BOOST: Transport-Layer Multi-Connectivity Solution for Multi-WAN Routers

Kariem Fahmi, Douglas Leith
Trinity College Dublin, Ireland

***Abstract*—In this paper, we discuss the challenges faced by MPTCP when used to aggregate multiple WAN/Internet connections in Multi-WAN Routers (MWR). We observe that the two architectural variants, proxying and tunneling, used to deploy MPTCP in MWR suffer from key performance problems. First, the proxy variant creates one MPTCP connection for each TCP connection, which results in a large number of parallel uncoordinated MPTCP connections, which we explain leads to underutilization of the available capacity, suboptimal multi-path scheduling, and increased loss rate. Second, the tunnel variant, which relies on encapsulating TCP over MPTCP, stacks two reliability layers and leads to a large number of spurious retransmissions, an issue known as TCP meltdown. Instead, we propose a new multi-path solution more suited to MWR, called BOOST. This solution eliminates the problems with both the proxy and tunnel approaches by multiplexing TCP connections over a single persistent multi-path connection. BOOST also takes a novel approach to multi-path scheduling that combines multi-path load balancing and scheduling. In particular, short flows are transmitted across a single link to avoid HoL blocking while longer flows are opportunistically transmitted across multiple paths, utilizing left-over capacity. Evaluations show that BOOST provides better throughput, lower losses, and retransmissions while requiring less memory compared to both MPTCP variants**

I. Introduction

Multi-WAN Routers (MWR) provide WAN/Internet connectivity by combining two or more backhaul links (e.g. 2 LTE connections) for improved capacity and reliability. They are commonly used in public transportation services, such as trains, buses, ships and even airplanes to provide seamless Internet connectivity to a large number of passengers using wireless backhauls, which are known to deteriorate under high mobility. For example, providers of train-to-ground (T2G) communication combine LTE/5G from different carriers to improve the capacity for on-board internet access and increase reliability for mission-critical applications such as communication based train control (CBTC) or closed circuit TV (CCTV) [1].

MPTCP is used by some commercial MWRs as a transport layer solution for allowing individual TCP connections from user devices to utilize the capacity of all the available backhauls [2]. An MPTCP MWR needs to either transparently split incoming TCP connections into a TCP and an MPTCP portion, like a proxy, or encapsulate them inside MPTCP, like a tunnel. In both cases, a network-edge proxy-server/tunnel-server is used to translate/decapsulate the traffic back into TCP to be forwarded to the content server. Figures 1 and 2 show the architecture and network topology for the proxy and tunnel variants respectively.

In the proxy variant of the MWR, each user TCP connection is terminated at the MWR and its packets are transmitted using MTCP to the network-edge proxy. Thus for each TCP connection, a corresponding MPTCP connection is created. Our measurements of passenger traffic in a train to ground (T2G) MWR show that this results in thousands of concurrent MPTCP connections, each with their independent congestion control on each link plus their own multi-path scheduler, and we find that this lead to low capacity utilization and high loss, as will be explained in more detail later.

In the tunnel variant of the MWR, the packets from user TCP connections are encapsulated and transmitted over all the available backhauls using a single persistent MPTCP connection. That is, a single MPTCP connection carries all of the user TCP connections over the backhaul. This eliminates the large number of connections observed with proxy variant but creates new issues. In particular, the tunnel stacks two reliability layers (MPTCP and TCP), each with its own congestion control and retransmission mechanism for lost packets. This can lead to so-called TCP meltdown [3], where a high number of spurious re-transmissions are constantly triggered. Additionally, there is cross-flow Head of Line (HoL) blocking between separate user flows sharing the same MPTCP connection.

A second key issue that is common to both architectural variants is frequent HoL blocking due to the heterogeneity of the wireless links. This type of blocking occurs when packets arrive out of order, leading to them being placed into a reordering queue until they can be delivered in-order, increasing delay. This delay disproportionately impacts short flows (e.g. web browsing) as they are much more latency-sensitive compared to long flows (e.g. file download). HoL blocking is also more prevalent when scheduling over heterogeneous links with different delay profiles, which are commonly used in MWRs. For example, a T2G MWR tends to utilize multiple LTE links from different carriers, which are likely to have different delay profiles due to different coverage, cross-traffic and core-latency.

In this paper we propose a new multi-path transmission solution called BOOST that addresses the aforementioned issues. Concretely, we make the following contributions

- We propose a MWR architecture that eliminates the problems with both the MPTCP proxy and tunnel approaches by multiplexing user traffic over a single, persistent multi-path connection using modified version of the QUIC protocol that re-uses its streaming feature.
- We propose a hybrid approach to multi-path scheduling. Namely, short flows are transmitted across a single link to avoid HoL blocking while longer flows are opportunistically transmitted across multiple paths, utilizing left-over capacity, thereby improving efficiency and ro-

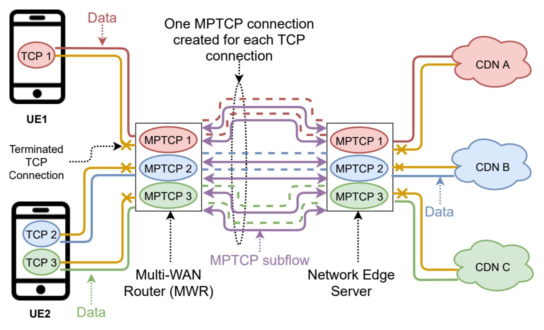


Fig. 1: MPTCP Proxy MWR

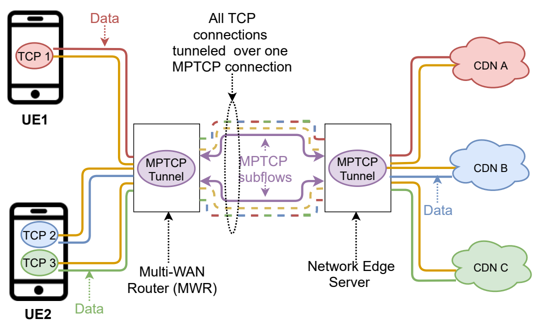


Fig. 2: MPTCP Tunnel MWR

bustness.

- We evaluate BOOST against both MPTCP variants in a trace-driven emulation and show that it provides better throughput, with lower losses and re-transmissions, while requiring less memory.

II. Issues with an MPTCP proxy MWR

As briefly discussed in the introduction, an MPTCP proxy MWR variant establishes one MPTCP connection for each TCP connections, as shown in Figure 1. According to measurements of passenger traffic in a production T2G communication system, this creates thousands of uncoordinated MPTCP connections, most of which are short or on/off (i.e. long-lived with low frequency of data exchange such as HTTP connections with keep-alive). In this section we discuss why the distributed multi-path scheduling and link estimation, and the repeated establishment of secondary subflows is likely to lead to poor performance in MPTCP proxy MWRs.

a) Distributed multi-path scheduling: The MPTCP default scheduler, MinRTT, works by filling up the CWND of all subflows in order of the lowest smooth round trip time (SRTT). In order for a connection to utilize more than one link, it needs to have enough packets in-flight at once to fill the CWND of the link with the lowest SRTT. First, short connections that transmit less than 10 packets will only ever use one link, since all their data will fit into the initial CWND [4] of the default subflow. Furthermore, since that subflow is always established over the default route configured in the operating system routing table, it will cause the performance of short connections to be highly sensitive to the choice of default route. Second, on/off connections, which are long-lived connections with in-frequent data exchange such as HTTP connection with keep-alive, will frequently not have enough data in flight at once to utilize more than one link, leading to poor capacity utilization.

b) Distributed link estimation: MPTCP relies on data transmission for estimating the CWND and the SRTT of its subflows. Accurate estimation of these values is necessary for optimal multi-path scheduling and capacity utilization. However, the MPTCP connections created in the MWR will perform isolated estimations of the links and will not share information between each other. As a result new connections and on/off connections will have no estimate or an expired estimate for these key two link indicators. Having an inaccurate estimate for the CWND can cause undershooting of capacity, wasting capacity, or overshooting of capacity, causing losses. Particularly for short flows that are latency sensitive this can significantly increase their completion time. Having an inaccurate estimate for the SRTT can cause suboptimal multi-path scheduling as packets will be placed on the slower link first. Finally, even long flows that use loss-based congestion control (e.g. CUBIC [5]) will each have to experience a loss before lowering the sending rate, creating un-necessary losses.

c) Repeated secondary subflow establishment: Every new MPTCP connection needs to first establish the default subflow using the TCP 4-way handshake and then perform the same handshake for each additional subflow it establishes before it can be used to transmit data. Depending on the RTT of the secondary subflows, multiple CWND worth of data may be transmitted on the default subflow before the secondary subflows are ready. This may cause applications that transmit more than the initial CWND worth of data to also utilize a single link, further lowering capacity utilization.

III. Issues with an MPTCP tunnel MWR

MPTCP tunnel MWRs are based on encapsulating TCP, and possibly UDP, connections and transmitting them over a single persistent MPTCP connection, as shown in Figure 2. While this approach eliminates the high number of parallel MPTCP connections in the proxy variant, it introduces new performance issues due to stacking multiple layers of reliability and cross-flow HoL blocking.

a) Stacking multiple layers of reliability: When a new TCP connection is created, it will often have a short re-transmission timer than the established MPTCP tunnel connection, as the latter is more adapted to the link properties. When data is lost in the MPTCP tunnel, it will queue up a re-transmission and increase its timeout. However, since the MPTCP tunnel will be blocked during the re-transmission period, the TCP connection does not get any acknowledgments, and it will thus queue up its own re-transmission of the same data. Since the timeout in the tunnel is less than the TCP connection, it may queue up even more re-transmissions causing the transmission of new data to stall completely, and every re-transmissions further compounds this problem. This behavior is referred to as TCP meltdown [3]. In wireless links under mobility, such as in trains, which experience frequent losses due to capacity fluctuations and handovers, frequent TCP meltdowns will occur.

b) Cross-flow HoL blocking: When data is received out of order due to loss or multi-path transmission over heterogeneous/time-varying links, received packets are held

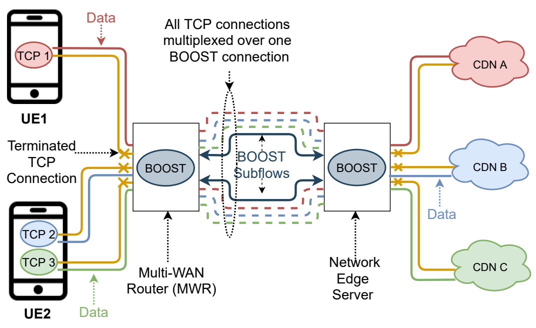


Fig. 3: BOOST MWR

in a reordering buffer until they can be delivered in order. This is referred to as HoL blocking. In an MPTCP tunnel MWR, there is a single reordering buffer for all TCP connections being transported. Therefore, packets belonging to one TCP connection will have to wait in the re-ordering buffer needlessly until packets from another connection can be delivered in-order simply because packets for the latter were sent first. This issue is similar to the well-known HTTP head of line blocking problem [6].

IV. BOOST

The proposed solution, referred to as BOOST, is based on two main design features: multiplexing application flows over a single persistent multi-path connection and hybrid multi-path scheduling. In this section we will discuss both in detail.

A. Persistent multi-path connection with multi-plexing

The first feature of BOOST is that it transparently proxies and multiplexes TCP connections over a single persistent multi-path connection between the MWR and a network-edge server, as shown in Figure 3. When the BOOST client first starts, it establishes a connection with the BOOST server and then establishes all secondary subflows (details of the protcol are discussed in the next section). The BOOST connection has a single multi-path scheduler and a single instance of congestion control for each link. When new TCP connections are routed through the MWR, BOOST will terminate each TCP connection and multiplex the data over the existing multi-path BOOST connection. Data from individual TCP connections is tagged using a unique stream identifier to allow the receiver to distinguish between them.

This feature of BOOST addresses the problems in the variants of the MPTCP MWR as follows.

a) Underutilization of capacity: Since all application traffic is handled by one multi-path scheduler, it allows the scheduler to optimally distribute all application traffic across the available links, eliminating the underutilization of capacity by short flows and on/off flows.

b) Increased loss rate and suboptimal scheduling: BOOST maintains a single estimate for the CWND and the SRTT on each link that is generated by data transmitted from all application flows. This allows it to derive a much more accurate estimate of these parameters, leading to lower loss, better capacity utilization and more optimal multi-path scheduling. Additionally, since there is only one single congestion control process, a single loss on a link is sufficient for BOOST to reduce its sending rate, compared to distributed congestion control where each flow has to experience an independent loss.

c) Repeated subflow establishment: BOOST establishes all subflows once during its lifetime, avoiding the need to re-establish the subflows for each TCP connection

d) Stacking multiple layers of reliability: BOOST functions as a proxy by splitting TCP connections and thus does not stack multiple layers of reliability

e) Cross-flow HoL blocking: BOOST stores a unique stream identifier inside each packet that links to its parent application flow and hence can avoid cross-flow HoL blocking.

B. The BOOST protocol

The BOOST protocol is based on QUIC with modifications to enable multiplexing of TCP connections and multi-connectivity.

1) Multi-connectivity: To enable multi-connectivity, we adopt an existing proposal [7] for a multi-path version of QUIC (MPQUIC) that modifies the protocol in two ways. First, the main QUIC packet header includes a new PathID field which identifies which subflow the packet belongs to. Furthermore, the existing ConnectionID field is used to link a path to the original connection. There are no handshakes needed to establish secondary subflows like in MPTCP. Second, acknowledgement (ACK) packets also include the PathID field to identify which path the ACK packets belong to. Finally, the PacketNumber field is specific to each path.

2) Multiplexing: QUIC already includes a multi-streaming functionality, so we leveraged this to enable our multiplexing feature. In the BOOST implementation of QUIC, we modify the use of streams to refer to TCP connections instead of the traditional application streams. Each TCP connection is assigned a unique StreamID by BOOST.

To enable each proxied TCP connection to be forwarded to its correct final destination, BOOST implements a new QUIC extension/frame called SET_FORWARD_ADDRESS. It contains the following information: ConnectionID, StreamID, and the application destination ip, port and Protocol. This frame is transmitted with the first packet sent for an application flow.

C. Hybrid multi-path scheduling

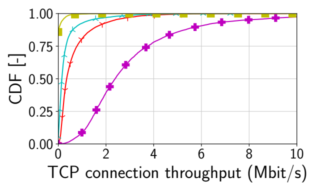
The second key feature of BOOST is the hybrid multi-path scheduling approach which uses multi-path load balancing for short flows in order to minimize delay by preventing HoL blocking and opportunistic multi-path scheduling for longer flows to make use of left-over capacity.

The scheduler is split into 4 phases as follows

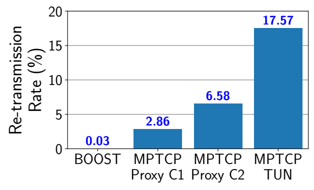
1) Phase 1 (Flow-link assignment): The scheduler assigns application flows to available links according to the ratio R_i, which is calculated as follows

$$R_i = \frac{t_i}{\sum_{j=0}^{m} t_j} \quad \text{and} \quad t_i = \frac{CWND_i}{SRTT_i} \tag{1}$$

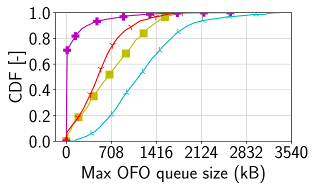
where $CWND_i$ is the CWND of link i, $SRTT_i$ is the SRTT of link i, and m is the number of available links. The goal is to load balance flows over links proportionate to the ratio of the throughput of the links. Load balancing occurs periodically every γ milliseconds, where γ is a design



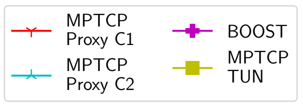
(a) Throughput



(b) Retransmissions



(c) Max out-of-order queue size



(d) Legend

Fig. 4: Evaluation of the transmission of passenger internet traffic from the on-board WiFi in a Train-to-Ground communication system, using a MWR with two LTE carriers, when using MPTCP Proxy with default route on carrier 1 (MPTCP Proxy C1), on carrier 2 (MPTCP Proxy C2), MPTCP tunnel (MPTCP TUN) and BOOST.

parameter we choose to be 100ms, and every time a new flow is established. Applications flows that have not received new data for over one second, are removed from the assignment.

2) Phase 2 (load balancing slot allocation): After assigning a flow to a link, it is inserted into a priority queue ordered by the amount of bytes the flow has transmitted so far. If a flow has not transmitted anything yet, this value is 0. The front flow in the priority queue is then selected and at most one maximum transmission unit (MTU) worth of data, which is typically 1500 bytes, is allocated on the assigned link's send queue for use by this flow. The bytes transmitted counter for the flow is then incremented by the amount of data allocated on the link and the flow is re-inserted into the priority queue. This process is repeated until all flows assigned to that link have been allocated space equal to the data in their send queue (i.e. they have no more data) or the total allocated data on a link is equal to its available CWND.

3) Phase 3 (Opportunistic multi-path scheduling slot allocation): This phase of the BOOST scheduler will only trigger if at least one link has space in its CWND and at least one flow has data that has not been allocated space on a link. All eligible application flows are placed in another priority queue that is also ordered by the amount of data transmitted just like before. A multi-path scheduling algorithm will then select between all available links to allocate space for data from the selected flow. For example, the MinRTT scheduler would pick the link with the lowst SRTT amongst the links that have space in their CWND. Afterwards the flow's transmitted data counter is incremented and it is returned to the priority queue.

4) Phase 4: Multi-path scheduling: In this phase packets are taken from the application flow send queue and placed on links with allocated space. If phase 3 did not occur (i.e. a flow only has allocated space on one link) then packets are placed on that link in order of their sequence number. If phase 3 did occur, the same multi-path scheduling algorithm as in phase 3 assigns packets from the application flow send queue to the allocated space on the links.

D. Evaluation

We evaluated the performance of BOOST and the MPTCP variants using a trace-driven emulation of the transmission of passenger internet traffic from an on-board WiFi service in a production Train-to-Ground communication system. Figure 4 shows the results for MPTCP Proxy with default route on carrier 1 (MPTCP Proxy C1), on carrier 2 (MPTCP Proxy C2), MPTCP tunnel (MPTCP TUN) and BOOST.

Figure 4(a) shows the CDF of the TCP connection downlink throughput during its periods of activity. Since many TCP connections in the passenger traffic are on/off and almost all the traffic is downlink traffic, throughput is only calculated for the periods when a connection is actively transmitting data on the downlink, and the mean throughput over all periods of activity in each connectionis used. As can be seen, around 50% of the TCP connections under BOOST had a mean throughput of more than 2.2 Mbit/s, which is around 10 times higher than any of the MPTCP variants. Figure (b) shows the re-transmission rate, and it can be seen that the MPTCP TUN had extremely high loss rate, due to the TCP meltdown effect, while BOOST maintained a much lower loss rate compared to any MPTCP variant. Figure (c) shows the CDF of the total out-of-order queue size for all schemes. The hybrid multi-path scheduler allowed BOOST to practically eliminate the need for an out-of-order queue since most TCP connections were load balanced over individual links.

Overall, it can be seen that BOOST provides better throughput, with lower re-transmissions, and a smaller memory footprint compared to all MPTCP variants.

References

[1] Train to ground. [Online]. Available: https://www.nokia.com/networks/solutions/train-to-ground/

[2] Multipath TCP solutions for better connectivity. [Online]. Available: https://www.tessares.net/

[3] I. Coonjah, P. C. Catherine, and K. M. S. Soyjaudah, "An investigation of the TCP meltdown problem and proposing raptor codes as a novel to decrease TCP retransmissions in VPN systems," in *Information Systems Design and Intelligent Applications*, ser. Advances in Intelligent Systems and Computing, S. C. Satapathy, V. Bhateja, R. Somanah, X.-S. Yang, and R. Senkerik, Eds. Springer, pp. 337–347.

[4] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin, "An argument for increasing TCP's initial congestion window," vol. 40, pp. 26–33.

[5] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," vol. 42, pp. 64–74.

[6] How does HTTP/2 solve the head of line blocking (HOL) issue. [Online]. Available: https://community.akamai.com/customers/s/article/How-does-HTTP-2-solve-the-Head-of-Line-blocking-HOL-issuelanguage=en-US

[7] Q. De Coninck and O. Bonaventure, "Multipath QUIC: Design and evaluation," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. Association for Computing Machinery, pp. 160–166. [Online]. Available: https://doi.org/10.1145/3143361.3143370