# RTTPROBS: A RTT Probing Scheme for RTT Aware Multi-path Scheduling

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

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

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

*Abstract*—**In multi-path scenarios, the default multi-path TCP (MPTCP) scheduler prefers the subflow with the minimum round-trip time (RTT) to send packets. Therefore, the scheduler requires reliable and timely information on the RTTs for the individual subflows to make proper scheduling decisions. We show for various situations that a reliable and timely RTT estimate is not always given. This shortcoming is induced from the original TCP behavior that the RTT is only updated if data is transmitted on a TCP flow. This means for MPTCP, that a subflow, which suffers from a short-term RTT increase, may not be selected again by the scheduler and so never gets an RTT update. Moreover, we demonstrate for various scenarios, that this leads to non-optimal scheduling decisions. We propose an RTT probing scheme to achieve reliable and timely RTT updates. Our probing scheme is designed for a fast reaction as well as little probing traffic. The advantages are demonstrated in emulated as well as real-world vehicular experiments.**

## I. Introduction

Due to the widespread deployment of devices with several network interfaces, like Smartphones, the usage of multi-path transport protocols become more and more expedient. Especially, MPTCP is already deployed for example in Apple's iOS and MacOS and is going to become prevalent in Linux based devices, such as Android Smartphone, by the recent integration of MPTCP in the mainline kernel. The concurrent use of multiple paths for data transmission offers improvements in data rate, reliability, and end-to-end latency. However, to exploit these benefits, a sophisticated scheduling decision has to be made for each packet. Current state-of-the-art transport layer multi-path approaches such as MPTCP, MPQUIC, or CMP-SCTP rely on information provided by individual subflows. Characteristics that are typically measured or estimated are the round-trip time and the congestion window to adapt the transmission of data to the characteristics on individual subflows in multi-path scenarios. The default scheduler in MPTCP and MPQUIC, for example, relies on RTT measurements for its scheduling decision. In detail, packets are scheduled on the subflow with the lowest RTT first until the subflow becomes unavailable, e.g. by an exhausted congestion window. Afterwards, the subflow with the second lowest RTT is selected.

This minimal RTT (minRTT) scheduler is a rational approach, since first the subflow with the lowest RTT is selected to guarantee transmissions with minimal latency. Only if the capacity of the subflow is not sufficient, additional subflows are used in the order of increasing RTT of the respective subflow. The RTT is a viable metric, as it does not only include the transmission and propagation delay, but also accounts for queuing delays and delays due to lower layer retransmissions. These are, for example, common in wireless networks with variable link quality. Therefore, the RTT metric also reflects the load and the reliability of the subflow. The default scheduler is mentioned in [1] and evaluated in [2]. Besides the default scheduler, various other schedulers rely on RTT estimates, e.g. [3], [4], [5], [6].

However, single-path TCP only measures the RTT if data packets are in flight, by measuring the time a packet sent until the related acknowledgment arrives. Therefore, if no data is sent, the RTT is not estimated. In single-path protocols, this is acceptable because there is no scheduling decision. Moreover, loss-based congestion protocols, such as NewReno and Cubic, do not consider the RTT for congestion control and as far as data is transmitted on the single-path, the RTT is updated.

Opposed to this, we contribute, that in multi-path scenarios an outdated RTT estimate leads to sub-optimal scheduling decisions. From observations in the scenarios, we deduce various criteria for a probing scheme to update the RTT on used subflows. We propose *RTTPROBS*, an RTT probing scheme that reacts fast on RTT changes but still avoids excessive probing traffic. The scheme is evaluated in various scenarios to measure the benefits and the overhead induced by probing.

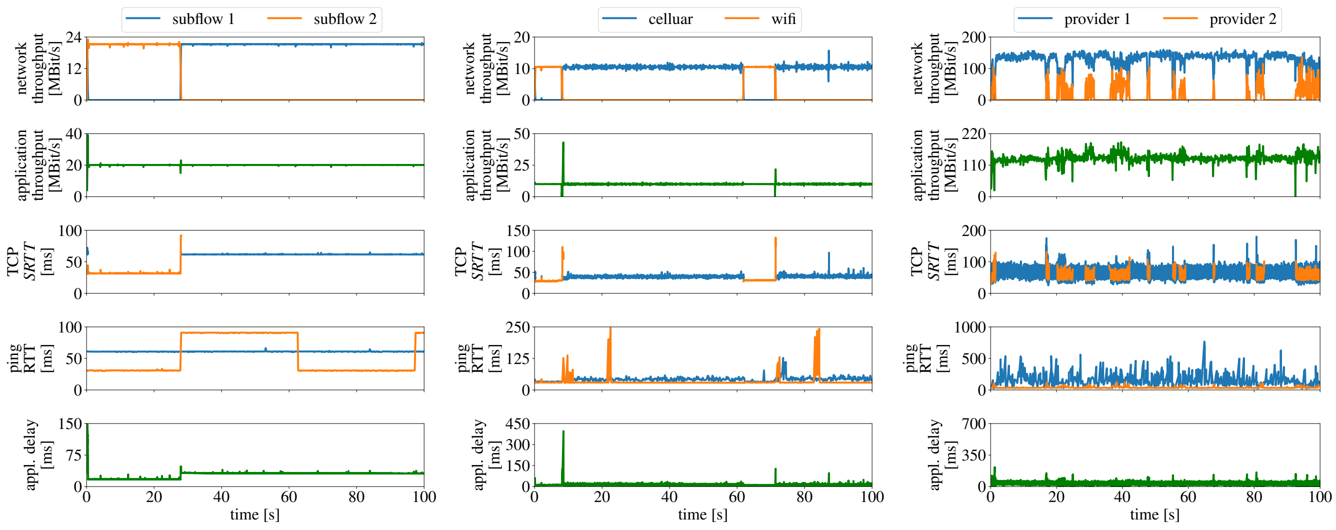
The outline of the paper is as follows: The single-path TCP estimation procedure and the related work is presented in Sec. II. Sec. III introduces scenarios in which an outdated RTT measurement occurs. Sec. IV presents RTTPROBS that is evaluated in Sec. V. Sec. VII concludes the findings.

## II. Related Work

Single-path TCP updates the RTT only if data is sent and related acknowledgments arrive at the sender. It uses an exponentially weighted average

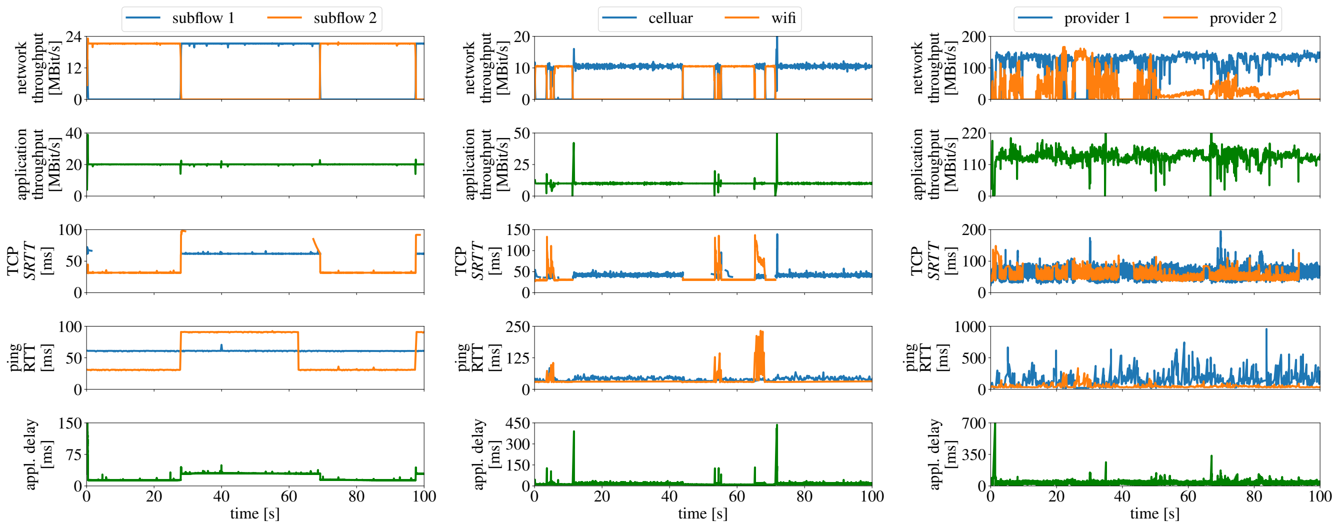
$$SRTT = (1 - \alpha)SRTT + \alpha CRTT, \qquad (1)$$

with the smoothed RTT SRTT, the current RTT measurement CRTT, and $\alpha = \frac{1}{8}$ as defined in [7]. CRTT values may be gathered from measuring the interval from sending data and reception of related acknowledgments according to Karn's algorithm [8] or estimated from TCP timestamps, see [9].



(a) Emulation (b) Wi-Fi Scanning (c) Bufferbloat

Fig. 1. Various examples of network scenarios in which an outdated RTT estimate leads to sub-optimal scheduling decision. In (a) the decrease of path RTT is not recognized, in (b) Wi-Fi scanning leads to short-term increasing RTT and causes permanent shift of data traffic, in (c) bufferbloat causes a strong RTT increase, which lead to outdated RTT measurements.



(a) Emulation (b) Wi-Fi Scanning (c) Bufferbloat

Fig. 2. Repetition of the scenarios of Fig. 1 with the implementation of RTTPROBS, which updates the RTT and improves the scheduling decision. In (a) the decrease after the long-lasting RTT is detected and enforce data traffic to the subflows with the lower RTT, in (b) the Wi-Fi path is used after the scanning process is finished, in (c) both subflows are utilized more equally.

Only updating the RTT on data transmissions is sufficient for single-path protocols. If no traffic is sent, the RTT requires no update. If traffic is sent, the RTT is updated immediately by arriving acknowledgments.

Nevertheless, this behavior changes in multi-path protocols if the RTT is included for scheduling decisions [3], [4], [5], [6] as well as congestion control algorithm decisions [10], [11], [12], [13], [14], [15]. The scheduler requires an up-to-date estimate for all active subflows for an optimal decision, even if no data is scheduled on the subflows. In current implementations, only subflows on which packets are scheduled receive updates of the RTT. A high RTT variation as prevalent in cellular networks, see [16], makes things worse.

The issue of an outdated RTT estimate is already raised in [17] for thin streams, e.g. streams that typically do not utilize all subflows of a multi-path connection. A scheme is proposed, that probes the subflows with constant multiplier of the RTT. Opposed to this work, we propose an adaptive scheme that accounts for short-term and long-term changes and reduces the probing traffic. Furthermore, we show that the underlying problem applies to further scenarios besides thin streams.

So far, single-path TCP only implements keepalives, but due to [18], the first keepalive is sent after 2 hours by default. Also keepalive packets and related acknowledgments do not update the SRTT.

## III. Scenarios

In this section, we evaluate three typical scenarios, in which an outdated RTT leads to sub-optimal scheduling decisions. Whereas the first scenario is an artificial setup in an emulated testbed to emphasize the problem, the later two are from real-world networks.

### A. Basic: Alternating RTT

In this first scenario, a simple network is emulated, in which one path has a constant RTT of 60 ms and the second path features an alternating RTT of 30 ms and 90 ms, respectively. Each subflow has a capacity of 30 MBit/s and the application data rate is tuned to be 20 MBit/s, i.e. the throughput is application limited, e.g., emulating a constant bitrate video streaming application. The application level values are measured by D-ITG [19], network layer throughput is measured from packet traces, TCP RTT is extracted from TCP's probing facility included in the Linux networking stack, and `ping` is used to obtain network level delays. Fig. 1a presents the application and network throughput, the RTT measurements performed by the TCP subflows and from a simultaneous `ping` measurement, and the application delay.

We observe, that at about 25 s, the RTT of the first subflow increases above the RTT of the second subflow. Consequently, the traffic is shifted to that second subflow. After the switch, the TCP SRTT is not updated for the first subflow, since no packets are scheduled on it. So the decrease of the RTT at about 60 s is not detected, although the first subflow would be the better choice now.

### B. Wi-Fi Scanning

In this real-world experiment, we create a static setup using one Wi-Fi and one cellular network, e.g. as in hybrid access networks [20]. We switch the Wi-Fi network on and off repeatedly, which triggers the deletion and creation of MPTCP subflows. The network level RTT measurements indicate that the Wi-Fi network is available from time 0 s to 25 s and 60 s to 90 s. We limit the application throughput to a packet rate of 1250 pkt/s with a packet size of 1000 bytes. The Wi-Fi hotspot is connected to the Internet using a digital subscriber line with a capacity of about 13 MBit/s and the cellular connection typically achieves a data rate of more than 100 Mbit/s in the downlink.

We observe in Fig. 1b, that the RTT is slightly smaller for the Wi-Fi connection in comparison to the cellular connection. But if the Wi-Fi device starts scanning for networks, it stops forwarding packets on the current active connection to switch the channel frequency. This causes an increase in the RTT, which is greater than the RTT experienced in the cellular network. Therefore, the scheduler selects the cellular network and does not return to the Wi-Fi network, which is indicated by the network throughput as well as the TCP SRTT measurement in the figure. After the first peak in the SRTT measurement the scheduler prefers the cellular connection and the SRTT measurement is not updated again, even if the RTT decreases as indicated by the `ping` measurements. The SRTT measurement shows peaks at about 10 s and 70 s, afterwards the subflow is not used anymore until the Wi-Fi network is toggled and a new subflow is created. The optimal behavior in this situation would be to use the Wi-Fi connection directly after its RTT decreases again.

### C. Bufferbloat

In this scenario, we target at the aggregation of two cellular connections, each features usually a capacity significantly above 100 MBit/s. Cellular networks typically deploy large buffers in the network, see e.g. [21], [16]. The experiments are executed in a static urban real-world cellular network with two independent cellular network providers. In [22], we already demonstrated that large buffers inside the network require large buffers at the sender and receiver for a full utilization of the subflows. If buffers are small, the performance is non-optimal, whereas the effect is even amplified by outdated RTT measurements. Fig. 1c presents the utilization of the two cellular networks. The application is greedy, but the throughput is limited by the buffer size setting at the sender and receiver[1]. The results indicate a preference of one provider, the second provider is only used sporadically. For example, at time 2 s and 43 s, the SRTT increases and the related subflow is not used for a notable interval. Furthermore, missing measurement points in the graph indicate, that no ACK arrives and the SRTT is not updated. Especially, this effect is visible at about 17 s, when the SRTT of the first provider increases, which leads to an update of the SRTT and the usage of the flow.

## IV. RTT Probing

The previous section indicates the need for a timely RTT estimate if scheduling decisions use RTT measurements. The main reasons for outdated and over-estimated measurements are bufferbloat, network jitter, or scanning in Wi-Fi networks.

Since TCP only updates the RTT estimate if an ACK of previously sent data is received, a subflow never receives an update without data transmitted. Therefore, a probing scheme is required to enforce an update in multi-path protocols. As carved out in the previous section, multiple reasons are responsible for outdated measurements that can last for a short or long duration. Hence, a probing approach has to account for these different timescales. For example, when to probe a subflow depends on various parameters: The extent of bufferbloat at intermediary devices, the ratio of RTTs of the available subflows, the link capacity of the overloaded segment, or the decline of the short-term increase of delays in wireless networks. As we show, these are due to network operations such as scanning for available networks or handover between different cells in wireless networks. These volatile characteristics require a probing that accounts for short-term increase as well as long-lasting increase of delay.

Furthermore, TCP implements an exponential weighted averaging for RTT estimates, so that a single estimate is not sufficient to reduce an outdated measurement to the

[1]Sender buffer and receive buffer size are 4 MByte and 6 MByte respectively, the default settings in Linux operating system.

current value. Therefore, we develop RTTPROBS to meet the following criteria:

1) Timely update if the new RTT estimate reflects a significant decrease in RTT.
2) Fade out probing if RTT estimates do not indicate a significant decrease in RTT.
3) Avoid probing if no application traffic is transmitted.

For the first criteria, the RTT probing starts with short inter-probing intervals, if the subflow has no packets in flight. The interval doubles if the RTT estimate does not decrease. If the RTT estimate decreases, the probing interval decreases fast to trigger further updates.

In detail, the first RTT probe is scheduled in the interval $RTT_{min}$ if the subflow is not selected and has no packets in flight. The value $RTT_{min}$ is the minimal RTT observed in a window of $k$ seconds, by default $k = 300$ in Linux. Therefore, this values gives a rough estimate of the end-to-end delay excluding queuing delays. The probing interval doubles after each probing, if the new estimate does not indicate a decrease of the RTT estimate. So the probing interval $p$ is

$$p = 2^n RTT_{min} \tag{2}$$

with $n = \{0, 1, 2, 3, ...n_{max}\}$. The parameter $n$ is increased up to a maximal value to also detect a decreasing RTT after a long-lasting RTT increase. We set $n_{max} = 7$, so with an RTT of 30 ms the subflow is probed each 3.84 s.

If the current RTT estimate decreases, the new probing interval is $2^{\max(0,n-\theta)} RTT_{min}$ with

$$\theta = \left\lfloor \frac{SRTT}{CRTT} \right\rfloor + 1 \tag{3}$$

So $\theta$ accounts for the relation of the previous $SRTT$ measurement and the current RTT estimate, i.e. if the new estimate is close to the previously seen $SRTT$, the decrease of the probing interval decreases by the relation, i.e. a sudden strong decrease speeds up the probing more than a small decrease.

Using RTTPROBS, we account for short-term as well as long-term effects with little probing traffic of 1 packet in the interval $2^{n_{max}} RTT_{min}$.

To account for the last criteria, the probing is implemented by a timer started by the MPTCP scheduler only if data is transmitted. The timer is started if a subflow has no packets in flight, the subflow is not selected for transmission by the current scheduled packet, and the timer is not already pending. Therefore, probe packets are only scheduled if the application transmits data.

### A. Implementation

The approach is implemented in the Linux kernel based on MPTCP 0.95.1. As described in [18], a probe packet is sent with the next expected sequence number minus one. For a reliable RTT estimation, we rely on TCP timestamps to explicitly match probe packets and related acknowledgments, since a receiver may silently discard probe packets due to rate limiting for duplicate acknowledgments as implemented in the Linux OS. A Linux TCP network stack sends a duplicate acknowledgment at most each 500 ms by default.

Furthermore, TCP senders and receivers do not update the recent timestamp on probe packets. Therefore, we modify the network stack such that a receiver returns an acknowledgment packet in response to a probe packet with a recent timestamp and modify the probing packet sender to update the RTT on the reception of probe packets acknowledgments.

RTTPROBS is implemented by a new timer. However, we want to emphasize that, due to different time-scales, the approach may reuse the already existing TCP keepalive timer.

## V. Evaluation

In this section we repeat the experiments from Sec. III with the implementation of RTTPROBS described in Sec. IV. We use the Linux operating system with MPTCP enabled kernel version 4.19, MPTCP version 0.95.1 and the default congestion control algorithm, Cubic [23]. The network throughput is derived from packet traces recorded with `tcpdump`, the application throughput and delay is extracted from `D-ITG`, and the SRTT is recorded by TCP's `ftrace` implementation. Note, that the SRTT is recorded on each incoming acknowledgment, i.e. if no packets are sent and no acknowledgments arrive no point is plotted and the SRTT is not updated. To receive timely updates, we disable the rate limitation for duplicate acknowledgments for all following experiments.

### A. Alternating RTT

The improvement of RTTPROBS is presented in Sec. IV for the scenario from Sec. III-A and is shown in Fig. 2a. Due to the probing, the subflow receives continuous RTT updates and switches back to the subflow with the lower RTT.

Still, the interval between the network layer RTT returns to the low delay value (indicated by RTTs measured by `ping`) and the handover is about 5 s. This accumulates from the maximum probing interval of about $2^7 RTT_{min} \approx 4$ s plus multiple measurements until the smoothed RTT decreases below the RTT of the second subflow. Due to the adaptive probing scheme, if a decrease is detected the probing is speed up and the RTT estimate is updated fast.

### B. Wi-Fi Scanning

In Sec. III-B, we showed that the scanning in a Wi-Fi network increases the RTT and leads to usage of the cellular network only, which features a slightly larger RTT. After the client starts a scan, the Wi-Fi network is not used again, due to outdated and large RTTs during the scanning process.

As described in Sec. IV, the RTTPROBS reacts fast on short-term increases of the RTT. The results are demonstrated in 2b. During the scanning at about 5 s, 55 s, 65 s, the delay of the Wi-Fi connection increases. Hence, the scheduler shifts packets to the cellular connection. Due to the improved probing, the RTT is updated timely and decreases after the scanning. Therefore, the scheduler utilizes the Wi-Fi connection again.

In the previous section Sec. V-A, the RTT increase lasted about 30 s. Here, the period is much shorter due to the adaption of the probing interval. It reacts much faster and returns to the original subflow due to updated RTT estimates.

## C. *Bufferbloat*

Sec. III-C indicates that a short increase of the SRTT of one provider, leads to long-term degradation of this provider.

Packets are only scheduled if the SRTT measurement of the current selection subflows increases above the previous level. Fig. 1c shows this transition between providers. RTTPROBS shows a more frequent usage of the provider in Fig. 2c without large gaps as indicated in Fig. 1c.

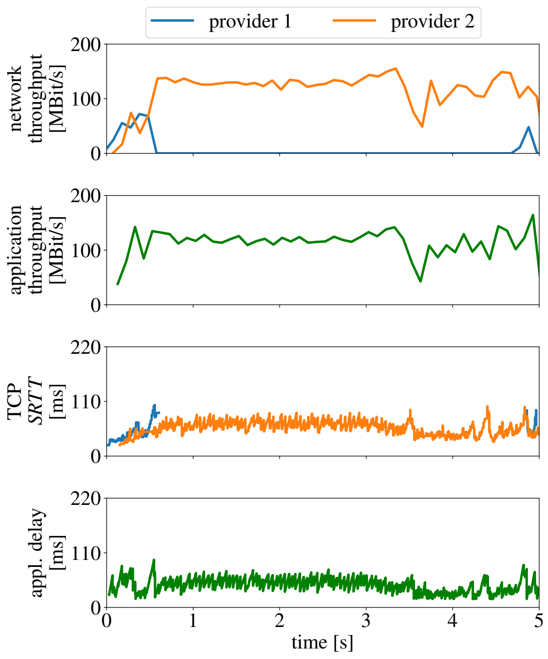
## VI. Cellular Experiments

In [24] and [22], we already showed that MPTCP does not fully utilize the connection paths. This is due to an interaction of buffer size of the receive and send window, the congestion control, as well as deep queues existing inside cellular networks [21]. We showed in [24] and [22] that buffer size tuning, congestion control selection, and also probing improve the utilization. Here, we evaluate in detail the RTTPROBS in a buffer constraint cellular and vehicular scenario.

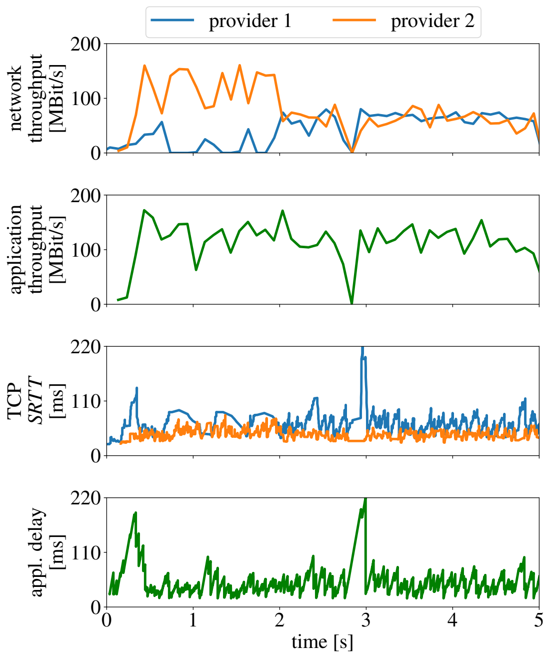
For the measurements, a x86 mini computer is equipped with two LTE category 12 modems. For measurements of the position and accurate time synchronization the embedded global navigation satellite system receiver included in the modem is used. Each modem is connected to two antennas. For the static position experiment, the antennas are placed indoor. For the vehicular experiment the equipment is placed inside a vehicle.

The implementation and software version is the same as described in Sec. V. Measurements are performed in the downlink direction from a cloud server to the vehicle.

Each measurement is performed for 5 s with a greedy traffic source. To compare the probing approach to the regular implementation without probing, measurements with and without probing are taken in turns to avoid temporal and spatial effects. The buffer size for the connection is limited to typical values of 2 MByte for the sender buffer and 3 MByte for the receiver buffer. For the vehicular experiment, the measurements are performed along an urban track of about 10 km and 41 measurements are collected with and without RTTPROBS. For the static experiment 30 runs were performed at a fixed position. The figures Fig. 4 and Fig. 5 show the statistical analysis as boxplots, including the median, the 25th and 75th percentile as box, and the whiskers as the minimal and maximal value. To compare the approaches the average delay, the throughput, and the minimal traffic share of both providers are analyzed. Fig. 3 compares two time series of throughput, SRTT, CWND, and application delay from the vehicular experiment. We observe in Fig. 3a that without a probing scheme only one provider is used for a long-lasting duration after the increase of the SRTT. A switch only occurs after a related SRTT increase. On the other hand, in Fig. 3b we see a more evenly distributed traffic between providers. The aggregated throughput is similar in both examples, due the buffer constraints that limit the maximal throughput.



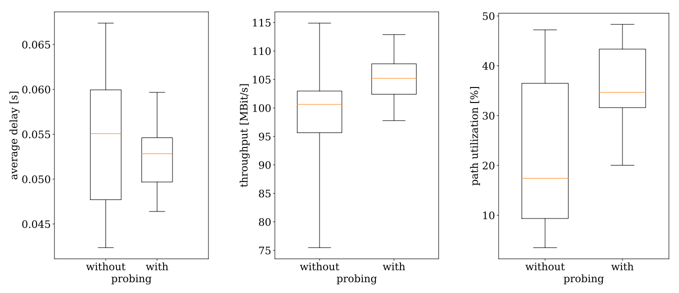
(a) without probing



(b) with probing

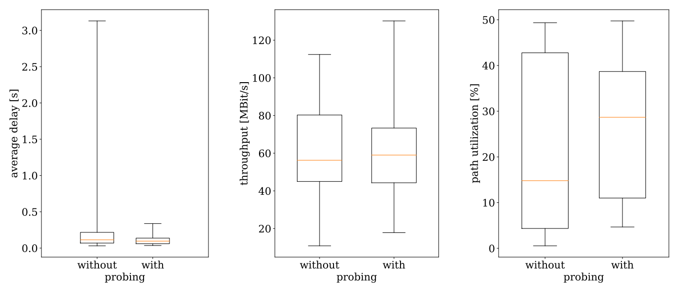
Fig. 3. Two time series of network, application throughput, SRTT and appl. delay for the vehicular experiment with and without probing. The series show that for large intervals only one provider is used without the proposed probing approach in (a). These gaps are significantly reduced using RTTPROBS as highlighted in (b).

The statistical analysis in Fig. 4 and Fig. 5 with and without probing presents the analysis of the average delay during the measurement, the average throughput, and the utilization of the subflows. We observe an improvement for all metrics due to RTTPROBS. The delay is reduced and at the same time the



(a) average delay (b) throughput (c) utilization

Fig. 4. Comparison of end-to-end quality for static cellular scenario. With the implementation of RTTPROBS the average delay and its variance decreases (a), at the same time the throughput increases by about 5% (b), and moreover the traffic is distributed more equally between the subflows (c), it increases from only about 18 % to about 36%.



(a) average delay (b) throughput (c) utilization

Fig. 5. Comparison of end-to-end quality for the vehicular experiment. Similar to the previous experiment, there little improvement in delay (a) and throughput (b), but the traffic is distributed more equally between the subflows (c), the utilization increases in average from 15% to about 29%.

throughput increases. The decrease in delay and increase in throughput is more dominant in the static scenario, here, the delay decreases and at the same time the throughput increases by 5%. The cellular scenario only show little improvement in delay and throughput due to similarity of the providers. In both scenarios, the traffic is balanced more equally between the subflows if probing is used. This guarantees the selection of the better path.

## VII. Conclusion

In this paper, we first emphasize the need for up-to-date RTT estimates in multi-path protocols. We demonstrate in various scenarios that an outdated RTT estimate leads to sub-optimal scheduling by the default MPTCP scheduler, which uses the SRTT as decision criteria. The outdated estimate originates from the usual TCP behavior that the RTT estimate is only updated if data is transmitted. To improve the behavior we introduce RTTPROBS, a probing scheme that on the one hand reacts to short-term as well as long-term RTT changes of the subflows that are currently not used for data transmission. On the other hand, it avoids extensive probing traffic. The benefit is demonstrated in the aforementioned scenarios and is further evaluated in a real-world vehicular scenario. The vehicular experiment demonstrates that our proposal reduces the delay, while simultaneously increasing the throughput and balancing the traffic between subflows more equally.

## References

[1] 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* , Apr. 2012, pp. 399–412.

[2] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental Evaluation of Multipath TCP Schedulers," in *Proc. ACM SIGCOMM Workshop on Capacity Sharing Workshop*, 2014, pp. 27–32.

[3] F. Yang, Q. Wang, and P. D. Amer, "Out-of-Order Transmission for In-Order Arrival Scheduling for Multipath TCP," in *Proc. IEEE WAINA*, 2014, pp. 749–752.

[4] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent Delay-aware Packet Scheduling for Multipath Transport," in *Proc. ICC*, 2014, pp. 1222–1227.

[5] S. Ferlin, O. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking Estimation-based MPTCP Scheduler for Heterogeneous Networks," in *Proc. IFIP Networking)*, 2016, pp. 431–439.

[6] S. Ferlin, T. Dreibholz, and O. Alay, "Multi-path Transport over Heterogeneous Wireless Networks: Does it really pay off?" in *Proc. IEEE Globecom*, 2014, pp. 4807–4813.

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

[8] P. Karn and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 17, no. 5, pp. 2–7, Aug. 1987.

[9] D. Borman, R. T. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance," RFC 7323, Sep. 2014.

[10] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," in *Proc. ACM USENIX NSDI*, 2011, pp. 99–112.

[11] C. Raiciu, M. J. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356, Oct. 2011.

[12] R. Khalili, N. G. Gast, M. Popovic, and J.-Y. L. Boudec, "Opportunistic Linked-Increases Congestion Control Algorithm for MPTCP," IETF, Individual Submission, Internet Draft, Jul. 2014.

[13] A. Walid, Q. Peng, J. Hwang, and S. H. Low, "Balanced Linked Adaptation Congestion Control Algorithm for MPTCP," IETF, Individual Submission, Internet Draft, Jan. 2016.

[14] Yu Cao, Mingwei Xu, and Xiaoming Fu, "Delay-based Congestion Control for Multipath TCP," in *Proc. IEEE ICNP*, 2012.

[15] S. Ferlin, O. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl, "Revisiting Congestion Control for Multipath TCP with Shared Bottleneck Detection," in *Proc. IEEE INFOCOM*, 2016.

[16] Y. Chen and D. Towsley, "On Bufferbloat and Delay Analysis of Multipath TCP in Wireless Networks," in *Proc. IFIP Networking*, 2014.

[17] A. Froemmgen, J. Heuschkel, and B. Koldehofe, "Multipath TCP Scheduling for Thin Streams: Active Probing and One-Way Delay-Awareness," in *Proc. IEEE ICC*, 2018.

[18] R. T. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122, Oct. 1989.

[19] 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.

[20] N. Keukeleire, B. Hesmans, and O. Bonaventure, "Increasing Broadband Reach with Hybrid Access Networks," *IEEE Comm. Standards Magazine*, vol. 4, no. 1, pp. 43–49, 2020.

[21] N. Becker, A. Rizk, and M. Fidler, "A Measurement Study on the Application-level Performance of LTE," in *Proc. of IFIP Networking*, Jun. 2014.

[22] R. Lübben and J. Morgenroth, "An Odd Couple: Loss-Based Congestion Control and Minimum RTT Scheduling in MPTCP," in *Proc. IEEE LCN*, October 2019.

[23] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[24] R. Lübben and J. Schwardmann, "Application Level Performance Measurements of Multi-Connectivity Options in Cellular Networks for Vehicular Scenarios," in *Proc. IEEE LCN*, Oct. 2019.