. 7. Packet latency vs. packet-loss rate in Google Maps tests with the same RTT on both paths.

(a) $RTT_1 = 10$ ms, $RTT_2 = 10$ ms.

(b) $RTT_1 = 40$ ms, $RTT_2 = 40$ ms.

(c) $RTT_1 = 100$ ms, $RTT_2 = 100$ ms.

Fig. 8. Packet latency vs. packet-loss rate in Google Docs tests with the same RTT on both paths.

V. LATENCY OVER LOSSY PATHS

The results from the tests with explicit packet losses on either or both of P1 and P2 are shown in Figures 6, 7, 8, and 9. Notice that contrary to the loss-free tests, there was a fairly large variability between the repetitions in these tests. Although, the repetitions within a single test had the same packet-loss rate, the packet latency was very much dependent on the actual packets being lost.

Let us consider the tests with the same RTTs on both paths. The bar graphs in Figures 6, 7 and 8 summarize the results

from these tests. As could be expected, when there were no packet losses on P2, the gain in reduced packet latency of using MPTCP increased with increasing packet-loss rate on P1 for all three considered traffic types. However, we also observe that Netflix traffic saw substantial reductions in packet latency with MPTCP in the tests with only packet loss on P2, even when the packet-loss rate was as high as 4%. The reason to this was a combination of us using a large receive buffer, and that only a small part of the traffic was sent on P2. In other words, the actual number of packet losses was small, and the

Fig. 9. Packet latency vs. packet-loss rate in tests with longer RTTs on P2 than on P1.

reordering caused by these packet losses could be managed on the Client side. As regards the Google Maps and Google Docs traffic, they experienced more or less the same packet latency with MPTCP as with standard TCP in the tests with only packet loss on P2.

In the tests with packet loss on both paths, MPTCP several times offered significant reductions in packet latency for Netflix as well as for Google Maps traffic, especially in the tests with a packet-loss rate of 4% on P1. For example, MPTCP provided a relative reduction in packet latency of around 25% for Netflix in the test with a 40 ms RTT and a 4% packet-loss rate on both paths. In the corresponding test for Google Maps traffic, the relative reduction was close to 35%.

Next, let us consider the tests with a longer RTT on P2 than on P1. The results from these tests are shown in Figure 9. As follows, in the same way as in the corresponding tests over loss-free paths (cf. Figure 4), a path asymmetry deteriorated the MPTCP latency performance. In fact, a longer RTT on P2 had a larger impact on the MPTCP latency performance than the studied packet-loss rates. MPTCP only provided a reduced latency in the Netflix and Google Maps tests with a 4% packet-loss rate on P1 and no packet loss on P2. In many cases, the latency experienced with MPTCP was significantly larger than with standard TCP – in a few cases more than 100% larger. The reason to this was that the momentary bandwidth requirement during some periods was larger than what was permitted to be sent by the congestion window on P1. As a result, a fairly large part of the traffic was sent on P2, in spite of its much longer RTT. Since packet losses result in a smaller congestion window, this behavior became even more pronounced in the tests with several percent packet loss on P1.

VI. CONCLUSION

In this paper, we study whether or not MPTCP offers any improvement in terms of latency as compared with standard TCP for cloud-based mobile applications. Our study suggests that MPTCP many times do provide significant reductions in latency, especially for more intense traffic such as the one generated by Netflix, but also for less intense Google

Maps traffic. Moreover, it seems that MPTCP continues to provide reduced latency in spite of a few percent packet loss, and in spite of limited differences in the RTT between the network paths. Still, it is evident from our study that the packet scheduling policy employed by MPTCP, shortest-RTT-first, has difficulties in efficiently scheduling packets on network paths whose RTTs differ greatly. As a result, it often fails to efficiently manage high- and mid-intensity traffic, such as the traffic generated by Netflix and Google Maps. In our future work, we intend to work on a packet scheduling policy that address this limitation, and thus would make MPTCP an even better choice for cloud-based mobile applications.

REFERENCES

- [1] “Traffic and Market Report – On the Pulse of the Networked Society,” Market Report, June 2012.
- [2] R. Stewart, “Stream Control Transmission Protocol,” IETF, RFC 4960, September 2007.
- [3] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, “TCP Extensions for Multipath Operation with Multiple Addresses,” IETF, RFC 6824, January 2013.
- [4] C. Raiciu, D. Niculescu, M. Bagnulo, and M. Handley, “Opportunistic Mobility with Multipath TCP,” in *MobiArch*, 2011, pp. 7–12.
- [5] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, “Exploring Mobile/WiFi Handover with Multipath TCP,” in *ACM SIGCOMM Workshop on Cellular Networks*, 2012, pp. 31–36.
- [6] C. Paasch and S. Barre. Multipath TCP in the Linux Kernel. <http://www.multipath-tcp.org>.
- [7] M. Carbone and L. Rizzo, “Dumynet Revisited,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, April 2010.
- [8] (2014, September) Telia 4G. <http://www.telia.se/privat/4g>.
- [9] “The Verizon Wireless 4G LTE Network: Transforming Business with Next-Generation Technology,” White Paper, 2012.
- [10] T. Blajic, D. Nogulic, and M. Druzijanic, “Latency Improvements in 3G Long Term Evolution,” in *MIPRO*, Opatija, Croatia, May 2006.
- [11] A. Botta, A. Dainotti, and A. Pescapè, “A Tool for the Generation of Realistic Network Workload for Emerging Networking Scenarios,” *Computer Networks*, vol. 56, pp. 3531–3547, 2012.
- [12] A. Rao, A. Legout, Y.-S. Lim, D. Towsley, C. Barakat, and W. Dabbous, “Network characteristics of video streaming traffic,” in *CoNEXT*, Tokyo, Japan, December 2011.
- [13] B. Augustin and A. Mellouk, “On Traffic Patterns of HTTP Applications,” in *GLOBECOM*. Houston, Texas, U.S.A.: IEEE, December 2011.
- [14] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, “Architectural Guidelines for Multipath TCP Development,” IETF, RFC 6182, March 2011.