Multipath Transport for Virtual Private Networks

Daniel Lukaszewski
Department of Computer Science
Naval Postgraduate School

Geoffrey G. Xie
Department of Computer Science
Naval Postgraduate School

Abstract

An important class of virtual private networks (VPNs) builds secure tunnels at the transport layer leveraging TCP or UDP. Multipath TCP (MPTCP), an ongoing IETF effort that has been adopted into Linux and iOS, extends TCP to allow data to be delivered over multiple network interfaces and paths simultaneously. In this paper, using a testbed that can emulate a range of path characteristics between the VPN end points, we first empirically quantify the potential of using MPTCP tunnels to increase the goodput of VPN communications when multiple data paths are available. We further design and implement a preliminary version of Multipath UDP (MPUDP) to address the adverse effect of the duplicated congestion control actions that is known with a TCP-in-TCP tunnel. We observe that a severe asymmetry of path delays may cause an excessive amount of packet reordering at the receiving end and consequently degrade the overall performance of TCP-in-MPUDP tunnels. Moreover, we find that a packet scheduler capable of tracking path delays and allocating more packets to path(s) with shorter delay(s) to be an effective and relatively lightweight solution for MPUDP, instead of an elaborate data sequencing mechanism like the one used by MPTCP.

1 Introduction

Virtual Private Networks (VPNs) are important to enterprises and government agencies as they provide secure data communication at reasonable costs by leveraging public or third-party network infrastructures. An important class of VPNs builds secure tunnels at the transport layer leveraging TCP or UDP. Several widely deployed open source software systems such as OpenVPN [4] are of this kind.

As computer hosts and mobile devices are increasingly multi-homed, the Internet Engineering Task Force (IETF) recently proposed a multipath TCP (MPTCP) extension (i.e. RFC 6824) to leverage the additional data paths to increase throughput and support seamless mobility. Essentially, MPTCP enables one or more TCP connections (called subflows) to be added to a primary TCP connection between two MPTCP capable end hosts. Each subflow is added using the standard three-way handshake along with a new TCP header option that has been created to facilitate the creation of subflows as well as identification of such packets at the receiving end.

In addition, MPTCP performs four major functions. First, it incorporates a data sequence signal (DSS) mechanism to map the sequence numbers of individual subflows into an overall master sequence number space at the receiving end. Second, it employs a path manager to identify available paths (i.e. socket pairs with distinct IP address pairs or source ports) for creating new subflows. Third, it runs a packet scheduler to distribute packets among the active subflows. The default scheduler sends more packets on paths with shorter round trip times (RTTs) [12]. Lastly, it introduces several new congestion control algorithms to ensure that multiple subflows of a same MPTCP connection will not be too aggressive (i.e. take an unfair share of bandwidth) when sharing a bottleneck link with regular TCP flows.

Several MPTCP implementations have been created [11, 16]; the most frequently deployed being those created for the Linux kernel and Apple iOS. Given this availability, we have built an emulation testbed to evaluate the potential of using multipath VPN tunnels to increase the performance of VPN communications when multiple data paths are available from the VPN client to the VPN server. The evaluation focused on utilizing TCP services (e.g. file downloads and database transactions) over VPN tunnels as these are the typical tasks performed by a VPN user.

There is a well-known TCP-in-TCP effect [14], where the application throughput may be unnecessarily degraded due to duplicated congestion control actions of

the two TCP connections involved. Therefore, in addition to an evaluation of MPTCP, this paper describes our design and evaluation of a MPUDP prototype. More specifically, our contributions are as follows:

1. We conducted an empirical performance evaluation of MPTCP *specific to VPN scenarios*. The results show that MPTCP can leverage an additional data path between the VPN end points to increase the application performance for a wide range of packet loss and propagation delay configurations of the two paths.

2. We designed and implemented basic MPUDP functionality into the Linux transport layer and evaluated its performance using the same VPN scenarios. The results show that by using a packet scheduler that allocates more packets to the path with the shorter RTT, the relatively lightweight MPUDP can outperform MPTCP even when the paths have a high degree of asymmetry in terms of their propagation delays.

The rest of the paper is organized as follows. Section 2 discusses related work and we present the evaluation of MPTCP tunnels in Section 3. The design and evaluation of MPUDP is described in Section 4, followed by a short discussion of limitations and future work in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

There is a large collection of empirical studies (e.g. [7, 11, 16]) done by the MPTCP designers and other researchers to understand and improve the performance of MPTCP. Specific to VPN scenarios, Boccassi *et al*. developed a system called Binder [1], which uses two OpenVPN gateways and an MPTCP tunnel between them to aggregate the throughput of multiple data paths within the core of a network. Our evaluation of potential VPN performance boosts from MPTCP is complementary to these prior studies. We additionally conduct experiments with a bursty packet loss model [13] to understand the relative performance of MPTCP under more stressful conditions where network congestion may cause consecutive packet losses to a user connection.

In comparison, there is little work on MPUDP we can find from the open literature. The most relevant study known to us is a system called Multipath Mobile Shell (MOSH) [3], which allows two communicating end hosts to set up multiple UDP connections between them. Due to its signaling with application specific messages, the system is not a general transport layer solution like the one presented in this paper.

Additionally, there are several studies covering bandwidth aggregation techniques at the transport layer for mobile hosts (e.g. [6, 10]). Unlike the approach of MPTCP, these solutions require modifications of applications. However, it should be noted that some of the path monitoring mechanisms they employ may be useful for further development of MPUDP.

Finally, the adverse effects of using a TCP over TCP connection is well documented. Olaf Titz [14] in the early 2000's explained how the outer TCP layer, the tunnel, may have a shorter retransmission timeout (RTO) value than the inner TCP layer, which may trigger retransmissions earlier than necessary and ultimately cause the connection throughput to degrade significantly. Our work provides evidence that MPTCP tunnels may suffer from similar effects.

3 Evaluation of MPTCP Tunnels

We first ran a series of experiments to validate the reported performance benefits of MPTCP, and more importantly, to obtain baseline data for comparison with MPUDP. To achieve a comprehensive and reproducible performance evaluation, we decided to use an emulation testbed, with which we can test production VPN and MPTCP software implementations that generate real packets and at the same time, have fine-grain control over the performance characteristics of data paths.

3.1 Testbed Setup

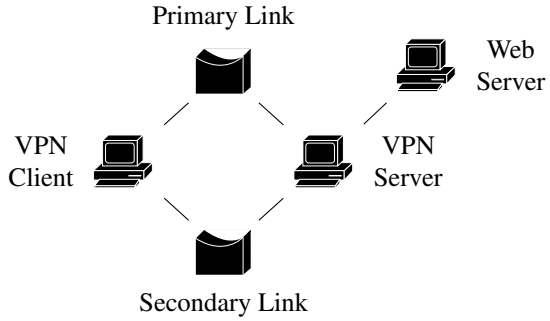


Figure 1: Multipath emulation testbed.

As illustrated in Figure 1, the testbed consists of five nodes, which are 1.86GHz dual-core desktop computers with 4GB of RAM and running the Ubuntu 14.04 version of Linux. Only the tunnel end points, i.e. the "VPN Client" and "VPN Server" nodes, are MPTCP enabled with the version 0.90 of the MPTCP software distribution, and additionally, these nodes are installed with the OpenVPN 2.3.12 software. The "Web Server" node acts as an enterprise intranet resource for the remote VPN

client and is configured to run the Apache2 web server software. 10-Mbps Ethernet links are used for physical node to node connections between the VPN end points to mimic the typical range of WiFi and cellular data rates, while a Gigabit Ethernet link is used to connect the "VPN Server" and "Web Server" nodes.

3.1.1 Path Configurations

A wide range of performance characteristics of the primary and secondary data paths, in terms of the round trip time (RTT) of signal propagation and packet loss rate, can be precisely controlled by running the Linux traffic control command `tc` [8] with appropriate parameters on the "Primary Link" and "Secondary Link" nodes. Specifically, the following configurations are used for the experiments reported in this paper.

Packet loss rate: We used two random models to simulate packet losses on the data paths: (i) uniform per-packet loss probability, which has been extensively used in prior work, and (ii) Gilbert-Elliot (GE) packet loss model [13], which we believe can capture more accurately the likely occurrences of bursty packet loss events on the Internet due to router buffer overflows. The GE model uses two states: a Gap mode and a Burst mode, and it is customizable with four parameters (p,r,k,h) as follows. The model starts in the Gap mode and will shift between the Gap and Burst mode with probability p and r, respectively, after processing each packet. It will drop a packet with probability $1-k$ and $1-h$ while in the Gap and Burst mode, respectively, as illustrated in Figure 2. We have estimated the overall packet loss rates for various (p,r,k,h) combinations by collecting 100,000 `ping` statistics over ten 10,000 `ping` intervals for each combination. From the results, we decided to fix three of the parameters as illustrated in Figure 2 and vary only $(1-k)$ to control the overall loss rate for the GE model while ensuring that most of the loss bursts ($> 90\%$) consist of no more than two packets as one would expect for one TCP or MPTCP flow during typical network congestion. Table 1 shows the set of $(1-k)$ values used in our GE model experiments and their corresponding observed overall loss rates.

Table 1: GE Model Configuration

$1-k$	**Observed Packet Loss Rate**
0%	1%
0.5%	2%
1%	3%
3%	7%

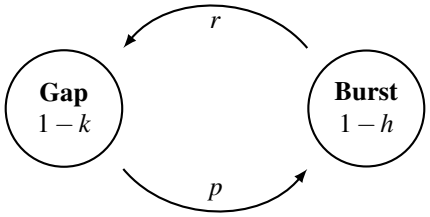


Figure 2: Gilbert-Elliot model in action, with three of the parameters fixed: $p = 1\%, r = 75\%$, and $1-h = 50\%$.

Propagation round trip time (RTT): The first set of path RTT configurations we used are symmetric, i.e. identical RTT values were configured with the `tc` command for the two paths. Seven RTT values were tested: 1ms, 10ms, 20ms, 40ms, 60ms, 80 ms, and 100ms.

It is often the case that the multiple packet forwarding paths available for a remote VPN client to reach its home VPN server have different RTTs as these paths typically are provided by different network operators. For this reason, we decided to also evaluate the performance of MPTCP tunnels in the presence of asymmetric path conditions. For the asymmetric tests, both the primary and secondary paths were configured with the same loss rates, but different RTT values. More specifically, we experimented with five primary to secondary path RTT ratios. Table 2 shows these primary and secondary path RTT values and their ratios.

Table 2: Path RTTs Used in Asymmetric Tests

RTT Ratio	**Primary Path**	**Secondary Path**
1:1	20ms	20ms
1:2	20ms	40ms
1:3	20ms	60ms
1:4	20ms	80ms
1:5	20ms	100ms

MPTCP path managers: The 0.90 version of MPTCP software comes with two path managers for controlling the creation of subflows over available data paths. The *Fullmesh* path manager is designed to explore all paths between two hosts by attempting to connect each interface of one host to each interface of the other [2]. By default, it creates one subflow per path; this value can be increased with additional configuration.

The *Ndiffports* path manager is designed to "exploit the equal costs multiple paths that are available in a data center" [2]. Additionally, it allows nodes with only one interface to utilize MPTCP by creating multiple subflows for one path (i.e. the same source and destination IP address pair) through the use of different source port numbers. By default, it creates two subflows per path; this value can be increased with additional configuration.

3.1.2 Test Scenario and Performance Metric

We modeled the typical VPN scenario where a remote user establishes a VPN connection with a home VPN server and then downloads data from an intranet web server. Specifically, for each experiment, we first set up the VPN tunnel with a specific transport protocol and path configuration, and then ran the `wget` utility on the "VPN Client" node to download a 16-MB file from the "Web Server" 20 times.

As the tunnel carries standard TCP traffic with built-in congestion control, we use the *average goodput* as the main performance metric and skip other common transport layer issues such as fairness of bandwidth sharing. The goodput of a successful download was calculated by dividing the file size (i.e. 16 MB) by the total time required for the download. The average goodput and 95% confidence interval for each experiment were then derived from the 20 goodput samples.

It should be noted that the VPN tunnel was established without the use of compression or encryption to allow for ease of data collection and analysis. Furthermore, after preliminary tests and survey of guidelines from various sources regarding TCP performance tuning, we decided to optimize some of the TCP specific OS parameters for our specific testbed using the `sysctl` command. The results are illustrated in Table 3.

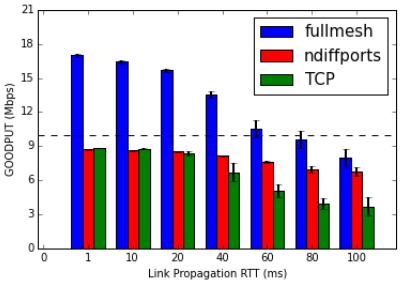
Table 3: Baseline TCP Configuration

TCP specific OS Parameters	Value
net.ipv4.tcp_congestion_control	Cubic or BALIA
net.ipv4.tcp_rmem (min, default, max)	4096, 87380, 6291456
net.ipv4.tcp_wmem (min, default, max)	4096, 16384, 4194304
net.core.rmem_max	212992
net.core.rmem_default	212992
net.core.wmem_max	212992
net.core.wmem_default	212992
net.ipv4.tcp_no_metrics_save	Enabled

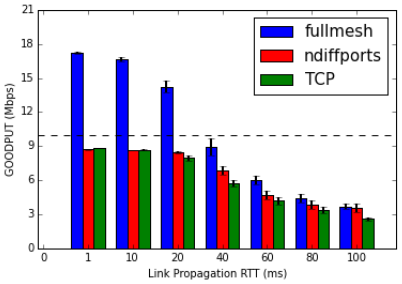
3.2 MPTCP over Symmetric Paths

Our first set of experiments over symmetric paths was aimed to compare the Cubic and BALIA congestion control algorithms. The default TCP congestion control algorithm used in Linux is Cubic. Cubic does not key on ACK messages to adjust the congestion window. Instead, Cubic will quickly increase the TCP congestion window to a threshold level and then probe for additional bandwidth that may be available. This form of congestion control has been shown to be optimal for single paths with high latency. However, prior work [15, 16] reports that when Cubic is used with MPTCP and there is a shared bottleneck with other TCP connections, the MPTCP user is able to obtain a larger share of the bottleneck's bandwidth. As one of the solutions to this problem, the BALIA algorithm [15] has been developed.

Figure 3 shows the performance of the two algorithms under a uniform packet loss rate of 0.1%. (The trend is similar with no packet loss or other loss rate settings.) The results confirm that Cubic is much more aggressive than BALIA in terms of bandwidth usage. Given Cubic's known unfriendly behavior toward regular TCP connections [15, 16], we used BALIA exclusively for the rest of the experiments.



(a) Cubic

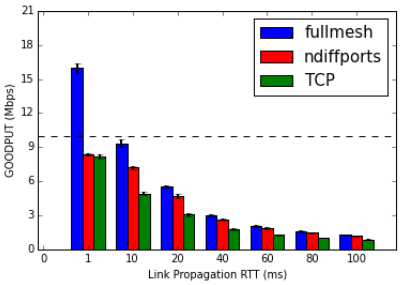


(b) BALIA

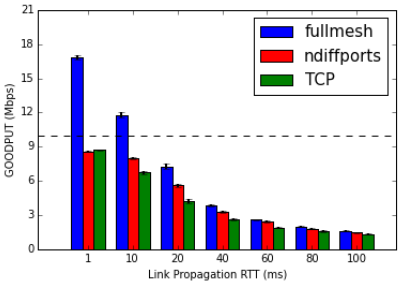
Figure 3: Performance of Cubic vs. BALIA congestion control under 0.1% uniform packet losses

We introduced the GE packet loss model in the next set of experiments. Figure 4 shows the results under 1% packet losses. As expected, the fullmesh MPTCP VPN tunnel configuration shows a noticeable improvement over the single path TCP tunnel. This improvement becomes less significant as the path propagation RTTs are increased. Surprisingly, MPTCP performed better under the GE bursty loss model. We believe this is due to the TCP fast recovery mechanism (RFC 2001), which halves the congestion window only once upon consecutive packet losses. It is also interesting to note the performance of the ndiffports MPTCP path manager. Some MPTCP developers consider this path manager to have little practical use [2], but our results show that using

ndiffports can improve the goodput experienced. Finally, the relative performance trend seen in Figure 4 remains much the same for different packet loss rate settings. We omit those plots for brevity.



(a) Uniform loss probability



(b) GE bursty loss model

Figure 4: Performance of MPTCP vs. single path TCP under 1% packet losses

3.3 MPTCP over Asymmetric Paths

In this set of experiments, we evaluated only the fullmesh path manager. The ndiffports path manager would not make sense for asymmetric testing since it is designed for hosts with a single interface. The packet loss rates were controlled using the GE model according to Table 1, and the path RTTs were varied according to Table 2.

The results are shown in Figure 5. As expected, the best performance was obtained with the path RTT ratio of 1:1. Interestingly, the performance degradations due to path asymmetry, while noticeable, did not worsen as the degree of asymmetry was increased. The results show that MPTCP's built in mechanisms for handling path asymmetry [11] were largely effective in our experiments.

4 Design and Evaluation of MPUDP

The previous section demonstrated the potential VPN performance gains from using a MPTCP tunnel when multiple physical network paths are available. However, as MPTCP applies standard TCP congestion con-

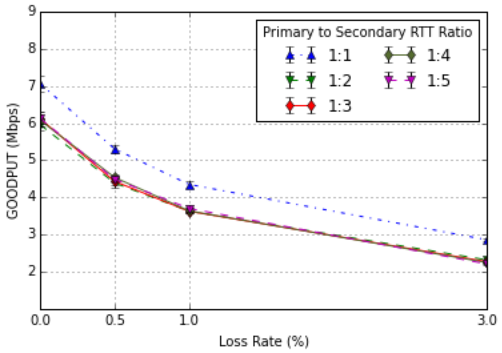


Figure 5: MPTCP performance over asymmetric paths under the GE packet loss model

trol for each subflow, we hypothesized that the use of MPTCP will face the same adverse effects of nested and often excessive congestion control as in typical TCP over TCP settings [5, 14]. To confirm such effects, using the physical testbed and symmetric link test procedures from Section 3, we compared single path UDP and TCP tunnel performance. Figure 6 plots the percent of goodput increase by a TCP-in-UDP VPN file download relative to that of a baseline TCP-in-TCP connection for a spectrum of RTT and packet loss rate combinations. The results clearly show the performance advantages of using UDP. For this reason the default transport protocol for various VPN software is set to UDP.

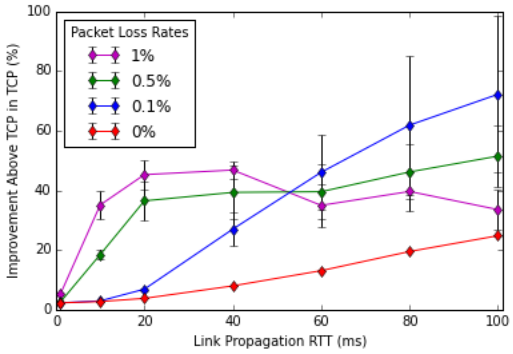


Figure 6: Goodput increase by using UDP vs. TCP for a single path VPN tunnel with uniform packet loss rates. Error bars show 95% confidence intervals.

4.1 Adding MPUDP to Linux with Loadable Kernel Modules

In order to evaluate our hypothesis, we have chosen to design and implement basic MPUDP functionality into one of the latest MPTCP capable Linux kernels (i.e. v.

90). While it is feasible to add this functionality by revising the core Linux networking kernel code, as has been done for MPTCP [11], to expedite debugging and testing, we decided to implement MPUDP primarily as loadable kernel modules (LKMs) as doing so requires a minimum amount of kernel recompilation.

Specifically, we have created two LKMs in C, with source files named `MPUDP_send.c` and `MPUDP_recv.c` respectively. We have also added 6 lines of code[1] to the Linux kernel implementation of UDP (i.e. `udp.c`) for invoking functions defined in the LKMs as required. The main function of the send side LKM identifies an additional outgoing interface to support a new subflow and splits traffic in a 50/50 probabalistic fashion (i.e. 1:1) among the two outgoing interfaces while the function of the receive side LKM coalesces the two subflows into a single input stream of packets to the destination UDP socket used by the VPN tunnel. More details of our MPUDP prototype are described in reference [9], which includes the design of a general method for the two tunnel end points to enable MPUDP and agree on the set of interfaces to leverage for multipath communication.

4.2 Evaluation

To evaluate the performance of MPUDP vs. MPTCP, we first performed file downloads using each in symmetric link settings with zero simulated packet losses. For additional comparison, we also performed single path UDP downloads under the same conditions. The results are plotted in Figure 7. They show that MPUDP performs better than MPTCP and the difference becomes more significant as the RTT increases. This confirms our hypothesis that MPTCP is prone to TCP-in-TCP performance issue; in this case, the TCP subflows are slow to grab the available bandwidth due to the TCP slow start and congestion avoidance algorithms.

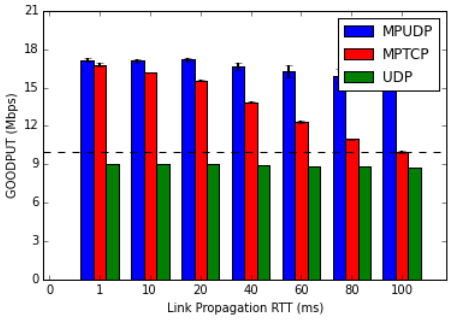
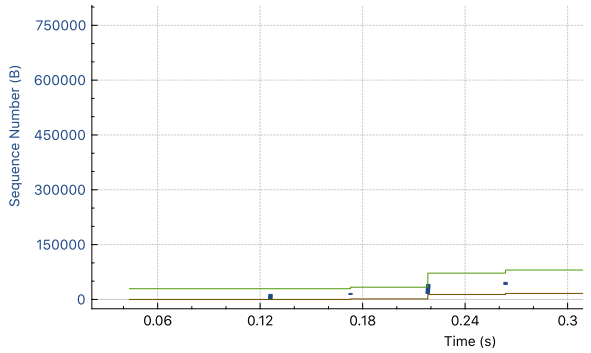


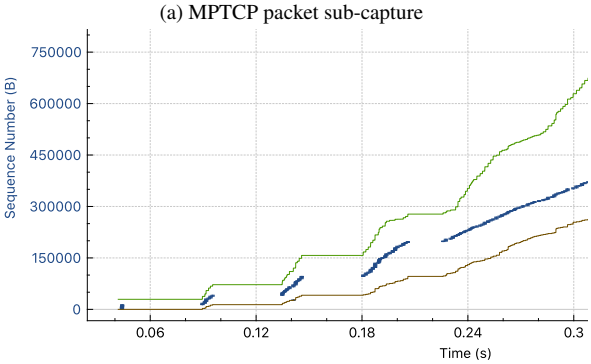
Figure 7: Performance of MPUDP vs. MPTCP and single path UDP under zero packet loss.

[1]The code of our implementation is available at Github: https://github.com/danluke2/mpudp_vpn_thesis.

To better see the effect of TCP-in-MPTCP, we examine the 40ms RTT test of Figure 7 more closely using packet traces collected at the web server. We randomly select one trace each for MPTCP and MPUDP and use Wireshark to drill down to the evolution of the send sequence number (blue curve), acknowledge number (brown curve) and expected received byte count (green curve) during the first 0.3 seconds of the downloads. Figure 8 illustrates that the MPTCP tunnel was much slower than its MPUDP counterpart in grabbing the available bandwidth.



(a) MPTCP packet sub-capture

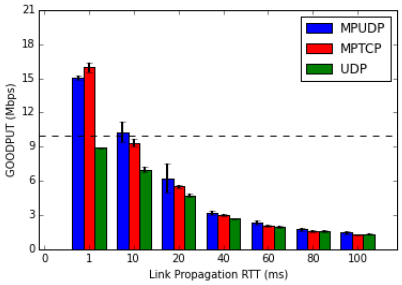


(b) MPUDP packet sub-capture

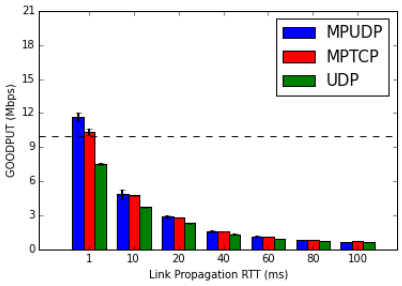
Figure 8: Close up analysis of MPTCP and MPUDP performance with path RTT = 40ms

Next, we introduced packet losses into the evaluation. We used two types of loss distributions: (i) uniform, and (ii) GE burst model, as done in Section 3. The representative results are plotted in Figure 9. They show that with packet losses, MPUDP still outperformed MPTCP; however, the advantage is not as significant because the redundant congestion control with MPTCP likely had some positive effect in reducing packet retransmissions.

Last, our MPUDP implementation does not have any mechanism to re-sequence packets from multiple subflows while MPTCP has a complex method to ensure packets to be in order at the receiver. To understand the performance impact of this design difference, we repeated the downloads with asymmetric link RTTs that



(a) 1% Packet loss rate (uniform probability)



(b) 3% Observed packet loss rate (GE burst model)

Figure 9: Performance of MPUDP vs. MPTCP and single path UDP under packet losses.

should exacerbate the packet reordering problem. We experimented with each of the five RTT configurations as given in Table 2 under four different packet loss rates as given in Table 1. We illustrate the results in Figure 10, where the percentage of goodput change from MPTCP to MPUDP is plotted. It is clear that our MPUDP implementation, which remains lightweight like the standard UDP – by not having a re-sequencing mechanism and splitting traffic using a probabilistic 50/50 method, can perform poorly relative to MPTCP when the paths have different delay characteristics. As expected, the effect was more pronounced for a higher degree of link delay asymmetry. Interestingly, the relative MPUDP performance improved as the loss rate increased.

4.2.1 Path Aware Traffic Splitting

One observation from Figure 10 is that the packet losses didn't seem to compound the effect of packet re-ordering as one might expect. That led us to a hypothesis that a path aware traffic splitting method, i.e. one that takes the path RTT difference into consideration might mitigate to a large extent the negative effect of asymmetric link delays on MPUDP performance.

To evaluate this hypothesis, we revised the function of the send LKM to support custom traffic split ratio (i.e. $x\%/(100-x)\%$), and then repeated the downloads of Figure 10 using two new splits: 75%/25% and 90%/10%.

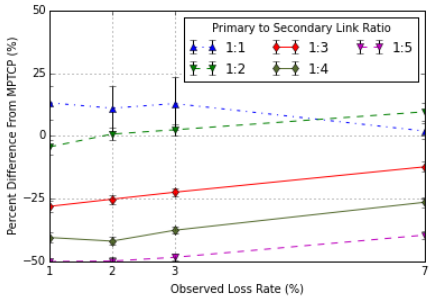
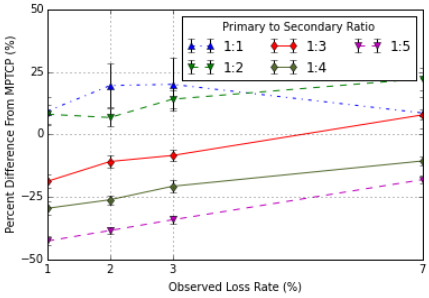


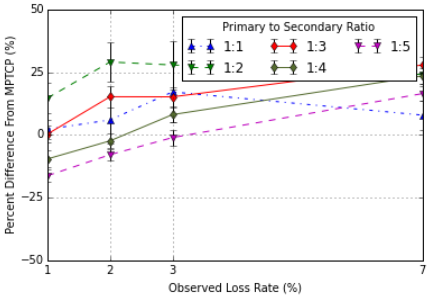
Figure 10: Percent of goodput change from MPTCP to MPUDP under different link RTT ratios and loss rates. A positive (negative) percentage value indicates an increase (decrease) of performance with MPUDP.

The results are plotted in Figure 11. They show that by shifting traffic away from the link of larger RTT, the relative performance of MPUDP improved.

Therefore, if the MPUDP sender has the ability to estimate the RTT differences among the available paths, it should split traffic according to the RTT estimates, proportionally putting more packets on the faster path(s). One method to acquire such estimates might be to selectively add timestamps to a subset of packets on each path and have the receiver echo back these timestamps.



(a) 75/25 traffic split



(b) 90/10 traffic split

Figure 11: MPUDP relative performance improved by sending less traffic on the path of larger RTT.

5 Discussion

In this section, we discuss additional performance factors beyond the scope of our evaluation. We also identify some limitations of our study and pertinent future work.

First, from Figures 7, 9, and other relevant data, we observe that the performance of MPTCP, MPUDP and UDP all declined sharply for paths with moderate to long RTTs (20ms to 100ms) and low to moderate loss rates (0.1% to 3%). One can hypothesize that a forward error correction (FEC) scheme might improve the overall goodput in these settings. The main trade-off would be the communication and computational overhead incurred for providing FEC.

Second, our experiments with asymmetric link configurations did not include asymmetric link transmission rates. As discussed in Section 4.2.1, it may be important for MPUDP to be informed of such rate differences in order to maximize performance. Therefore, the question of how to maintain up to date knowledge of individual path performance characteristics should be a fertile ground for future work on MPUDP.

Last, some enterprise applications use UDP, and typically their performance concern is not goodput, but message timing and/or fair sharing of bandwidth with other applications. While this paper focuses on TCP as the end-to-end transport protocol, it will be beneficial to study the impact of multipath VPN transport on UDP applications.

6 Conclusion

The empirical results presented in this paper provide strong evidence that MPTCP can increase the performance of TCP applications over a VPN tunnel when multiple data paths are available between the VPN end points. Consistent performance gains have been observed for a wide spectrum of path characteristics in terms of packet losses, propagation delays, and path asymmetry.

The additional performance gains from the basic MPUDP functionality, which is the case even in the presence of path asymmetry, is both surprising and encouraging, given the design's simplicity and relatively weak assumption of a path-aware packet scheduler. We believe more studies are necessary to understand the TCP-in-MPUDP dynamics and find a good balance point between controlling complexity and maximizing performance.

Acknowledgements

We would like to thank Henry Foster and anonymous reviewers for their helpful comments.

References

[1] BOCCASSI, L., FAYED, M. M., AND MARINA, M. K. Binder: A system to aggregate multiple Internet gateways in community networks. In *Proc. ACM MobiCom Workshop on Lowest Cost Denominator Networking for Universal Access* (2013).

[2] BONAVENTURE, O. Blog entry: Recommended Multipath TCP configuration, 2014.

[3] BOUTIER, M., AND CHROBOCZEK, J. User-space Multipath UDP in MOSH. Accessed January 15, 2017.

[4] FEILNER, M., AND GRAF, N. *Beginning OpenVPN 2. 0. 9: Build and Integrate Virtual Private Networks Using OpenVPN*. Packt Publishing Ltd, 2009.

[5] HONDA, O., OHSAKI, H., IMASE, M., ISHIZUKA, M., AND MURAYAMA, J. Understanding TCP over TCP: Effects of TCP tunneling on end-to-end throughput and latency. In *Proc. SPIE* (2005), vol. 6011.

[6] HSIEH, H.-Y., AND SIVAKUMAR, R. ptcp: An end-to-end transport layer protocol for striped connections. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on* (2002), IEEE, pp. 24–33.

[7] KHALILI, R., GAST, N., POPOVIC, M., UPADHYAY, U., AND LE BOUDEC, J.-Y. MPTCP is not pareto-optimal: performance issues and a possible solution. In *Proc. ACM Int. Conf. on Emerging networking experiments and technologies* (Dec. 2012).

[8] KUZNETSOV, A. N. Linux traffic control (tc) command. https://linux.die.net/man/8/tc.

[9] LUKASZEWSKI, D. Multipath transport for virtual private networks. M.S. thesis, Naval Postgraduate School, 2017.

[10] MAGALHAES, L., AND KRAVETS, R. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *Network Protocols, 2001. Ninth International Conference on* (2001), IEEE, pp. 165–171.

[11] PAASCH, C. *Improving Multipath TCP*. PhD thesis, UC Louvain, Belgium, Nov. 2014.

[12] PAASCH, C., AND BARRE, S. Multipath TCP in the Linux kernel. Accessed January 15, 2017.

[13] PIERRE EBERT, J., AND WILLIG, A. A Gilbert-Elliot bit error model and the efficient use in packet level simulation. Tech. rep., Technical University Berlin, 1999.

[14] TITZ, O. Why TCP over TCP is a bad idea. http://sites.inka.de /bigred/devel/tcp-tcp.html. Accessed January 15, 2017.

[15] WALID, A., PENG, Q., LOW, S. H., AND HWANG, J. Balanced linked adaptation congestion control algorithm for MPTCP. Internet-draft, Internet Engineering Task Force, Jan. 2016.

[16] WISCHIK, D., RAICIU, C., GREENHALGH, A., AND HANDLEY, M. Design, implementation and evaluation of congestion control for multipath TCP. In *Proc. USENIX conference on Networked systems design and implementation* (2011), pp. 99–112.