Fast Reinjection for Intermittent MPTCP Connections

Ralf Lübben
Flensburg University of Applied Sciences
Flensburg, Germany
Email: ralf.luebben@hs-flensburg.de

Sascha Gübner
Robert Bosch GmbH
Hildesheim, Germany
Email: Sascha.Guebner@de.bosch.com

Philip Wette
Robert Bosch GmbH
Hildesheim, Germany
Email: Philip.Wette@de.bosch.com

***Abstract*—Combining multiple networks at a host to access the Internet (multi-homing) allows for continuous data transmission even in scenarios with one intermittent connection. To this end, a multi-path capable transport layer protocol, such as multi-path TCP (MPTCP), is required to simultaneously use two or more networks for data transmission. A typical use case is to combine WLAN and cellular networks, in which the user stays connected to the cellular network, if the WLAN connection is lost due to user mobility. In that scenario, multi-path protocols have to detect the connection loss and have to retransmit corresponding lost packets immediately, so that the receiving network stack can forward the packets in sequence to the application.**

This paper analyzes MPTCPs connection loss detection mechanism and the corresponding process of the reinjection of lost packets. We show that the mechanisms for connection loss detection originate from single-path TCP (SPTCP) and have been simply adopted to MPTCP. They work slowly and cause large application level delays. Therefore, we propose a new reinjection method to speed up the reinjection after a connection loss. We evaluate our approach with emulation and real-world experiments and show that we reduce application level delays by up to 80%.

I. Introduction

Nowadays, many devices are equipped with multiple network interfaces. Especially modern smartphones commonly have a WLAN and a cellular interface. The usage of those interfaces depends on the availability of the respective networks. WLAN is often the preferred technology, since it is usually less expensive, and cellular networks are only used if no WLAN connection is available. Moreover, the concurrent use of networks at the same time enable use cases, that require increased capacity or reliability. MPTCP is already implemented in major operating systems, such as Apple iOS, and is going to experience a large prevalence by the recent inclusion in the Linux mainline kernel.

When users are mobile, they constantly enter and leave communication ranges of wireless networks. Hence, on the transport layer, connections have to be established or removed, respectively. This paper focuses on the concurrent use of WLAN and cellular networks, while the user leaves the communication range of the WLAN. In this process, packets sent over the WLAN network may get lost. Thus, they have to be retransmitted over the still active cellular network. This retransmission on the remaining network is denoted as reinjection in the following, whereas retransmission means sending the packets over the same network again.

Handovers between WLAN and cellular networks are analyzed in [1] and the results indicate the benefits of using MPTCP for a seamless connection. However, the evaluation still shows short outages during the handover for a real-time voice application. MPTCPs reinjection strategy relies on the retransmission timeout (RTO) of SPTCP, which triggers retransmissions if the reception of previously sent data is not acknowledged. This timer is set conservatively, since it goes along with a strong decrease of the transmission rate due to congestion control. Thus, SPTCP relies on additional and faster mechanisms for retransmissions such as duplicate acknowledgements and fast retransmit, see [2]. Hence, the RTO is the last resort to trigger retransmissions that makes a conservative choice reasonable in SPTCP, but slows down the reinjection in MPTCP. This work contributes a detailed analysis of the timings that determine the outages on a connection loss. To speed up the transition from one network to the other and to reduce the outage time, a new timer is introduced that handles reinjection. The proposal is afterwards evaluated in an emulated network setup as well as in a real-world vehicular scenario with high mobility.

This paper is structured as follows: Sec. II elaborates on the mechanisms for packet reinjection in MPTCP and identifies reasons for slow reaction on a connection loss event. Sec. III introduces a new trigger for reinjections, which is evaluated in Sec. IV. Related work is given in Sec. V and Sec. VI summarizes the main findings of the paper.

II. Detection of Connection Loss

When using MPTCP over two or more networks, the logical connection between two endpoints is divided into multiple so-called subflows. A multi-homed host creates one subflow for each connected network interface. When data is to be sent, it is divided into (possibly multiple) packets, and they are assigned to and sent over the different subflows. This process is called scheduling. The receiver then reconstructs the logical connection by jointly processing the packets received over the different subflows.

In the following, a mobile device connected to a server using MPTCP with two subflows, one using WLAN and one using a cellular connection, is assumed. The default MPTCP scheduling is based on the minimal round-trip time (RTT) of the connection. Nevertheless, we want to emphasize that

the following description is universally applicable to scenarios in which multi-path protocols loose connections and have to retransmit packets in flight to achieve a reliable and in-order transmission of data.

A connection loss can be detected directly by data-layer events or indirectly by using timeouts (i.e., lack of acknowledgments). For example, if the connection is lost because the user leaves the communication range of a wireless network, the network stack learns about the loss from data-layer events. In consequence, a connection manager typically removes the configuration of all related interfaces. In MPTCP, this in turn triggers the removal of all related subflows. In our scenario, where the client is connected to a WLAN and traffic flows mainly in downstream direction (from a server to the client), the server learns about the WLAN connection loss by the client. The client does this by sending the MPTCP option REMOVE_ADDR to the server on one of the remaining subflows (which in our scenario is routed over the cellular network). Then, the server triggers a TCP keepalive for security purposes, see [3]. If the keepalive message is not answered, the server assumes the subflow to be broken.

The time between loosing the connection and transmission of the REMOVE_ADDR option is usually in the order of multiple RTTs. For example, a WLAN stack assumes a signal loss if multiple beacon frames, typically send periodically each 100 ms, fail to appear and if a subsequent active probing attempt fails. Afterwards, the signal loss event triggers the removal of the network configuration, which is in turn signaled to the MPTCP stack that triggers the transmission of the REMOVE_ADDR option to the server. Note that the option REMOVE_ADDR is not mandatory to be sent by a station, however, faster triggers for reinjection of packets are typically due to sender-site timer expiration.

This sequence of events already shows the slow reaction on data-layer events, where the removal takes several hundreds of milliseconds. Compared to the RTT of today's Internet and radio access networks, the time interval to detect the connection loss and reinject packets is far too large. Connections established via cellular and WLAN show RTTs often significantly below 200 ms [4], [5].

The loss of a data-layer connection can only be directly detected if it occurs at the access network of an end host. Loss of a connection due to errors on intermediate network segments between client and server cannot be detected directly since no information about those intermediate data-link layers is available at the end hosts [6]. Therefore, reinjection of lost packets on a broken connection need additional mechanisms to recover. Independent of the removal of a subflow, MPTCP reinjects packets after the retransmission timeout (RTO) of a subflow expires. Packets sent on that subflow, that are not acknowledged, are reinjected on remaining subflows. Hence, the reinjection of packets after RTO is independent of any data-layer events and does not lead to the removal of a subflow. Obviously, the order of the two events "RTO timer fires" and "transmission of the MPTCP option REMOVE_ADDR" depend on the setup. In the assumed scenario, where the RTT

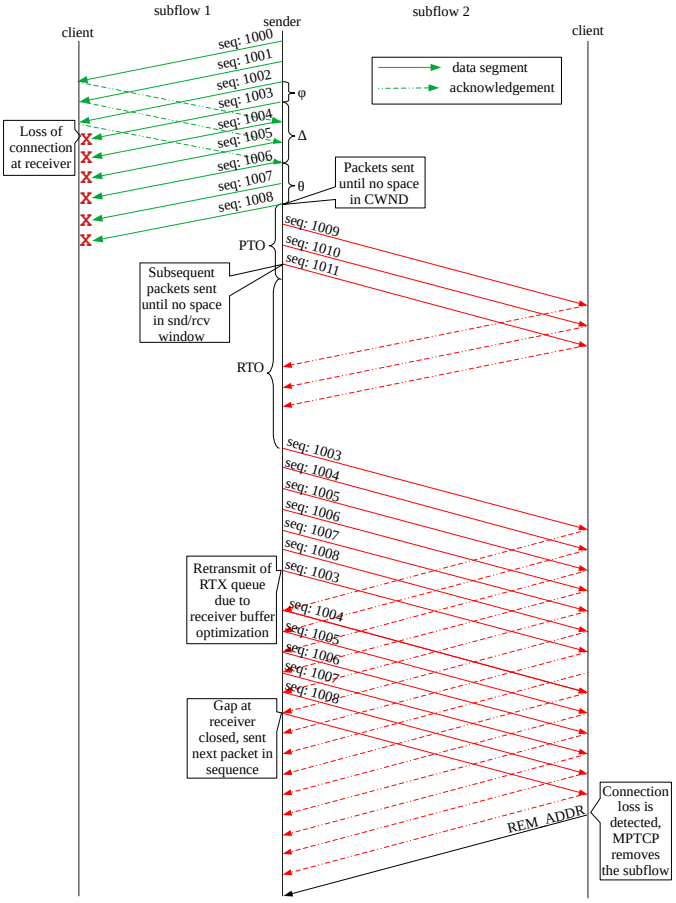


Fig. 1. Duration between connection loss of packet 1003 and following and reinjection of packets on the remaining subflow. For simplicity, the delayed acknowledgments algorithm is omitted and sequence numbers are assigned per segment not per byte.

is typically small and below the minimal RTO, the RTO fires first and leads to the reinjection of packets. So, the RTO timer is the main event that effects the reinjection of packets. Next, the sequence until the timer fires is described in more detail.

The retransmission timeout mechanism is transferred from SPTCP to MPTCP. The following analysis shows that the usage of the RTO prevents a fast reinjection of lost packets on remaining subflows. Furthermore, the extension tail-loss probe (TLP) [7], introduced in SPTCP to improve the retransmission behavior [8], slows down the reinjection of packets in MPTCP. The sequence of timer and events discussed next is depicted in Fig. 1.

The RTO is calculated as follows:

$$RTO = \max(RTO_{min}, SRTT + \max(G, K \cdot RTT_{VAR})), \quad (1)$$

where $SRTT$ is a smoothed RTT, G the clock granularity, RTO_{min} the minimal RTO, RTT_{VAR} the variance of RTT, and $K = 4$ a multiplication factor. RFC 6298 [9] states a minimum of one second for RTO_{min}. Linux uses an alternative implementation so that the RTO is

$$RTO = SRTT + \max(RTO_{min}, K \cdot RTT_{VAR}), \quad (2)$$

whereas $RTO_{min} = 200\ ms$. On each incoming ACK, the timer is reset to the current RTO estimate.

Although not explicitly depicted in Fig. 1, the last ACK that arrives at subflow 1 resets the RTO timer to the current RTO estimate. Note, that the first lost packet was sent Δ seconds before. If the packet interarrival time φ is small, $\Delta \approx RTT$ under the assumption of a small variance. That is, the reinjection of the lost packets takes

$$\begin{aligned} REJ_{RTO} &= \Delta + RTO \approx RTT + RTO \\ &= RTT + \max(RTO_{min}, SRTT + \max(G, K \cdot RTT_{VAR})) \end{aligned} \tag{3}$$

using Eq. (1) for RTO calculation. Until packets are transferred in sequence to the receiver, it takes REJ_{RTO} plus the one way delay from the sender to the receiver. For the assumed scenario, this implies a long pause in the transmission, even though further subflows may be available. If the RTT and its variance is small, the RTO is governed by RTO_{min}. The TCP feature TLP (which, in Linux, is enabled by default) even increases the time until packets are reinjected as depicted in Fig. 1. TLP avoids the RTO in SPTCP if packets at the end of a burst are lost, i.e. no further packets are in-flight that could trigger duplicate acknowledgements and so a fast retransmission. TLP is implemented by an additional timer. It retransmits the last sent packet after a probe timeout (PTO) of

$$PTO = \min(RTO, 2RTT + \delta) \tag{4}$$

to trigger acknowledgements. The parameter δ depends on operating systems setting and is typically 8 ms. If only one packet is in flight, to account for delayed acknowledgements $\delta = 200\ ms$. After the PTO fires, the RTO follows up. The PTO timer is activated if a first packet is sent on a previously empty connection, it is restarted on incoming ACKs and further packet transmissions. In the assumed scenario, the TLP is sent on the lost connection path first, therefore without any response. Afterwards, the RTO follows up, leading finally to the reinjection. Thereby, the time until packets are reinjected increases to

$$REJ_{TLP} = REJ_{RTO} + PTO + \theta \approx RTT + RTO + PTO + \theta, \tag{5}$$

if $RTO > 2RTT$ it is about $3RTT + RTO + \theta$. Note that the PTO timer is also restarted on each packet sent on a particular subflow. If the CWND and buffers are large, the sender may transmit packets for a while, so the timer is restarted until the sender keeps sending packets on the lost connection. This interval is denoted as θ.

Illustrated on Fig. 1, the first packet lost is the packet with sequence number 1003, the PTO timer is restarted after packet with sequence number 1008 is sent. The restransmission of the first lost occurs after $PTO + RTO$.

Additionally, receive buffer optimization implemented in MPTCPs MinRTT scheduler also causes a retransmission of queued packets in the retransmission queue, which holds the same data as the reinjection queue, as introduced as opportunistic retransmission in [10]. This retransmission occurs likely, since before reinjection, packets are sent until the congestion window or buffer limit stops the transmission.

Using connections with RTTs of 30 ms to 50 ms and little jitter, the time to reinject packets is multiple times larger than the RTT. The relation between the RTT and time to reinjection may even increase with upcoming technologies that include design goals for delay reduction such as 5G cellular networks without adaptation of RTO_{min}, since the relation between minimal timeout and RTTs worsen. Therefore, reliable multi-path protocols require a fast reinjection of packets.

To summarize the findings of this section:

- The minimal timeout RTO_{min} is large compared to path RTTs in radio access networks which causes slow reinjection of packets after a connection loss.
- The apposition of the timers PTO from tail loss probe and RTO increases the time until reinjection of packets.
- RTO and PTO are restarted on the last ACK arrived or last packet sent, respectively. This may be much later than the first packet loss.
- Receive buffer optimization in the MinRTT scheduler often leads to retransmission of previously reinjected packets.

III. Reinject Timer

In this section, a detection and reinjection strategy based on a new reinjection timer is proposed. Note, that this is a sender-side only extension without interaction with remaining subflows.

As discussed in Sec. II, the timers RTO_{min}, PTO, and the restart of the timers on each ACK, delay a fast reinjection. Furthermore, the RTO timer includes four times RTT_{VAR} to avoid unnecessary retransmission. The new reinjection therefore reduces the timer value setting by removal of RTO_{min}, PTO, and refers to the sending time of the packet not on the last received ACK:

$$REJ = SRTT + K \cdot RTT_{VAR} - \Delta \tag{6}$$

where K is an integer to account for multiple of RTT_{VAR}. Eq. (6) is similar to the RTOR algorithm proposed in [11] to take into account the elapsed time since the earliest outstanding packet. The term Δ is a correction factor from the last ACK received, which restart the timer. It corrects the timer to the transmission time of the first outstanding packet. In Fig. 1, the timer refers to the transmission time of packet 1003. Approximately, $\Delta \approx RTT$, as in Sec. II.

Equal to the RTO timer, the timer is set on the first packet sent if no packet is in flight and is restarted on each incoming acknowledgement. This also includes duplicate acknowledgements since these indicate that the connection is still available. The timer is canceled if no further packets are in flight.

The values are less conservative as the RTO timer value, since the RTO in SPTCP assumes that a timeout occurs in serious conditions, in which the load on the flow should be reduced. In MPTCP, reinjected packets are sent on the remaining subflows, so the timer can be less conservative,

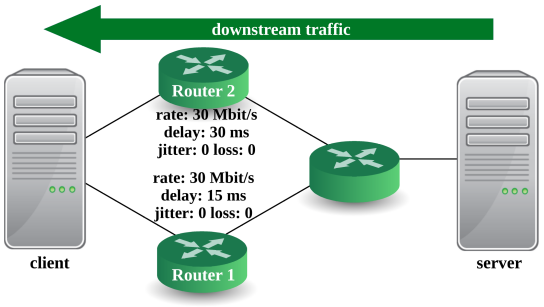
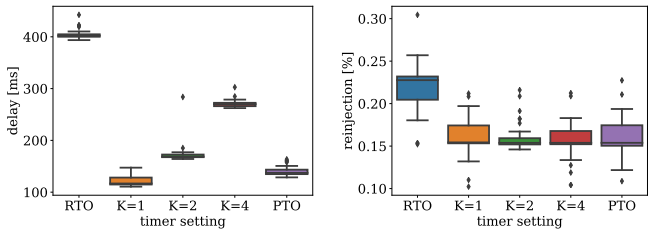


Fig. 2. Experimental setup: MPTCP connection between client and server with two subflows. The connection which has the smaller RTT is turned off by removing the IP configuration of the related interface on the client to emulate a connection loss.



(a) increase of delay (b) reinjection rate

Fig. 3. Comparison for $RTT << RTO_{min}$ for a non-greedy data rate: During the connection loss the delay increases for the original timer settings to about 400 ms and is reduced down to about 120 ms for $K = 1$. Also the number of reinjections decreases.

since it does not increase the load on the affected subflow. Still, the selection of the timer should not cause unnecessary reinjections.

IV. Experimental Evaluation

To evaluate the new approach for reinjection, different settings are evaluated in various scenarios. The first experiments are conducted in a controlled environment to evaluate the approach in basic setups with known parameters, whereas the last experiment demonstrates the benefits of the new reinjection timer in a real-world vehicular scenario.

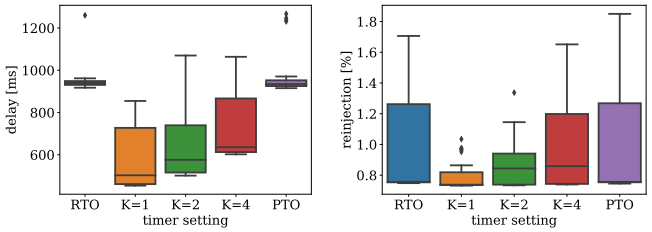
A. Experiment: On/Off Path

In this scenario, MPTCP establishes two subflows between the client and the server, one subflow per client interface. The network setup including emulated rates, delays, and jitter is shown in Fig. 2. All devices (client, server, and routers) are Linux Debian 10 based virtual machines using Kernel-based Virtual Machine for virtualization and Vagrant for deployment of the complete network topology. The routers use unmodified kernels, whereas the client and server use a MPTCP kernel based on version 4.19 and extensions for the additional reinjection timer as described in Sec. III. The buffer size at the routers is 200 packets for each interface. Constant bitrate traffic of 1800 pkt/s with an application layer payload size of 1400 byte is generated by the Distributed Traffic Internet Generator (D-ITG) [12], which is able to measure throughput and one way delays at the application level. Here, basic constant bitrate is used. On the one hand, the effect occurs independently of the traffic pattern, as long as data packets are in transmit on the path that is going to break, on the other hand, constant bitrate traffic resembles characteristics of real-time streaming application to be found in vehicular use cases. Packet traces are collected with `tcpdump` and reinjections of segments are extracted from packet traces by `mptcptrace` [13]. The default MPTCP scheduler is used, so the subflow with the minimal RTT is preferred.

In this experiment, the configuration of the IP address of the network interface at the client of the preferred flow is deleted and added (including route settings) repeatedly. This mimics a basic scenario in which a client leaves and enters a WLAN and stays connected to the cellular network permanently. This behavior corresponds to network managers that delete the IP and routing configuration when a client leaves the range of a WLAN, and adds it back when the device is within communication range again.

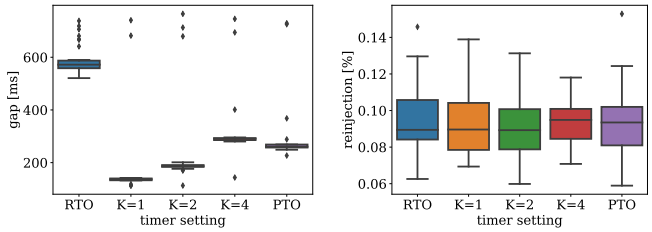
1) $RTT << RTO_{min}$: The first scenario emulates the setup assumed in Sec. III, where the RTT is significantly below 100 ms, i.e. below the default RTO_{min} value used in Linux. To focus on the effects after the loss of one connection, jitter and loss are neglected in this network setup. Fig. 3 shows the delay increase, i.e. the difference between the minimum and the maximum application delay during the loss event, and the reinjection rate. The reinjection rate is defined as the ratio of reinjected packets to the total number of packets sent over the network while the WLAN subflow is established. This period begins with the MPTCP option MP_JOIN on establishment and ends with the related REMOVE_ADDR option. So it accounts also for all reinjections while the WLAN connection is established.

Fig. 3 compares the default reinjection behavior (RTO), different timer settings according to Eq. (6) with $K = 1, 2, 4$, and reinjections based on the PTO timer as proposed in [14]. The client joins and leaves the WLAN 30 times for each timer setting and measure delay and reinjections. The delay increase and reinjection rate is shown in Fig. 3 as boxplots including the median, the 25th percentile ($q1$), the 75th percentile ($q3$), and the whiskers defined by the last measurement less than $q3 + 1.5(q1 - q3)$ and the first measurement greater than $q1 - 1.5(q1 - q3)$. Data outside this range is classified as outlier. The data clearly shows that the proposed reinjection strategy strongly reduces the delay at the connection loss for all timer settings. It is reduced to 120 ms with the most aggressive setting of $K = 1$. Moreover, the reinjection rate decreases too, indicating that a more aggressive reinjection strategy does not lead to suspicious reinjections. Furthermore, the reduction of the time to reinject also decreases the reinjection rate as indicated in Fig. 3b. At first view this is contradictory that a more aggressive reinjection strategy reduces the reinjections. These retransmissions are due to the opportunistic retransmission of the buffer optimization as explained in Sec.II. Since the reinjection is triggered faster, the buffers at the



(a) increase of delay (b) reinjection rate

Fig. 4. Comparison for $RTT > RTO_{min}$: During the connection loss the original MPTCP stack shows a large increase of the application level delays of about 1 s, our proposed approach speeds up the detection of the connection loss and thereby reduces the delay, without increasing the reinjection rate.



(a) packet gap (b) reinjection rate

Fig. 5. Comparison for $RTT << RTO_{min}$ for greedy data rate: Instead of the increase of the delay the maximum interarrival time of packets at the application layer is shown since the greedy traffic causes large delay due to buffering on the saturated links.

sender and receiver are not exhausted, so that the opportunistic retransmission is not triggered.

2) $RTT > 200\ ms$: In this scenario the RTT of the network is set above RTO_{min}, more precisely one path has an RTT of 250 ms, while the second path has 300 ms. Furthermore, the queue size of the routers is increased to 500 packets to account for the larger bandwidth-delay product. This way a single flow can achieve the application layer throughput of 20 MBit/s on a single-path. Fig. 4 shows the delay increase and reinjection rate of again 30 runs for each timer setting. When using an unmodified MPTCP implementation, the application level delay increases to about 1 s when the path with the lower RTT is lost. Our approach reduces the delay increase to about 500 ms in the median and also lowers the reinjection rate. However, the PTO timer does not show any benefit here. Since it equals the RTO if $2RTT > RTO$ as given in Eq. (4) and the REMOVE_ADDR option arrives in this scenario at the server in similar time range both approaches perform similar.

3) Experiment: On/Off Path - Greedy Traffic: In this setup, we use the same parameters as described in Sec. IV-A1, except for the application data rate, which is increased to 78 MBit/s. This way it exceeds the aggregated throughput of both paths and saturates the links at all times.

The saturation of the paths by the greedy traffic flow produces a large application level delay of about 1 s. This masks the gap between packet arrivals at the application layer, therefore the largest interarrival time between packets at the application layer is evaluated. Nevertheless, also in this scenario the slow detection of the connection loss generates significant gaps between packet arrivals at the application layer as shown in Fig. 5. The gap reduces from about 600 ms to 120 ms for the new reinjection timer with $K = 1$ and increases to about 300 ms for $K = 4$. These settings still halves the gap between packet interarrival times. The reinjection rate for all variants is similar.

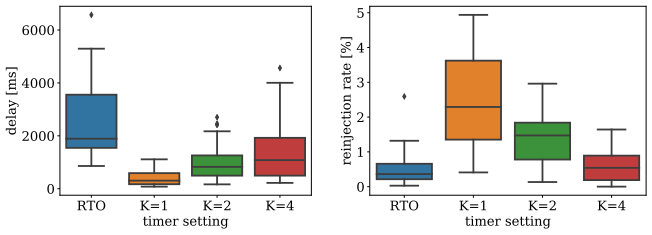
B. Vehicular Experiment

This experiment evaluates the proposed timer in a real-world scenario with high mobility that results in a large variability of the WLAN connection. The previous experiments show that our approach performs well in a well-controlled environment. However, complex signal characteristics in mobile scenarios with varying signal quality are difficult to emulate.

Therefore, in this scenario, we equipped a vehicle with a cellular interface as well as a WLAN interface and enter and leave the communication range of the WLAN network repeatedly with a speed of about 30 km/h. To achieve similar network conditions, we drive along a circular route of about two kilometers, in which for about 200 m a connection to the same WLAN network can be established. Note that, this experiment does not guarantee equal conditions as the emulated experiments presented in previous section due to e.g. signal conditions of further WLAN networks around and vehicle speed variations due to traffic. For the following statistical analysis, 30 measurements, i.e. connection losses, are recorded for each setting.

In detail, the cellular modem is a LTE category 12 module with UMTS fallback and the WLAN adapter supports IEEE 802.11ac. Both adapters use two antennas, that are mounted on the vehicle roof. A WLAN access point is mounted outdoors using an 80 MHz channel at the center frequency of 5540 MHz. We use Ofono [15] and ConnMan [16] as modem and connection manager, respectively. We modified the WLAN scanning for a fast detection, i.e. the scanning for new WLAN networks is performed each second if no network is connected, and ranges between three and ten seconds if a WLAN network is connected in dependence on signal quality. Since for our setup, the WLAN connection and cellular connection have roughly the same RTT, often the cellular connection is preferred using a MinRTT scheduling. Therefore, we use for the presented experiments the round-robin scheduler to enforce data packets on both paths. In a productive setup the MinRTT scheduler may be a better choice, still, the problem of slow reinjections remains as long as a packet is in transmit on the lost connection. We use a non-greedy application with a 10 MBit/s application layer throughput as in Sec. IV-A1.

The increase in delay and the reinjection rates are presented in Fig. 6. In this highly volatile experimental setup, the increase in application level delay is about 2 s and can be reduced by a fast connection loss detection. Our approach



(a) delay increase (b) reinjection rate

Fig. 6. Increase of delay and reinjection rate in the vehicular scenario: Without a dedicated reinjection timer delay increases by about 2 s in the median, a reinjection timer reduces this significantly but increases the reinjections in this volatile environment. Setting $K = 2$ and $K = 4$ shows a good tradeoff.

lowers the increase of 1.87 s to 0.28 s in the median for $K = 1$, an improvement of more than 80%. However, compared to the previous experiments, the short reinjection timer goes along with a higher reinjection rate of 2.28%. Nevertheless, setting $K = 2$ or $K = 4$ mitigates this increase and also lowers the delay to 0.79 s and 1.05 s, respectively, as a tradeoff between delay increase and reinjections.

V. Related Work

MPTCP is deployed and analyzed in various mobile scenarios. In [17], its is shown that MPTCP suffers from many handovers that let the queue size of the out-of-order queue increase. The authors also recommend a fast reinjection of lost packets. Further an in-depth analysis of MPTCP in mobile environments is executed in [18]. The results also indicate a conservative reinjection strategy for MPTCP and propose a smart reinjection based on timeouts and utilization of receive window, signaled by receiving acknowledgements. This prototype addresses single packet loss, but not the connection loss due to received acknowledgements. Especially, the tail loss is also improved by a redundant loss probe in [19].

According to [6], the major trigger for reinjections is the timer to reinject non-acknowledged packets on existing paths. We observed the same behavior and confirm that the second reinjection based on link-layer events is too slow. The authors also conclude that a sophisticated timer setting is required and propose a new MPTCP option to set the RTO timer to detect underperforming subflows. The timer dates back on a TCP option defined for SPTCP in [20]. Also in [6], the MPTCP backup mode is considered, where a primary subflow transmits the data and a backup flow takes over after the primary flow fails. In [1], the impact of the timer setting and small outages during a handover between WLAN and cellular connections for concurrent transmissions are analyzed. This work contributes additional measurements and proposes a new reinjection timer. In SPTCP, this timer only decides on the retransmission of packets, but already in SPTCP the retransmission due to expiration of the RTO is avoided and lost packets are detected by duplicate acknowledgements. Many approaches propose improvements to the fast detection and retransmission of lost packets and avoid the reaction of the RTO [21], [7], or propose the reduction of the RTO [22], [11]. In [14], the problem of lost packets at the end of a data burst for MPTCP is evaluated and packet reinjection at the PTO is proposed. Moreover, the improvement for cloud applications is experimentally evaluated in [19]. Similar to our work, the time to reinject packets is shortened, but instead the PTO timer is reused for reinjection. The reduction of the RTO timer in connections with short path RTTs is also discussed in [23] in conjunction with delayed acknowledgements for single packet losses. The scenarios considered in [14], [23] mainly focus on packet losses, whereas in our work, connection losses are addressed. Nevertheless, the reduction of the time to reinject packets is beneficial in both cases.

VI. Conclusion

The sudden loss of a network connection leads to the reinjection of packets in flight. Currently, MPTCP reuses the RTO calculation from SPTCP to trigger reinjections. However, this RTO is chosen conservatively to not stress networks that are already overloaded, a strategy well suited for SPTCP. We showed through emulated and real-world experiments, that the adaptation of the RTO causes large delays in multi-path setups, if a connection is lost. We propose to use a new timer to lower application level delays that arise at the loss of a connection due to a too conservative RTO setting. We observed benefits of up to 80% in scenarios with small and large RTTs and low jitter as well as in a volatile vehicular scenario. In setups with little jitter, the delay as well as the number of reinjections are significantly lowered. In the volatile scenario, the timer setting is a tradeoff between the increase of delay and a slight increase in data rate by the number of reinjections. Therefore, a new retransmission method improves the continuity of the transmission for intermittent connections.

References

[1] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, "Exploring Mobile/WiFi Handover with Multipath TCP," in *Proc. ACM CellNet*, 2012, pp. 31–36.

[2] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control," RFC 5681, Sep. 2009.

[3] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 8684, Mar. 2020.

[4] M. Trevisan, D. Giordano, I. Drago, M. M. Munafò, and M. Mellia, "Five Years at the Edge: Watching Internet From the ISP Network," 2020.

[5] T. Høiland-Jørgensen, B. Ahlgren, P. Hurtig, and A. Brunstrom, "Measuring Latency Variation in the Internet," in *ACM CoNEXT*, ser. CoNEXT '16, New York, NY, USA, 2016, pp. 473–480.

[6] O. Bonaventure, Q. D. Coninck, M. Baerts, F. Duchene, and B. Hesmans, "Improving Multipath TCP Backup Subflows," Internet Engineering Task Force, Internet-Draft, Jul. 2015.

[7] Y. Cheng, N. Cardwell, N. Dukkipati, and P. Jha, "Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses," Internet-Draft, Feb. 2013.

[8] M. Rajiullah, P. Hurtig, A. Brunstrom, A. Petlund, and M. Welzl, "An Evaluation of Tail Loss Recovery Mechanisms for TCP," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 1, pp. 5–11, Jan. 2015.

[9] M. Sargent, J. Chu, D. V. Paxson, and M. Allman, "Computing TCP's Retransmission Timer," RFC 6298, Jun. 2011.

[10] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *Proc. USENIX NSDI*, San Jose, CA, Apr. 2012, pp. 399–412.
[11] P. Hurtig, A. Brunstrom, A. Petlund, and M. Welzl, "TCP and Stream Control Transmission Protocol (SCTP) RTO Restart," RFC 7765, Feb. 2016.
[12] A. Botta, A. Dainotti, and A. Pescapé, "A Tool for the Generation of Realistic Network Workload for Emerging Networking Scenarios," *Comput. Netw.*, vol. 56, no. 15, pp. 3531–3547, Oct. 2012.
[13] B. Hesmans and O. Bonaventure, "Tracing Multipath TCP Connections," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 361–362, Aug. 2014.
[14] K. Yedugundla, P. Hurtig, and A. Brunstrom, "Probe or wait: Handling tail losses using Multipath TCP," in *Proc. IFIP Networking*, 2017.
[15] "oFono," https://01.org/ofono.
[16] "ConnMan," https://01.org/connman.
[17] L. Li, K. Xu, T. Li, K. Zheng, C. Peng, D. Wang, X. Wang, M. Shen, and R. Mijumbi, "A Measurement Study on Multi-Path TCP with Multiple Cellular Carriers on High Speed Rails," in *Proc. ACM Sigcomm*, 2018, pp. 161–175.
[18] A. Nikravesh, Y. Guo, F. Qian, Z. M. Mao, and S. Sen, "An In-Depth Understanding of Multipath TCP on Mobile Devices: Measurement and System Design," in *Proc. ACM Mobicom*, 2016, pp. 189–201.
[19] V. Yedugundla, P. Hurtig, and A. Brunstrom, "Handling Packet Losses in Cloud-based MPTCP Application Traffic," in *Proc. CLOSER*, 01 2019, pp. 111–119.
[20] F. Gont and L. Eggert, "TCP User Timeout Option," RFC 5482, Mar. 2009.
[21] Y. Cheng, N. Cardwell, N. Dukkipati, and P. Jha, "RACK: A time-based fast loss detection algorithm for TCP," Internet Engineering Task Force, Internet-Draft, Mar. 2020.
[22] W. Wang, N. Cardwell, Y. Cheng, and E. Dumazet, "TCP Low Latency Option," Internet Engineering Task Force, Internet-Draft, Jun. 2017.
[23] Ming Li, A. Lukyanenko, S. Tarkoma, and A. Ylä-Jääski, "The Delayed ACK evolution in MPTCP," in *Proc. IEEE GLOBECOM*, 2013, pp. 2282–2288.