Work in Progress: Fast Reinjection for Intermittent Multi-Path Connections

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

***Abstract*—A common use case for multi-path protocols such as mutli-path TCP is the simultaneous use of Wifi and cellular connections since smartphones nowadays provide such interfaces.**

The combination of multiple networks for Internet access, e.g. a local area network and wide area network, allows for continuous data transmission even in scenarios with an intermittent connection. One example is the loss of the Wifi connection in a mobile scenario in which a user leaves the connection range of the Wifi network. For a seamless handover between the connections, protocols have to detect the connection loss and have to retransmit lost packets of the lossed connection immediately, so that the receiving network stack can forward the packets in sequence to the application.

Here, the behavior for detection of connection loss and the reinjection of lost packets after connection loss is carved out for multi-path TCP. The analysis of the protocol behavior shows that mechanisms that originate from single-path TCP and adopted to multi-path are slow to detect connection loss and cause therefore large application level delays. Furthermore, a new reinjection timer is proposed to speed up the reinjection of lost packets after a connection loss. The new approach is evaluated using different timer values and show that adaptation of the timer speed the recovery of lost packets after a connection loss.

I. MPTCP Connection Loss Detection

For the following description a mobile device connected to a server using MPTCP with two subflows, one using Wifi and one using a cellular connection, is assumed. Both subflows are established, but only the Wifi connection is used due to application limited traffic, e.g. a video stream. MPTCP scheduling is based on the minimal RTT of the connection, and the path manager is full-mesh. Nevertheless 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.

The detection of the loss of a connection can be identified directly by data layer events or indirectly by probing and timeouts, i.e. lack of acknowledgements. For example, if the connection loss is due to leaving the communication range of a wireless network, an end-host directly learns about the loss from data layer events. Afterwards, a connection manager typically removes the configuration of related interfaces if the loss is indicated. In MPTCP, this triggers the removal of the related subflows. In a scenario in which the Wifi connection is usually at the client site and traffic flows mainly in downstream direction from a server to the client, the server learns about the loss of the connection by the client. In MPTCP, the client signals the loss by the MPTCP option REMOVE_ADDR to the server on remaining subflows, afterwards the server should trigger a TCP keepalive for security purposes, see RFC6824. If the keepalive messages is not answered, the subflow is assumed to be broken. The detection of the Wifi signal loss and transmission of the signal requires often multiple times of the RTT of the subflow connection. For example, a Wifi networking stack assumes a signal loss if multiple beacon frames, typically send periodically each 100 ms, fail to appear[1] 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 triggering the transmission of the REMOVE_ADDR option to the server. This sequence of events already show the slow reaction on data layer events, whereas the removal takes several hundreds milliseconds. Compared to the RTT of today's Internet connections, established via cellular and local area networks with RTTs often below 200 ms, see [1], [2], the time interval to detect the connection loss is large. The loss of a data layer connection can be directly detected if it occurs at an end host. Loss of a connection on intermediate network segments cannot be detected directly since no information from the data link layers is available at the end hosts. Therefore, reinjection of lost packets on broken connection needs an additional mechanisms to recover. Independent from the removal of a subflow, MPTCP reinjects packets after the retransmission timeout (RTO) of a subflow expires. Packets sent on that subflow but not acknowledged are reinjected on remaining subflows.

Therefore, the reinjection of packets after RTO is independent of any data layer event and does not lead to the removal of the subflow. Obviously, the order of the events, RTO timer fires and transmission of the MPTCP option REMOVE_ADDR, depend on the setup. In the assumed scenario, where the RTT is typically small and below the minimal RTO, the RTO fires

[1]https://github.com/multipath-tcp/mptcp/blob/e1efcf0a63f9b62394458e48f6d0ade15c8b8155/net/mac80211/mlme.c#L65

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 single-path TCP 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) [3], introduced in single-path TCP to improve the retransmission behavior [4], slows down the reinjection of packets in MPTCP.

The timers RTO and PTO for TLP are calculated as follows: TCP introduces a minimum value for the RTO timer. The RFC 6298 [5] states a minimum of one second. In general, the RTO is calculated by

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

with smoothed RTT $SRTT$, the clock granularity G, the minimal RTO RTO_{min}, the variance of RTT RTT_{VAR} and a multiplication factor of $K = 4$. Linux use a alternative implementation[2] so that the RTO is about

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

whereas $RTO_{min} = 200\ ms$.

On each incoming ACK, the timer is reset to the current RTO estimate.

The last ACK that arrives on subflow 1 restarts the RTO timer to the current RTO estimate. Note that, the first packet lost, is sent Δ seconds before. If the packet inter arrival time φ is small, $\Delta \approx RTT$ under the assumption of a small variance. That is, the reinjection of the lost packets takes about

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

Until packets are transferred in sequence to the receiver, it takes REJ_{RTO} plus the one way delay from the sender to the receiver to successfully transmit packets in sequence. For the assumed scenario, this implies a long pause in transmitting data, even though further subflows may be available. If the RTT and its variance is small, the RTO is governed by RTO_{min}.

The feature tail-loss probe[3] even let the time until packets are reinjected increase as depicted in Fig. 1. TLP avoids the retransmission timeout in single-path TCP 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 retransmit. TLP is implemented by an additional timer. It retransmits the last packet sent after a probe timeout of $PTO = \min(RTO, 2RTT + \delta)$ to trigger acknowledgements. The parameter δ depends on operating systems

[2]https://github.com/multipath-tcp/mptcp/blob/c7fa07ab914c20dbde486d7c80fa72ff78d2d4d2/net/ipv4/tcp_input.c#L721

[3]TLP is enabled by default on recent Linux distributions, implementation see https://github.com/multipath-tcp/mptcp/blob/c7fa07ab914c20dbde486d7c80fa72ff78d2d4d2/net/ipv4/tcp_output.c#L2464

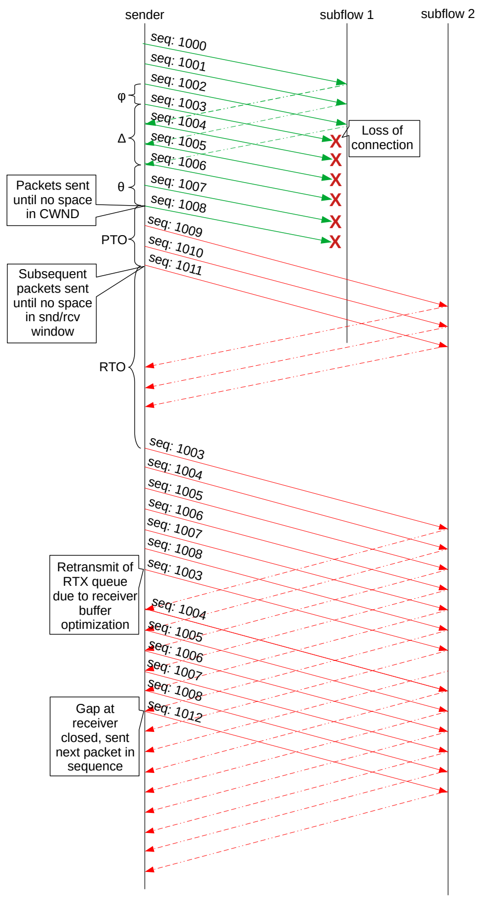

sender
subflow 1
subflow 2
seq: 1000
seq: 1001
seq: 1002
seq: 1003
seq: 1004
seq: 1005
seq: 1006
seq: 1007
seq: 1008
seq: 1009
seq: 1010
seq: 1011
φ
Δ
θ
PTO
RTO
Loss of connection
Packets sent until no space in CWND
Subsequent packets sent until no space in snd/rcv window
Retransmit of RTX queue due to receiver buffer optimization
Gap at receiver closed, sent next packet in sequence
seq: 1012

Fig. 1. Duration between connection loss and reinjection of packets on the remaining path with tail loss probing and receive buffer optimization, delayed acknowledgments algorithm is omitted for simplicity.

setting, $\delta = TCP\_MIN\_TIMEOUT = 2\ jiffies$ or $\delta = TCP\_RTO\_MIN = HZ/5$ if only one packet is in flight, typical values are $HZ = 250\ Hz$, whereas a jiffie becomes $1/HZ = 4\ ms$. 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

$$REJ_{TLP} = REJ_{RTO} + PTO + \theta \approx RTT + RTO + PTO + \theta, \tag{4}$$

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 expressed by θ.

Additionally, receive buffer optimization implemented in the

MinRTT scheduler[4] also causes a retransmission of packets queued in the retransmission queue of the MPTCP meta socket. The retransmission queue holds the same packets as reinjected before. This retransmission occurs likely, since before reinjection, packets are sent until the congestion window or buffer limit stops the transmission. The sequence is shown in Fig. 1.

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 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 multipath 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 common internet access networks that 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 the first packet lost.
- Receive buffer optimization in the MinRTT scheduler leads to retransmission of previously reinjected packets.

II. Example

The extraction of packets from a packet trace in Tab. I demonstrates the timing and sequences related to the events described in Sec. I. The packets trace is recorded in experimental setup shown in Sec. IV-A. The values relate the exemplary flow from Fig. 1 to realistic timing, sequence numbers, and extent of packets sent after the loss of the connection.

In the present packet trace it takes 404 ms from the first packet that is not acknowledged and has to be retransmitted until the reinjection of this packet. The REMOVE_ADDR option arrives at 569 ms later. After the first packet not acknowledged still 230 kByte are sent on subflow 1 and 485 kByte on subflow 2.

III. Detection and Reinject Strategy

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

For evaluation three alternative values are calculated for the reinjection timer (REJ). The first timer is similar to the RTO timer but the factor K is set to $K = 1$ and no RTT_{min} is taken into account:

$$REJ = SRTT + RTT_{VAR}. \tag{5}$$

The second value uses as estimate the minimal RTT compared to the previous timer:

$$REJ = RTT_{min} + RTT_{VAR}. \tag{6}$$

[4]https://github.com/multipath-tcp/mptcp/blob/c7fa07ab914c20dbde486d7c80fa72ff78d2d4d2/net/ipv4/tcp_output.c#L2464

The last value adjust the timeout based on the send time of the first packet on the retransmission queue, for this adjustment Δ see Eq. (3):

$$REJ = SRTT + RTT_{VAR} - \Delta \tag{7}$$

This last calculation is similar to the RTOR algorithm proposed in [6] to take into account the elapsed time since the earliest outstanding packet.

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

For evaluation purposes and to avoid modifications on single-path TCP behavior, a new timer is implemented. In Linux, the minimal RTO values can be adjusted, TLP can be deactivated, and the RTOR algorithm [6] could be implemented to achieve a similar behavior without the requirement for a new timer.

A. Reinjection and MPTCP RCV Buffer Optimization

The new implementation does not address the retransmission of segments reinjected before. The main reason for this reinjection is that MPTCP does not remember the subflow the reinjected segments are sent on. Typically, segments are reinjected or retransmitted by recv buffer optimization if not already sent on the subflow before, but it only remembers the original subflow. A basic solution would be to let MPTCP to store all subflows a segment (including retransmission and reinjections) was sent on. Nevertheless, implementation must taken into account to not lead to a stall, if no additional subflow is available if a segment was already sent on all subflows.

B. Reinjection and Duplication Acknowledgements

The described approach resets the timer reinjection timer also on duplication acknowledgements to avoid reinjection in this case. Still, if duplicated ACKs arrive it is difficult to decide if the retransmission on the remaining path or the reinjection on further paths is beneficial. Therefore, different strategies could be implemented: First, only reinject segments without retransmission if a second subflow is available, second, do a retransmission as well as a reinjection and duplicate segments. In both cases also information from selective acknowledgments can be used for a selective retransmission or reinjection, respectively.

First experiments to also reinject packets during recovery by duplicate acknowledgments show an increase of the send buffer size that leads to a stall of one subflow due to size

Event	sf	Time	TCP Seq	TCP Ack	DSN	Cause
First packet not acked	1	75.825676 (0)	60699801	x	3616571178	Connection broke
Last ACK	1	75.856112 (+30)	x	60701201	x	
Last packet sent	1	75.918575 (+93)	60935001	x	3616803578	CWND limited
First packet	2	75.919129 (+94)	63477569	x	3616804978	1. sf becomes CWND limited, 2. sf is used
Retransmission (PTO)	1	75.993254 (+168)	60935001	x	3616804978	PTO fires on 1. sf, no effect due to broken link
Last packet before retrans	2	76.116669 (+198)	63974569	x	3617303378	Buffer limitation, sender and receiver cannot clear their buffer (HoL at receiver, RTX queue at sender), 2. sf stops sending packets
Retransmission (RTO)	1	76.229354 (+404)	60701201	x	3616571178	RTO timer on sf 1 triggers retransmission
Reinject (RTO)	2	76.229603 (+404)	63975969	x	3616571178	RTO timer on sf 1 triggers reinjection on sf 2
Reinject RTX queue	2	76.251669 (+426)	64211169	x	3616571178	Due to buffer constraints, receive buffer optimization triggers, retransmission of RTX queue, which holds the same packets as reinjected before
REMOVE_ADDR option arrived	2	76.394444 (+569)	x	64297969	3617338378	Server receives REMOVE_ADDR option and closes sf, no further effect on retransmission

TABLE I
EXTRACT FROM PACKET TRACE, RELATED TO THE SECOND CONNECTION LOSS IN FIG. 3A.

limitation of the send buffer[5]. This effect is left for further evaluation, here, the evaluation focuses on connection losses. Due to the lost connection, the related subflow cannot receive any duplicated ACKs.

IV. Scenarios

To evaluate the approach the new timer is evaluated in various scenarios to measure parameters as the application level delay after a path loss and the number of reinjections to identify if the reduction of the timer causes additional reinjections. Therefore various experiments are performed:

- Done:
  - Network:
    - Basic: constant rate, loss of one connection
      - Application: Greedy, Non-Greedy,Thin streams (delayed ack behavior)
    - Complex: packet loss and jitter during handover
- ToDo:
  - Realistic: Emulation with varying throughput and loss before connection loss
  - Network: Real-world: WLAN/LTE

A. *Experiment: On/Off Path*

In this scenario, MPTCP establishes two links between the client and server. The network setup including rate, delay, and jitter settings is shown in Fig. 2. Using the default MPTCP scheduler the subflow with the minimal RTT is preferred. Here, this makes the subflow along Router1, which path features the smaller RTT, the preferred one. For the 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 a Wifi network and stays connected to a

[5]In detail, packets are not retransmitted due to NOMEM in `tcp_fragment()` in https://github.com/multipath-tcp/mptcp/blob/a289cca68246e5360287154b19f6d1aa8fa73a80/net/ipv4/tcp_output.c#L1338, nevertheless this behavior may improve in more recent kernel versions https://github.com/multipath-tcp/mptcp/blob/c7fa07ab914c20dbde486d7c80fa72ff78d2d4d2/net/ipv4/tcp_output.c#L1327

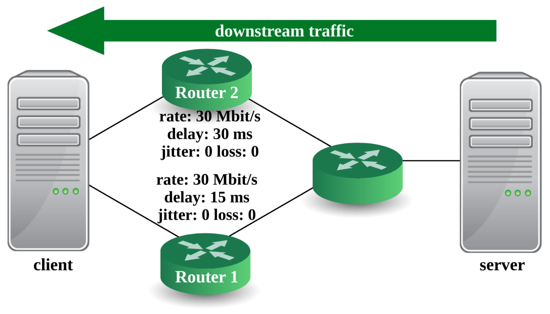


Fig. 2. Experimental setup: Multi-path connection between client and server. The connection which has the smaller RTT is turn off by removing the IP configuration of the related interface on the client to emulate a connection loss.

cellular network. This behavior corresponds to network managers that delete network configuration if the communication range of a Wifi network is left and adding the configuration of IP address and routing setting if the communication range is entered again. All devices (client, server, and routers) are Linux, Debian 10, based virtual machines using Kernel-based Virtual Machine for virtualization and Vagrant for deployment. The routers use unmodified kernels, where client and server use MPTCP kernel based on version 4.19 [6] and extensions for an additional reinjection timer as described in Sec. III. The buffer size at the routers is 200 packets for each interface. Constant bitrate traffic of 10 MBit/s with payload sizes of 1400 byte is generated by the Distributed Traffic Internet Generator (DITG) [7], which also allows for the measurement of throughput and one way delays. One way delays and application throughput is measured by DITG; estimates of the CWND, SRTT, packets in flight are extracted from the Linux TCP stack by the use of TCP's probing facility, network throughput is measured by analysis of packet traces collected with `tcpdump`, a reinjection of segments is extracted from packet traces by `mptcptrace` [8]. To focus on effects from connection loss and packet reinjection a simple constant bitrate traffic profile and basic path characteristic with constant delay

[6]Commit a289cca68246e5360287154b19f6d1aa8fa73a80

and without packet loss and jitter are chosen.

1) $RTT << RTO_{min}$: The first scenario mimics the scenario assumed in Sec. III where RTT is typically significantly below 100 ms, i.e. below default minimal RTO value used in Linux. To emphasize on the effects after the loss of one connection, complex parameters as jitter and loss are omitted in this network scenario. In Sec. IV-D, also results from experiments that include jitter and loss are presented.

The results in Fig. 3 show that the introduction of a new timer lowers the application level delay after the connection loss. The default behavior shows a application delay of more than 400 ms (connection loss occurs at 135 s, 185 s 235 s, 285 s, 335 s in all figures) in Fig. 3a, which is more than ten times larger than the largest one way path delay in this network scenario. For all alternative timer values proposed in Sec. III, the application level delay decreases. In Fig. 3d, it improves to about 125 ms.

2) $RTT > 200\ ms$: In this second scenario the RTT of the network is increased above the minimal RTT of 200 ms, the path with the lower RTT has a RTT of 250 ms and the second path of 300 ms. Also in this scenario the application level delays increases to about 1 s when the path with the lower RTT breaks using an unmodified MPTCP implementation, integration of an additional timer for reinjection decreases the delay, Fig. 4d shows an improvement to about 400 ms.

B. Experiment: On/Off Path - Greedy Traffic

This experiment is set to the same parameters as described in Sec. IV-A1 expected for the application data rate, which is set to 78 MBit/s. So it exceeds the aggregated throughput of both path.

Buffering of packets at the routers, sender, and receiver stacks lead to a application delay of about 1 s if only one path is available and is halved if the second path becomes active. Nevertheless, the application delay decreases after the connection loss from about 1 s to 650 ms, 600 ms, 550 ms for the three timer variants.

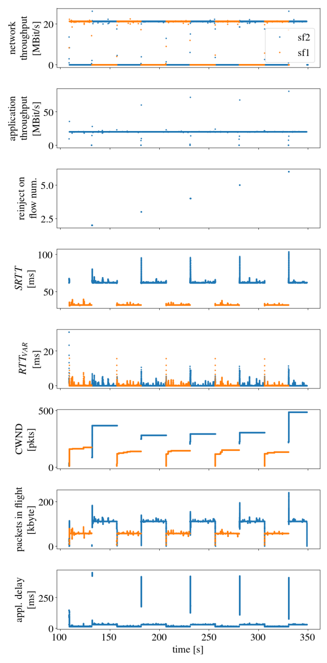
C. Experiment: On/Off Path - Thin Traffic

In this experiment the settings from Sec. IV-A are adapted: A thin data stream with 2 packets per seconds is configured. So the packet gap is 500 ms to evaluate if delay acknowledgments trigger suspicious reinjections. The related traffic trace does not show any suspicious reinjection besides reinjections after a connection loss. The application delay after a connection loss improves from 420 ms to about 110 ms.

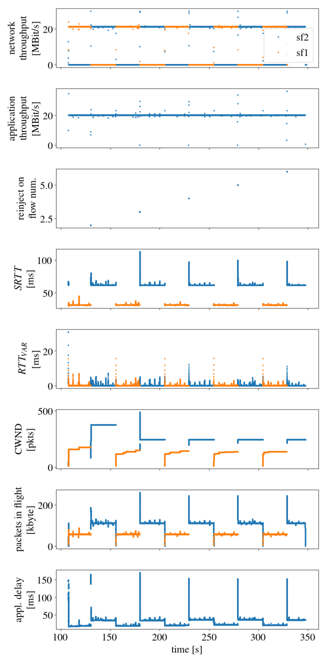
D. Experiment: Jitter & Loss

To evaluate if jitter and loss cause suspicious reinjections of packets, the experiment from Sec. IV-A1 is repeated with a jitter of 5 ms or 8 ms, respectively, on the paths with the small and large delays in each direction and loss of 0.01% on each path in each direction.

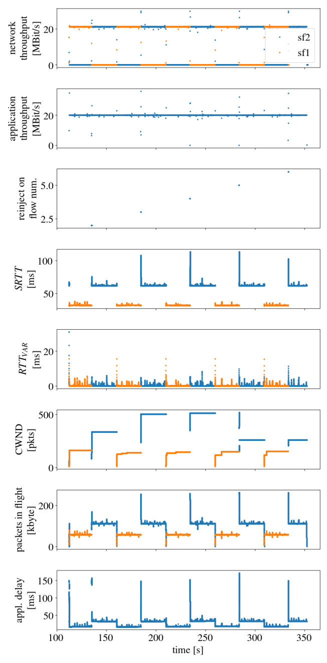
For comparison the original implementation and the new timer with settings defined in Eqs. (5),(6),(7) are evaluated. For each experiment the total number of reinjected packets



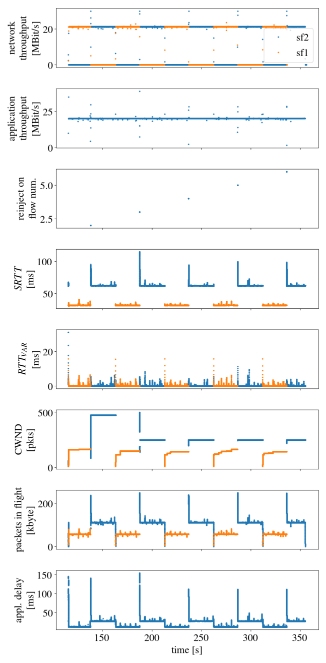
(a) oiginal



(b) $REJ = sRTT + RTT_{VAR}$



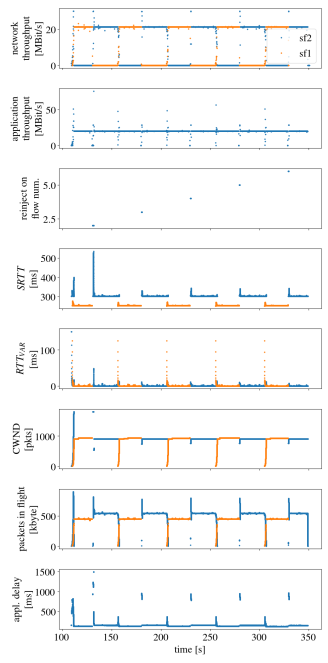
(c) $REJ = RTT_{min} + RTT_{VAR}$



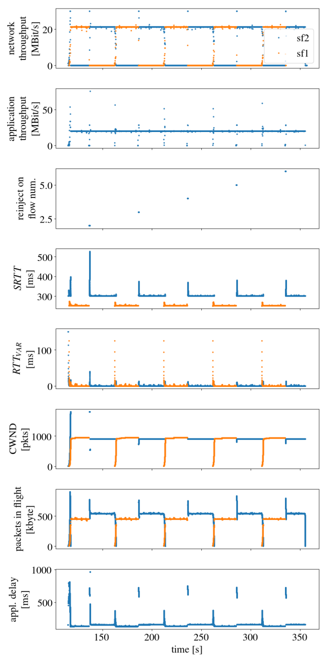
(d) $REJ = sRTT + RTT_{VAR} - \Delta$

Fig. 3. Basic network $RTT << RTO_{min}$ - Implementation of a new timer reduces the application level delay, the delay decreases from about 400 ms to less than 150 ms for timer setting presented in Eq. (7)

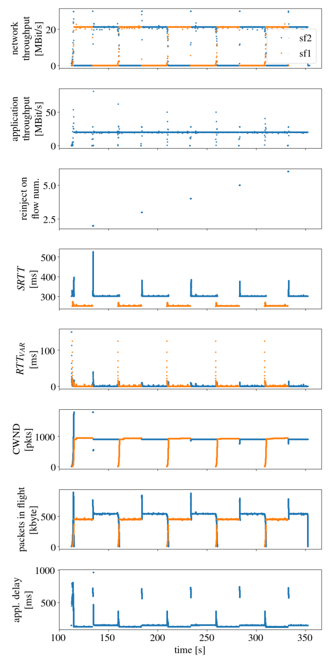
is related to the total number of application layer packets, which is denoted as retransmission rate. The retransmission rate is shown for each variant in Fig. 5. The experiment for each variant is repeated 30 times and summarized as boxplot. The results show a overall litte retransmission rate,



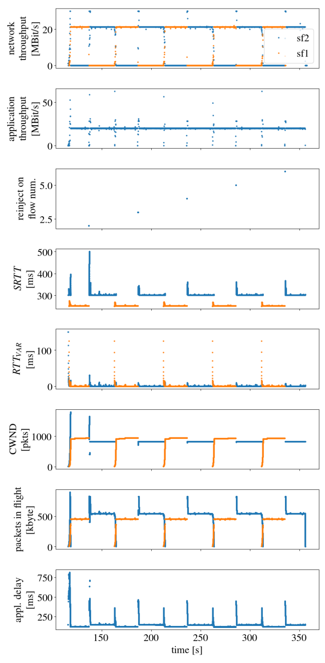
(a) unmodified



(b) $REJ = sRTT + RTT_{VAR}$



(c) $REJ = RTT_{min} + RTT_{VAR}$



(d) $REJ = RTT_{min} + RTT_{VAR}$

Fig. 4. Simple network $RTT > 200\ ms$ - Implementation of a new timer reduces the application level delay, the delay decreases from about 1000 ms to about 500 ms for timer setting presented in Eq. (7).

which increases from 0.094% to 0.112% for the timer setting given in Eq. (7). For the settings in Eqs. (5),(6), nearly no changes are observable in the median reinjection rate. From the results it follows that also in the appearance of jitter and loss the introduced reinjection timer is robust and does not

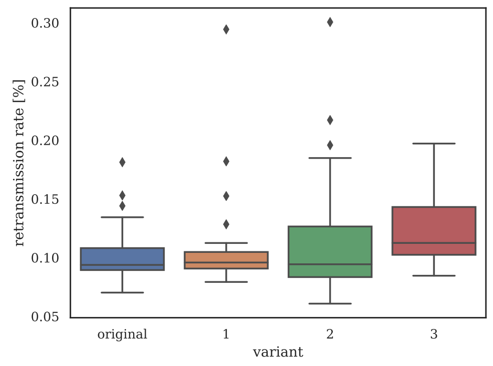


Fig. 5. Experimental setup: Multi-path connection between client and server. The connection which has the smaller RTT is turn off by removing the IP configuration of the related interface on the client to emulate a connection loss.

cause significant unnecessary reinjection of packets.

V. Conclusion

The sudden loss of a connection path leads to retransmission of in-flight packets. For fast reinjection of packets a rapid detection of the loss is essential. The in-depth analysis of the usual single-path TCP behavior for retransmission timouts, which is transferred to MPTCP to trigger reinjections, shows that the detection can last long due to minimal RTT timer values, tail loss probe, and timer activation.

The implementation of a new timer, which handles reinjection separately, shows a significant improvement in application level delays. Furthermore, experiments in scenarios with loss and jitter do not show a significant increase of suspicious retransmissions.

References

[1] M. Trevisan, D. Giordano, I. Drago, M. M. Munafò, and M. Mellia, "Five years at the edge: Watching internet from the isp network," *IEEE/ACM Trans. Netw.*, 2020.

[2] 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, p. 473–480. [Online]. Available: https://doi.org/10.1145/2999572.2999603

[3] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis, "Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses," Feb. 2013. [Online]. Available: https://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe-01

[4] 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, p. 5–11, Jan. 2015. [Online]. Available: https://doi.org/10.1145/2717646.2717648

[5] M. Sargent, J. Chu, D. V. Paxson, and M. Allman, "Computing TCP's Retransmission Timer," RFC 6298, Jun. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6298.txt

[6] P. Hurtig, A. Brunstrom, A. Petlund, and M. Welzl, "TCP and Stream Control Transmission Protocol (SCTP) RTO Restart," RFC 7765, Feb. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7765.txt

[7] 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.

[8] B. Hesmans and O. Bonaventure, "Tracing multipath tcp connections," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, p. 361–362, Aug. 2014. [Online]. Available: https://doi.org/10.1145/2740070.2631453