The HOP Protocol: Reliable Latency-Bounded End-to-End Multipath Communication

Federico Chiariotti, Andrea Zanella, Stepan Kucera, Kariem Fahmi, and Holger Claussen

***Abstract*—Next-generation wireless networks are expected to enable new applications with strict latency constraints. However, existing transport layer protocols are unable to meet the stringent Quality of Service (QoS) requirements on throughput and maximum latency: excessive queuing due to capacity-oriented congestion control inflates end-to-end latency well beyond interactivity deadlines. In this work, we propose a novel framework that evolves best-effort communications into reliability- and latency-aware communications for QoS-sensitive applications. The new protocol, named High-reliability latency-bounded Overlay Protocol (HOP), provides a novel combination of packet-level Forward Error Correction (FEC) and multipath scheduling to compensate for capacity drops and meet pre-defined QoS requirements. More specifically, the sender splits the data and the associated redundancy between the paths by using a stochastic forecast of their future capacity and decides the amount of redundancy necessary to meet the application's requirements without clogging the connections. We compare HOP's performance with state-of-the-art multipath protocols in ns-3 simulations using both synthetic and live network traces, and confirm that our scheme can reliably deliver high-throughput data, reducing the number of late blocks by 2 to 5 times with respect to optimized Multipath TCP (MPTCP).**

I. Introduction

Over the next few years, several new applications are going to be enabled by the novel capabilities of the fifth generation of cellular networks (5G) [1] and gigabit WiFi (WiGig) [2]. High-definition video conferencing over mobile networks, remote robotic operations on the factory floor, and Augmented Reality (AR) are just a few well-known examples: all these applications have a high throughput and strict reliability and latency constraints, which are expected to be met by the new generations of network services, which shall provide smooth, high-quality operation and not just a best-effort service.

However, current transport layer protocols were not designed to meet these demanding requirements, and indeed do not provide any built-in latency guarantees. In fact, the current versions of the Transmission Control Protocol (TCP) cause high delay in order to efficiently probe the path capacity, and the Multipath TCP (MPTCP) standard is not able to effectively support near real-time services [3] because, as for most of the available TCP variants, congestion control has been designed to provide high throughput rather than controlled delay. The unavoidable trade-off is then between reliability, latency and throughput: an increase of the minimum delivery rate is usually achieved at the cost of a lower reliability or higher latency bound. It is then necessary to design a multipath protocol that can explicitly consider Quality of Service (QoS), limiting latency and achieving the required reliability.

The ongoing virtualization of cellular networks makes the deployment of new protocols technically possible, as network operators can place server-side and user-side proxies that will implement new functionalities in a backward-compatible manner. For example, in [4], the authors show how a user-side proxy can be deployed in a user handset without any changes to the operating system kernel. A flexible, application-specific control of the trade-off between reliability, throughput and latency can then become an achievable goal.

In this context, we propose a novel transport protocol that can exploit path redundancy to guarantee the reliable transmission of application data with bounded latency, i.e., meet an application-specified latency target. In order to avoid the pitfalls inherent to MPTCP, we design a protocol called High-reliability latency-bounded Overlay Protocol (HOP) that runs in user space over multiple User Datagram Protocol (UDP) sockets, like the recently standardized QUIC protocol [5]. The first component of HOP is the multipath scheduler, which exploits Forward Error Correction (FEC) to guarantee the timely reception of the data. Redundancy is added to the application data in order to protect them from fluctuations in the capacity of the available paths: lost or late packets on any of the paths can be recovered using the redundant packets sent on the other paths. The scheduler's decisions are based on the capacity estimation mechanism, which tracks the capacity probability distribution of each path and allows the scheduler to make foresighted decisions.

HOP provides applications with the flexibility to choose the operating point in the trade-off between maximum latency, reliability and throughput. While the use of multipath FEC is not in itself a new idea, its optimization is crucial: too much redundancy will clog the buffers and cause additional delays, while too little will be ineffective at protecting the transmission. To the best of our knowledge, HOP is the first scheme to dynamically optimize the amount of redundancy to meet strict latency requirements. Furthermore, no existing protocol can optimize transmission over multiple paths: MPTCP is the only term of comparison, and HOP was able to reduce the frequency of late blocks by 2 to 5 times even working at a 50-70% higher goodput than MPTCP in all the scenarios considered in this work. On the other hand, real-time oriented protocols such as the Real-time Transport Protocol (RTP), have limited multipath extensions that cannot deal with and optimize FEC.

The paper is organized as follows: the relevant literature on the topic is presented in Sec. II, focusing on multipath congestion control and scheduling issues. The HOP protocol

F. Chiariotti (corresponding author, email: fchi@es.aau.dk) is with the Electronic Systems Department, Aalborg University, Denmark. A. Zanella (email: andrea.zanella@unipd.it) and F. Chiariotti are with the Department of Information Engineering, University of Padova, Italy. S. Kucera (email: stepan.kucera@nokia.com) is with Nokia Bell Labs, Munich, Germany. K. Fahmi (email: kariem.fahmi@nokia.com) is with Nokia, Ireland. H. Claussen (email: holger.claussen@tyndall.ie) is with Tyndall Institute, Dublin, Ireland.

is described in detail in Sec. III, while by its two main components, i.e., the scheduler and the capacity estimator, are presented in Sec. IV and Sec. V, respectively. The results of our simulations are described in Sec. VI, and Sec. VIII concludes the paper.

II. Related Work

MPTCP is a fully backward-compatible extension of TCP, published as an experimental standard in 2011 [6]. It is transparent to applications, which can use the standard socket interface. An MPTCP flow is composed of multiple TCP subflows on different paths [7], each of which is a full TCP socket following the protocol: this is necessary to avoid packet dropping in middleboxes, which often discard or block traffic with non-standard behavior [8].

In order to work, MPTCP needs a scheduler to decide which path to send the data on and a congestion control mechanism to adjust the overall traffic flow across the different paths. Most existing congestion control schemes consider all paths when adapting the congestion window. Packets also have an additional sequence number to allow the receiver to reassemble the split streams of the various subflows [9] and deliver the data in the correct order. Retransmissions need to be performed on the same subflow, since each subflow needs to be fully TCP-compliant, but data can be duplicated and retransmitted on other subflows to speed up loss recovery, since there is a connection-level sequence number, which allows the receiver to identify and discard duplicate packets.

MPTCP has multiple theoretical advantages over single-path TCP: using multiple flows can reduce delay, increase throughput and reliability, provide robustness to link failures, and relieve congested paths [10]. However, the design of the scheduling and congestion control mechanisms is critical, and the protocol often performs worse than the single-path version in wireless networks [11], [12].

Since data on each subflow are released in order, MPTCP suffers from the head-of-line blocking phenomenon [3]: when a packet on one path is missing, packets on other paths with higher connection-level sequence numbers are kept in the receiver buffer to avoid out-of-order delivery. Since the retransmission process might take long, particularly if the loss is on a path with a high Round Trip Time (RTT), the receiver buffer might get completely full, blocking any further transmission on the other paths until the missing packet is finally delivered [13]. In this way, any outage or retransmission on any of the paths can negatively affect all the other paths, and in some cases, MPTCP is clearly outperformed by a single-path TCP flow on the best path [14].

In order to avoid these issues and fully exploit the potential of multipath transmission, we have to jointly design both components of the protocol, optimizing the congestion control and scheduler for latency-bounded delivery of data.

A. Congestion control

The TCP Cubic [15] congestion avoidance mechanism uses packet losses to infer congestion and back off from its probing of the channel, exploiting most of the available capacity. However, this causes the protocol to respond sluggishly to changes in the network and suffer from high self-inflicted queuing delays and instability [16], as the only signal of congestion is a buffer overflow. This problem has only been exacerbated by the progressive increase in the size of buffers in cellular networks, which has led to the bufferbloat issue [17], i.e., the experiencing of very long delays, which can often be longer than a second, in some TCP flows because of long queues in some intermediate nodes. Modern congestion control mechanisms such as Bottleneck Bandwidth and Round-trip propagation time (BBR) [18] can achieve a similar throughput with much lower self-inflicted queuing delays, providing a more interesting solution for QoS-aware applications.

MPTCP adds a dimension of complexity to the problem, since the multiple congestion windows for the subflows can be updated jointly. The first design of MPTCP used fully *uncoupled* congestion control: each path had an independent congestion control mechanism, and no information was shared between the subflows. However, this solution is unfair to single-path TCP flows sharing one of the bottlenecks [19], and can often lead to higher inefficiency in the capacity utilization [20]. Linked Increases Algorithm (LIA) [21] improves fairness towards legacy TCP by adopting a *semi-coupled* approach to manage the growth of the subflows congestion windows in the congestion avoidance phase. Fully *coupled* schemes [22] were considered too conservative, as they often completely give up on low-bandwidth paths and never discover subsequent changes in their capacity. Opportunistic LIA (OLIA) [20] is a variant on the LIA scheme that can improve the resource pooling fairness. As of today, the most commonly used scheme is Balanced Linked Adaptation (BALIA) [23], which combines the strengths of LIA and OLIA.

However, these algorithms are often inefficient in wireless networks, as the head-of-line blocking problem is aggravated by the volatility of wireless links. Losses caused by the physical layer can further decrease their performance and stability [24]. There are some proposals for latency-aware congestion control [25], which would mitigate some of the issues of loss-based algorithms, but these proposals are still untested. The use of BBR, which is rapidly gaining traction in single-path TCP [26], is beginning to get traction in the multipath literature [27], as it promises to solve its predecessors' issues. However, our simulations show that even BBR has significant problems, which we will discuss in Sec. VI.

B. Scheduling

In order for MPTCP to work efficiently, designing an appropriate congestion control is not enough: the protocol needs a scheduler to decide which path to send the data on, a decision which is often critical to avoid latency and head-of-line blocking issues. The most basic scheduler, which is currently implemented in the Linux kernel, is the Lowest RTT First (LowRTT) policy: the first packet in the send buffer will be sent on the path with the lowest measured RTT among those with enough congestion window space. However, this heuristic is often inefficient [28], as waiting for a faster path to be free can be better than sending immediately on a slow one

if the difference in their RTTs is large enough. This can lead to significant performance losses [29] even when comparing it to other simple heuristics [30].

Round robin-based [31] and loss-based [32] scheduling schemes have also been proposed, but fail when there is a strong imbalance between the subflows. The most effective schedulers, such as the Slide Together Multipath Scheduler (STMS) [33] and Delay Aware Packet Scheduling (DAPS) [34], try to explicitly model the connection to send packets out-of-order on unbalanced paths so that they will arrive in-order at the receiver. In order to design the packet interleaving mechanisms, the schedulers need to have a model of each subflow and the expected delivery time. The FEC-based scheme in [35] also uses a similar scheduling strategy with good results. The Blocking Estimation (BLEST) [36] scheduler adds the awareness of the possibility of head-of-line blocking to this mechanism, explicitly trying to prevent it. The Shortest Transmission Time First (STTF), scheduler, presented in the same paper, is another scheduler that tries to estimate delivery time and schedule segments on the path that would deliver them earliest. It is important to note that the application features might affect the scheduler, which needs to be tuned to the specific application.

C. Using redundancy in multipath connections

In general, MPTCP only offers best-effort services, as the unpredictability of the end-to-end paths makes it difficult to explicitly support QoS. Wireless channels only exacerbate the issue [24], but there are some FEC-based techniques that can mitigate the head-of-line blocking problem and increase QoS.

FEC is a natural candidate to solve the head-of-line blocking problem: instead of waiting for a potentially lengthy retransmission, receivers can use redundant packets to reconstruct the missing packets. Several solutions [13], [35], [37] with different coding schemes have been proposed, using standard congestion control and different scheduling schemes. However, congestion control is a major part of the problem: self-inflicted buffer overflows and queuing delay still increase latency, and there is no way to measure and guarantee QoS.

The Decoupled Multipath Scheduler (DEMS) [38] is a first attempt to exploit the block-based nature of most applications: DEMS transmits data on one path starting from the beginning of the block, and on the other starting from the end. The delivery is complete when the two flows meet in the middle, and the scheduler foresees an adaptive redundancy mechanism to improve delivery times in variable network conditions.

The Latency-controlled End-to-End Aggregation Protocol (LEAP) [39], which is not compatible with MPTCP but provides the same interface to applications, was the first to consider explicit QoS guarantees, but it has some significant limits: namely, it requires applications to be fully flexible, adapting the size of the data blocks to the possibilities of the multipath connection. This assumption is limiting in modern networks, and our protocol overcomes it, achieving similar results even when applications are uncooperative. An older scheme that runs on Stream Control Transmission Protocol (SCTP) also used cross-path FEC for low-latency applications [40], but it did not provide any adaptation of the coding rate and just used it to recover from packet losses. Redundancy and multipath transport can also be used to improve reliability in other contexts, such as the Internet of Things (IoT) [41].

For a more thorough survey of the research on multipath transport protocols, we refer the reader to [42]. The use of coding at the transport layer is also being investigated by the IETF, which is drafting a standard that enumerates the different possibilities and required signaling in TCP and QUIC [43]. The interaction between coding and congestion control is interesting, as coding can hide losses, improving throughput, if recovered packets are considered as not lost. Existing FEC-based schemes promise to improve QoS as well, but to the best of our knowledge, HOP is the only scheme that dynamically optimizes the amount of redundancy to send and schedules it over the available paths.

III. The HOP protocol

The HOP protocol is designed to meet demanding QoS targets by sending redundant information on multiple volatile paths. If the application writes a batch of B bytes to the send buffer, the objective is to guarantee that the probability that such data are delivered to the destination within the maximum allowed time T is at least P_{thr}. The use of delivery deadlines, both hard (as in our case) and soft, has been studied in the literature as a way to express end-to-end latency requirements [40]. In order to reliably deliver the data with bounded latency, the transport layer protocol must be able to exploit path diversity and compensate for the failures on any of the paths by relying on the others. This calls for cross-path FEC, since retransmission is a slow mechanism that may not ensure a timely delivery in difficult conditions: if redundant packets are added to the transmission, the receiver may be able to retrieve any late or missing packets before the deadline expires. In other words, HOP can deliver the whole data block within the planned deadline even if some packets are lost or late, provided that enough packets are received on time on any combination of the paths and in any order.

As we discussed in the Sec. II-C, FEC is also used by other protocols to reduce packets latency. However, HOP can achieve much better performance by exploiting paths capacity estimates and block deadline information to tune the amount of redundancy to be sent over the available paths, so as to maximize the in-time delivery probability of the blocks without clogging the connections. The dynamic adaptation of FEC is critical, as adding too little redundancy can limit its usefulness, while adding too much will congest the network. As HOP is aimed at guaranteeing QoS for high-priority applications, fairness to flows with a lower priority is a secondary consideration, and the protocol will operate best over connections with no direct cross-traffic: this fits with the 5G network slicing paradigm [44], which separates different types of traffic so that they do not need to compete with flows with different characteristics and requirements [45]. In this context, the optimization performed by HOP can outperform other schemes significantly.

We define the following aims of our QoS-oriented protocol:

1) Exploit multiple paths in such a way that a path failure can be compensated by other paths without retransmis-

sions. This can be accomplished by using systematic packet-level codes, which are equivalent to other linear codes, but simplify the receiver-side decoding process.
2) Dynamically allocate the amount of redundancy and schedule packets on the various paths to maintain the latency below the application-specified maximum T.
3) Estimate the future capacity of the paths in order to make intelligent scheduling decisions.

These requirements are critical to support applications such as AR and teleoperation over mobile networks, with strict latency requirements and high throughput. The advanced network access services provided by 5G and beyond networks alone may not be sufficient to meet the stringent constraints, as a delay-agnostic transport layer may disrupt the performance objectives. HOP can avoid this issue by running in user space over standard UDP sockets. Like the recent QUIC protocol, it creates a reliable, connection-oriented flow over the connectionless UDP sockets by implementing its own acknowledgment and sequencing mechanisms. HOP has three components. The first is a systematic packet-level encoder and decoder with tunable coding rate. In the experiments, we used the implementation of shortened-and-punctured Reed-Solomon codes described in [46]. It also has a scheduler that optimizes the level of FEC and sends packets over the different paths, as well as a capacity estimation mechanism named Sender-side Kalman Inference Procedure (SKIP), which can predict the distribution of the future capacity of each path, giving the scheduler the full picture of the state of the connection.

If the estimate of the future capacity distributions provided by SKIP is correct, the scheduler can calculate the probability that a packet sent on any path will be late and set the amount of FEC to guarantee that the data block will be received within the planned deadline with a given probability. We define three schedulers with different objectives:

- The Just Use Multipath Protection (JUMP) scheduler deals with inelastic applications: it considers a fixed block size B, a fixed reliability P_{thr} and a fixed latency bound T. If adding FEC is enough to meet the reliability constraint, JUMP minimizes the redundancy. Otherwise, it signals a failure to the application.
- The Latency-controlled End-to-End Aggregation Protocol (LEAP) scheduler, named after the multipath protocol from [39], considers a fixed reliability P_{thr} and a fixed latency bound T. Like its namesake, it requires an adaptive application, which can change the amount of data in a block by compressing it, and it creates a schedule that maximizes the deliverable block size B while respecting the other constraints.
- The Flexible Latency Improvement Procedure (FLIP) scheduler considers a fixed block size B and a minimum reliability P_{thr}. It schedules the packets in the way that minimizes the latency bound T. It can be used for applications that can tolerate some latency fluctuations, but need reliability and are not throughput-adaptive.

These schedulers aim at maintaining the latency below the application-specified maximum T, and they will never schedule packets that are expected to cause unacceptable queuing. In this way, congestion control is implicit in the scheduling: as long as the bottleneck buffer on a path is larger than CT, where C is the capacity of the path, JUMP, FLIP, and LEAP will never cause a congestion loss. Since HOP's target applications have low latency targets and, as previously discussed, buffers in the network are growing, we expect this to be the case in most practical deployments. However, if the bottleneck buffer is shallow and packets are dropped, the capacity estimate given by SKIP will be decreased accordingly and the scheduler will adjust by reducing the number of packets on the affected path.

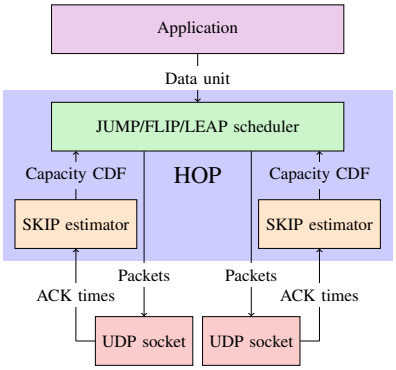

Fig. 1: Sender-side data flow. The HOP protocol components are highlighted in blue.

Scheduled packets are sent in a burst, since additional delay is not an effective strategy to guarantee in-time delivery. This burst-based mechanism affects other users sharing the bottleneck buffer, but most modern routers have per-user buffers, so the burstiness should not affect other flows. The Sprout congestion control mechanism [47] makes the same assumption and verifies it in cellular networks, and we will show that it holds for modern 802.11 routers in Sec. VI. We plan to investigate friendlier versions of the protocol that do not send packets in a burst and maintain fairness in legacy shared-buffer links, but this is outside the scope of this work.

In the following sections, we will describe and model the three schedulers, starting from JUMP, since its procedure is a basic building block for the other two. Then, we will present the SKIP capacity estimator.

IV. Scheduler design and analysis

The objective of JUMP is to efficiently schedule a block of k packets on the m available paths, minimizing redundancy while meeting the reliability and latency constraints. We assume that the size L of all packets sent on a certain path is always equal to the path Maximum Transfer Unit (MTU). Furthermore, we indicate by $F(i, \omega_i, T)$ the so-called *lateness probability* for path i, i.e., the estimated probability that a packet of length L sent on path i is delivered with a latency larger than the maximum allowed delay T. This probability depends on the path status ω_i, which consists of the path capacity distribution and the number of queued and in-flight packets. The computation of the function $F(i, \omega_i, T)$ is explained in Sec. V, being strictly coupled with the SKIP capacity estimation mechanism. If we call the function $F(i, \omega_i, T)$ repeatedly, we can get the lateness probability for each subsequent packet

sent on the same path. We hence denote by $\lambda_i(j,T)$ the lateness probability of the j-th packet in a block sent on path i. We assume that $\lambda_i(0,T) = 0$.

We now define a *schedule* as a vector $\mathbf{s} \in \mathbb{N}^m$, whose i-th element s_i represents the number of packets sent on path i for the current block. The size of the schedule is defined as the total number of packets scheduled on all the paths and denoted by $|\mathbf{s}| = \sum_{i=1}^{m} s_i$. The scheduled multipath transmission is successful if at least k packets are received on time, so that the receiver is able to decode the block. The failure probability $P_e(\mathbf{s},k,T)$ is then the probability of delivering fewer than k packets in total on all paths. We can now define the optimal schedule $\mathbf{s}^*$ as the minimum-size schedule that meets the reliability constraint:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}\in\mathbb{N}^m} |\mathbf{s}| \quad \text{s.t. } P_e(\mathbf{s},k,T) \leq 1 - P_{\text{thr}}. \tag{1}$$

We observe that, neglecting packet losses due to link errors and reordering, there is no advantage in sending more than k packets on the same path, since the reception of the first k packets will permit to recover the whole data block, making any later packet useless. Therefore, information and redundancy packets should be distributed across the different paths, but without exceeding a total of k packets per path.

We exploit an iterative strategy to generate the optimal schedule $\mathbf{s}^*$ by starting from an empty schedule and gradually adding packets until the condition in (1) is respected. Assuming First In First Out (FIFO) queuing, $\lambda_i(s_i,T)$ is monotonically non-increasing in s_i. The last packet injected in the path will always have a higher or equal probability of being late than previous packets, whose lateness probability is not affected by later packets. We can then expand the schedule at each step by adding a packet on the path i^* with the lowest lateness probability:

$$i^* = \arg\min_{i\in\{1,\dots,m\}} \lambda_i(s_i+1,T). \tag{2}$$

It is easy to prove that the schedule obtained with this iterative procedure is optimal, since the lateness probability of any packet not yet scheduled is higher than that of the already scheduled packets.

In order to compute $P_e(\mathbf{s},k,T)$, we define the delivery vector $\mathbf{d} = (d_1,\dots,d_m)$, whose i-th element d_i represents the number of packets that were delivered on time on path i. Naturally, $d_i \leq s_i$, since it is impossible to deliver more packets than those that were sent. We now define the set of delivery vectors such that the cumulative number of delivered packets on all paths is at least k:

$$D_k(\mathbf{s}) = \left\{ \mathbf{d} : \sum_{i=1}^{m} d_i \geq k, d_i \leq s_i, \ \forall i \in \{1,\dots,m\} \right\}. \tag{3}$$

The scheduler needs to iterate over the set in order to compute the lateness probability of the schedule, so the size of D_k is directly related to the computational complexity of the scheduler. Considering that each of the m elements $\{d_i\}$ of $\mathbf{d}$ can take values in the set $\{0,\dots,s_i\}$, and that $s_i \leq k$, the size of D_k cannot be larger than $\prod_{i=1}^{m}(s_i+1) = (k+1)^m$. Conservatively, we assume it to be $O(k^m)$. The lateness probability of the schedule is then given by

$$P_e(\mathbf{s},k,T) = 1 - \sum_{\mathbf{d}\in D_k(\mathbf{s})} \prod_{i=1}^{m} \psi_i(d_i,T), \tag{4}$$

where $\psi_i(d_i,T)$ is the probability that the first d_i packets sent on path i are delivered on time, while the following packets are late. Since we assume per-path in-order delivery, we get

$$\psi_i(d_i,T) = \begin{cases} (1-\lambda_i(d_i,T))\lambda_i(d_i+1,T), & \text{if } d_i < s_i; \\ 1-\lambda_i(d_i,T), & \text{if } d_i = s_i. \end{cases} \tag{5}$$

The overall complexity of the lateness probability calculation is then $O(mk^m)$, since evaluating a vector $\mathbf{d} \in D_k$ takes $O(m)$ steps and there are $O(k^m)$ elements in D_k. If $P_e(\mathbf{s},k,T)$ is known, we can add a packet to the schedule without recomputing everything. We use the iterative procedure from the previous section to decide which path to schedule the new packet on, then define the vector $\mathbf{s}'$, with $s'_i = s_i + 1$ and $s'_j = s_j, \ \forall j \neq i$. The set $D_k(\mathbf{s}')$ is then given by

$$D_k(\mathbf{s}') = D_k(\mathbf{s}) \bigcup \{ s_i + 1, D_{k-1}(\mathbf{s}_{-i}) \}. \tag{6}$$

Since all the vectors in $D_k(\mathbf{s})$ satisfy the condition in (3), the vectors with $d_i = s_i + 1$ are the only new combinations, so $D_k(\mathbf{s}')$ has $O(k^{m-1})$ elements that are not in $D_k(\mathbf{s})$. Using the old schedule information could be extremely significant, since the computational complexity of the schedule lateness probability is reduced from $O(mk^m)$ to $O(mk^{m-1})$. These results are similar to the work in stochastic job scheduling [48], [49], although in a different setting and with different assumptions. Additionally, the iterative building procedure allows us to reduce the computational complexity of the scheduler thanks to the stopping condition: if the optimal schedule has N packets, only $N-k+1$ schedules need to be evaluated, making the whole procedure $O((N-k)mk^{m-1})$. Non-iterative schedulers have an exponential complexity in N, since the set of possible schedules is combinatorial and they cannot exploit the monotonicity of $\lambda_i(s_i,T)$.

The impact of estimation errors on this method is limited, as the scheduler is self-correcting: if capacity is overestimated on any of the paths, the increased queue will make the scheduler shift part of its load to other paths, mitigating the effect of the estimation error. Naturally, less accurate estimates will result in more errors, but we include a control mechanism to avoid instability in any case. In order to maintain the latency on the path below T, the scheduler will never schedule more packets than those that can be delivered in time. Denoting by $\tau_{\min,i}$ the propagation and processing time on path i, and by μ_i the mean path capacity, then the maximum number of packets that can be scheduled on the path is $S_{i,\text{JUMP}} = \mu_i(T - \tau_{\min,i})$.

In practice, the capacity estimation mechanism described in Sec. V requires to send at least a few packets per block; in order to avoid underutilizing slower or more volatile paths, we set a minimum of 10 packets for each path in the practical scheduler. An additional correction can be added for small probability differences, privileging the least-used path if the

difference between $\lambda_i(s_i{+}1)$ and $\lambda_j(s_j{+}1)$ is lower than a minimum value ε. This allows the scheduler to avoid imbalances due to small variations in the capacity distribution estimates. In both cases, the slight sub-optimality of the schedule will be compensated for with additional FEC on the other paths, accepting a small loss of efficiency to maintain the output more stable and to avoid missing changes in the capacity distribution of less reliable paths.

A. The LEAP and FLIP schedulers

Searching for the optimal solution with flexible latency or block size is more complex: since the schedule construction can only evaluate one at a time, a search procedure becomes necessary. The LEAP and FLIP schedulers essentially reuse the JUMP iterative procedure, running it multiple times to find the optimum by combining it with this search algorithm over the relevant space, i.e., the possible block sizes or latency limits. In both cases, properties of the lateness probability can be exploited to reduce the complexity of the search, limiting the number of examined schedules.

LEAP's objective is to find the maximum number of packets k^* that can be delivered through any schedule:

$$k^* = \sup\left\{k \in \mathbb{N} : \max_{\mathbf{s}\in\mathbb{N}^m} P_e(\mathbf{s}, k, T) \leq 1 - P_{\text{thr}}\right\}. \tag{7}$$

In order to find this value, the scheduler also identifies the optimal schedule. The maximum deliverable block size is then passed to the adaptive application, and the resulting packets are sent according to the schedule. Naturally, the values of k are always between 0 (i.e., no payload data can be sent reliably) and $S = \sum_{i=1}^{m} S_{i,\text{LEAP}}$, the bound on the number of schedulable packets, which is stricter than for JUMP, as LEAP will tend to stay closer to the bound to maximize the block size:

$$S_{i,\text{LEAP}} = (\mu_i - \alpha\sigma_i)(T - \tau_{\min,i}), \tag{8}$$

where μ_i is the estimated path capacity, σ_i is the standard deviation of the estimate, and $\alpha > 0$ is a tunable parameter. The bound is more conservative, since LEAP will always operate close to the limit to maximize the size of the delivered information block. It is possible to determine the optimal schedule for a given value of k by running JUMP, with $O((N-k)mk^{m-1})$ complexity. We can then exploit one fact: if $k_1 < k_2$, the set of delivery vectors that result in a success for k_2 also results in a success for k_1. This can be expressed as $D_{k_2}(\mathbf{s}) \subseteq D_{k_1}(\mathbf{s}) \forall \mathbf{s}$: since the on-time delivery probability depends on the sum of the probabilities of the delivery vectors in the set, it is monotonically decreasing in k. This monotonicity then allows us to use a binary search on k, significantly reducing the number of steps in the search from $O(S)$ to $O(\log(S))$.

The LEAP scheduler is then a simple binary search over k, with values between 0 and S. The value of k is increased when JUMP returns a valid schedule, and decreased when it does not, until the minimum and maximum values of k converge. The overall complexity of the scheduler is $O(S\log(S)mk^{m-1})$.

FLIP works the same way, minimizing the delay bound T:

$$T^* = \inf\left\{T \in [0, T_{\max}] : \max_{\mathbf{s}\in\mathbb{N}^m} P_e(\mathbf{s}, k, T) \leq 1 - P_{\text{thr}}\right\}. \tag{9}$$

In this case, it is necessary to define a maximum acceptable latency $T_{\max}$, as well as a time quantum ε to discretize the search space. It is possible to use a binary search in this case as well, since $\psi_i(d_i, T)$ is monotonically increasing in T. The procedure is then the same as LEAP, but the value of T is decreased when JUMP returns a valid schedule. Since the number of steps in the search is $O\left(\log\left(\frac{T_{\max}}{\varepsilon}\right)\right)$, the overall complexity of FLIP is $O\left(\log\left(\frac{T_{\max}}{\varepsilon}\right) mk^{m-1}\right)$.

V. Estimating capacity: the SKIP method

The JUMP scheduler is the optimal scheduler if the capacity Cumulative Distribution Function (CDF) for each path is known, which is not the case in practical settings. We then design the SKIP mechanism to estimate the capacity CDF of the connection from the recently received ACKs. In the following, we will omit the path index from the notation for readability, but the SKIP estimation process is run independently for each path. Let us first introduce some notation. Let $\mathbf{t} = (t_1, \ldots, t_n)$ be the vector that collects the transmission instants of n packets sent on the path. The size of all packets is L, corresponding to the MTU. Furthermore, let $\mathbf{a} = (a_1, \ldots, a_n)$ be the vector of the corresponding ACK reception instants, and T_k be the period during which we collected the ACKs. It follows that $a_i \geq a_j \ \forall i > j$. This in-order delivery hypothesis is supported by observations in modern networks, and it is the basic assumption behind loss-based versions of TCP. The corresponding RTT vector is $\boldsymbol{\tau} = \mathbf{a} - \mathbf{t}$, and its j-th element is given by

$$\tau_i = \tau_{\min} + \frac{L}{C} + q_i. \tag{10}$$

where $\tau_{\min}$ is the propagation and processing time, C is the path capacity, defined as the available bottleneck bitrate, and q_i is the time required to get rid of the backlog when the packet is generated, which can be computed as

$$q_i = \left(\frac{X(t_i)}{C} - (t_i - t_1)\right) u(G(t_i) - C), \tag{11}$$

where $X(t_i) = iL$ is the number of bytes in flight at time t_i, $G(t_i)$ is the average sending rate between t_1 and t_i, and $u(x)$ is the step function. In practice, (11) states that, if the sending rate exceeds the bottleneck capacity, the queuing delay grows linearly in time. Naturally, this only holds if C is constant, which is not true in a general communication channel: however, this simplification is close enough to reality if the interval of time over which we measure is very short. Moreover, for more dynamic scenarios, the overall performance degradation due to this simplification is expected to be graceful. The relation between $X(t_i)$ and $G(t_i)$ is given by

$$G(t_i) = \frac{X(t_i)}{t_i - t_1}. \tag{12}$$

From (10) and (11), we know that

$$\tau_i = \tau_{\min} + \frac{L}{C} + \max\left(0, \frac{X(t_i)}{C} - (t_i - t_1)\right). \tag{13}$$

If we assume that $G(t_i) \simeq G(t_j)$ and apply (12), we get

$$\tau_n - \tau_1 = \max\left(\frac{G(t)(t_n - t_1)}{C} - (t_n - t_1), 0\right). \quad (14)$$

If we take the derivative over time of the RTT on the path, we get

$$\frac{\partial \tau}{\partial t} = \frac{G(t_n)}{C} - 1. \quad (15)$$

We can now estimate $\frac{\partial \tau}{\partial t}$ by using linear least squares fitting:

$$\frac{\partial \tau}{\partial t} = \frac{L \sum_{i=1}^{n} \left(t_i - \frac{\sum_{j=1}^{n} t_j}{n}\right)^2 \left(\frac{\tau_i}{L} - \frac{\sum_{j=1}^{n} \tau_j}{nL}\right)}{\sum_{i=1}^{n} \left(t_i - \frac{\sum_{j=1}^{n} t_j}{n}\right)^2}. \quad (16)$$

By combining (15) and (16), we can get an estimate of the average capacity, which we can store to get an empirical CDF of the path capacity. The norm of the residual error of the linear fit is useful to gauge the jitter of the path, i.e., capacity fluctuations that are faster than the estimation timestep T_k.

The derivation above only works if there is some queuing, i.e., if the send rate $G(t_n)$ is larger than the path capacity. Most capacity estimation methods share this limit, as Jaffe proved in 1981 [50]. However, HOP sends packets of the same block in a burst, and the issue is avoided. The value of $\tau_{\min}$ can be estimated online, as in BBR [18]. This estimation can only be performed when the data rate is large enough, so we perform a traditional Cubic slow start procedure when setting up a new link. This does not affect the performance once the path exits the slow start phase, when the SKIP method kicks in.

A. Kalman capacity tracking

The capacity estimation process we defined above is inherently affected by noise; in order to compensate for the resulting fluctuations in the measured capacity, we use a Kalman Filter (KF) to track and predict the evolution of the capacity. The KF, invented in 1960 [51], is the optimal way to track known linear systems with Gaussian noise. The basic system model we use is very simple, and operates on discrete timesteps with interval T_k, which is equivalent to the capacity estimation timestep:

$$x_{t+1} = x_t + w_t; \quad (17)$$

$$C_t = x_t + v_t. \quad (18)$$

In the model, the measured capacity C_t is a noisy observation of a hidden state x_t, whose evolution follows a simple Gauss-Markov process. The variances $\sigma_w^2 = Q$ and $\sigma_v^2 = R$ are required inputs of the filter, which needs to distinguish between temporary variations due to v_t (*estimation noise*) and long-term effects that change the state of the system due to w_t (*process noise*). The KF gain K_t is calculated based on the *a priori* and *a posteriori* estimate of the prediction error variance. The KF is the optimal estimator if its parameters are correct for the tracked system, but it needs the values of Q and R as inputs, and its results can degrade quickly if the settings are wrong [52]. If the variances are unknown or can change over time, as in our case, there are several techniques to estimate them online while tracking the underlying process [53]; this extension of the KF is called Adaptive Kalman Filter (AKF). In our work, we use the recursive AKF implementation from [54].

The AKF can deal well with gradual changes to the capacity, but it does not respond quickly enough to sudden capacity drops or even full outages. Wireless channels are often characterized by sharp drops and temporary outages, and this issue is particularly significant at higher frequencies, where transitions from line of sight to non-line of sight propagation due to blockage from walls, objects or humans are very fast.

In order to describe the operation mode logic, we will consider time step t, for which the AKF has a mean $\hat{x}_{t|t-1}$ and a total variance $\Sigma_{\text{AKF}} = \Psi_{t|t-1} + Q + R$. First, we examine the full outage case: if the measured capacity is 0 for at least t_{thr} samples, the link is in outage and the sending rate should be reduced drastically. This condition can be recognized by the occurrence of the following conditions:

$$\sum_{k=t-t_{\text{thr}}}^{t} C_k = 0 \quad \text{and} \quad \hat{x}_{t|t-1} > 0. \quad (19)$$

If this second condition is verified, the mean of the AKF is reset to the initial minimum value (50 kB/s in our setup) straight away. The number of necessary samples t_{thr} affects the robustness of the mechanism to random temporary changes in the capacity, but there is a trade-off between stability and responsiveness. We found that setting $t_{\text{thr}} = 3$ guarantees that the mechanism avoids following noise in the capacity measurement, while still strongly outperforming standard BBR in terms of responsiveness to outage event.

Other critical cases to consider involve deep and sudden variations in path capacity that significantly deviate from the Gaussian model assumed by the AKF. Therefore, we assume one such events occurs when the capacity measurement differs for more than 3σ form the estimate provided by the AKF filter, i.e., when

$$|C_t - \hat{x}_{t|t-1}| > 3\sqrt{\Sigma_{\text{AKF}}}. \quad (20)$$

We refer to such events as down-steps or up-steps, depending on the direction of the capacity change. The occurrence of this condition takes the capacity estimator into the so-called *step-mode*. In case of down-step, the AKF is not updated with the new capacity measurements, but evolved in open-chain, only performing the prediction step and, thus, gradually increasing its variance. The condition (20) is evaluated again at each update, with the new value of σ, and the down-step mode is exited when the down-step condition is not verified in at least τ consecutive steps. However, after $t_{\text{thr}} > \tau$ updating steps in this mode, the capacity drop is assumed to be long-standing and the mean of the AKF is reset to the mean capacity value measured during the down-step, i.e.,

$$\mu_{\text{step}} = \frac{1}{t_{\text{thr}}} \sum_{k=t-t_{\text{thr}}+1}^{t} C_k. \quad (21)$$

In case of up-step, the AKF evolves normally, slowly increasing the estimated capacity towards the current measured value, but after t_{thr} updating steps, the capacity change is assumed to be long standing, and the mean of the AKF is set as for (21).

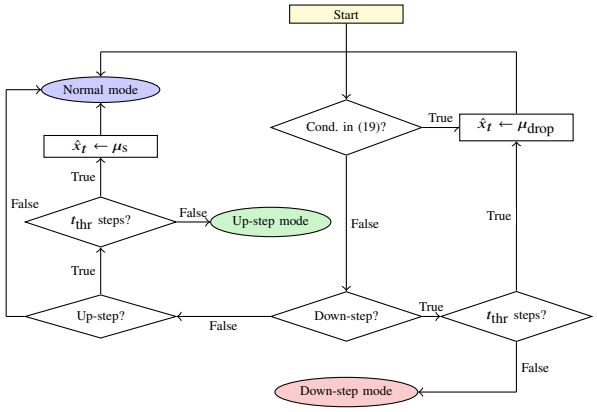

Fig. 2: Block diagram of the filter operation modes

The modes of operation of the filter are also graphically illustrated in the block diagram in Fig. 2. These conditions allow SKIP to maintain responsiveness to sudden events while following the trend of the capacity.

B. Packet lateness conditions

In order to schedule packets on the multipath connection, we need to determine the probability that a packet will be delivered within a given time on a path, given the feedback information. This can then be used by the multipath scheduler to determine which paths to send the packet on. Since the capacity estimation block returns a stochastic estimate of the path capacity, we consider its output, which has a constant timestep T_k, as the capacity CDF. To simplify the calculations, we consider the maximum latency T to be an integer multiple of T_k. We consider a window of n packets in flight, whose sending times are contained in the vector $\mathbf{t} = (t_1, \ldots, t_n)$, with $t_i \geq t_j \forall i > j$. The current time is t_n, as the scheduled packet is the last in the window. If we assume that the propagation delay $\tau_{\min}$ is symmetrical, the reception time r_i is $\frac{\tau_{\min}}{2}$ before the acknowledgment time a_i for the same packet i.

In case packet i suffers from a queuing delay q_i, its reception time r_i is

$$r_i = t_i + \frac{\tau_{\min}}{2} + q_i + \frac{L}{C}, \tag{22}$$

as a consequence of (10). The value of q_i is derived from (11). If we apply (22), we can get a recursive expression of the queuing delay:

$$q_i = \max\left(0, \frac{L}{C} - (t_i - t_{i-1})\right). \tag{23}$$

If we consider the time when the last acknowledgment packet was received, a_0, we get the expression of q_1:

$$q_1 = \max\left(t_1 + \frac{\tau_{\min}}{2}, a_0 - \frac{\tau_{\min}}{2}\right). \tag{24}$$

In order to calculate the in-time delivery probability for packet i, we can use the complementary of its lateness condition, which can be expressed as $r_i - t_i \geq T$. We can apply (22) to express this condition as a function of the path capacity, which yields

$$C \geq \frac{L}{T - \frac{\tau_{\min}}{2} - q_i}. \tag{25}$$

We can now use (23) to distinguish two cases:

$$C \geq \begin{cases} \frac{L}{T - \frac{\tau_{\min}}{2}}, & \text{if } q_i = 0; \\ \frac{2L}{T - \frac{\tau_{\min}}{2} - q_{i-1} + (t_i - t_{i-1})}, & \text{if } q_i > 0. \end{cases} \tag{26}$$

The same operation can be performed recursively, getting a set of i conditions that need to be respected to guarantee in-time delivery:

$$C \geq \frac{iL}{T - \frac{\tau_{\min}}{2} + (t_i - t_\ell) - q_1 \delta(\ell - 1)} \; \forall \ell \in \{1, \ldots, i\}, \tag{27}$$

where $\delta(x)$ is 1 if x is 0 and 0 otherwise. Since the violation of any of these conditions causes the packet to be late, we consider the most stringent one, denoting the corresponding capacity value as $C_{\min}$.

C. Computing the lateness probability

We now calculate the packet delay CDF in order to be able to evaluate the probability of meeting the conditions in (27). As above, we assume that $T = kT_k$. Since the first unacknowledged packet is still in flight, the packet capacity in the first timeslot of the AKF is limited by:

$$C_{\max} = \frac{L}{t_n - t_1 - q_1 - \frac{\tau_{\min}}{2} - J}, \tag{28}$$

where J is a term to account for the path jitter. We now know that $C_1 < C_{\max}$, but the usable time spans $k = \frac{T}{T_k}$ steps of the KF; if we know the KF's parameters $\hat{x}$, P, Q, and R, we need to evolve it blindly for $\ell = \frac{\tau_{\min}}{T_k}$ steps to account for the delay in the feedback: at the j-th step since the starting time t_i, the KF should have a mean $\hat{x}$ and a variance $(\ell + j)Q + R$. However, we consider the additional information given by the knowledge that $C_1 < C_{\max}$. Considering that the distribution of C_1 is now a truncated Gaussian with support $\left[0, \frac{L}{t_n - t_1 - q_1 - F - J}\right]$. The mean of the KF in successive steps changes, and it is given by:

$$\hat{x}_{2|1} = \frac{P + (\ell + 1)Q}{P + (\ell + 1)Q + R}\hat{x}_{1|0} + \frac{RC_1}{P + (\ell + 1)Q + R}. \tag{29}$$

We know that the mean of a truncated Gaussian variable $X \sim \mathcal{N}(\mu, \sigma^2)$ with support $[a, b]$ is given by:

$$\bar{X} = \mu + \sigma \frac{\left(\varphi\left(\frac{b-\mu}{\sigma}\right) - \varphi\left(\frac{a-\mu}{\sigma}\right)\right)}{\left(\Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)\right)}, \tag{30}$$

where $\varphi(x)$ is the normal Probability Density Function (PDF) and $\Phi(x)$ is the normal CDF. We can now substitute the mean of C_1 in (29), getting:

$$\hat{x}_{2|1} = \frac{P + (\ell + 1)Q}{P + (\ell + 1)Q + R}\hat{x}_{1|0} + \frac{R\bar{C_1}}{P + (\ell + 1)Q + R}. \tag{31}$$

The sum of the next $k - 1$ steps is a Gaussian variable, given by:

$$C_{2,\ldots,k} \sim \mathcal{N}\left((k-1)\hat{x}_{2|1}, \frac{(k + \ell)(k - 1)Q}{2} + R\right). \tag{32}$$

The CDF of the overall capacity can only be computed numerically [55], [56], since it involves a convolution with

a truncated Gaussian. The actual distribution is computed in the Appendix, but in order to reduce the computational load, we can approximate the truncated Gaussian variable C_1 as a normal Gaussian variable in the sum:

$$C_{1,\dots,k} \sim \mathcal{N}\left(\hat{x}_{1|0} + (k-1)\hat{x}_{2|1}, \frac{(k+\ell+1)Q + 2R}{2}\right). \quad (33)$$

Adding the outage probability p_{out} to the model to account for 3σ drop events, the lateness probability $\lambda(i) = P[r_i - t_i \leq T]$ is given by:

$$\lambda(i) = p_{\text{out}} + (1 - p_{\text{out}})\Phi\left(\frac{C_{\min} - (\hat{x}_{1|0} + (k-1)\hat{x}_{2|1})}{\sqrt{\frac{(k+\ell+1)Q}{2} + R}}\right), \quad (34)$$

where p_{out} is a running estimate of the down-step and outage probability. In case a drop is in progress, capacity is modeled as an exponential random variable with mean C_{out}. The sum of k exponential random variables is a Gamma random variable with shape parameter $\alpha = 2k$ and mean parameter $\mu = C_{\text{out}}$: $C_{1,\dots,k} \sim \Gamma(2k, C_{\text{out}})$. In this case, we conservatively assume that the outage will last until the packet deadline. The delay CDF can be computed using (34).

VI. Testing scenario and results

In order to test the performance of HOP, we simulated the protocol over the `ns-3` network simulator in a controlled scenario. Although the protocol can operate over any number of paths, we limit this analysis to the two-path case, as the results are easier to interpret. We used two `PointToPointChannel` objects to simulate the paths, varying their capacity according to traces, both controlled and from real wireless networks. In our simulations, a server periodically sends data blocks to a client through two parallel paths. The minimum RTT is set to 25 ms, and the deadline is 35 ms with 98% reliability: this is realistic for an edge application such as AR with strict real-time constraints. The MTU is set to 1500 B. The maximum latency T is set to less than 1.5 times the minimum RTT, so retransmissions are impossible without violating the latency requirement. The 98% reliability requirement is also very strict, as the failure rate is 5 times lower than what standard MPTCP can usually achieve in wireless links [39]. In order to compare HOP to state-of-the-art solutions, we test three other MPTCP variants in this challenging scenario:

- Uncoupled Cubic congestion control with the LowRTT scheduler. This version is slightly more aggressive than current MPTCP implementations, but its mechanism is similar to currently implemented versions of the protocol.
- Uncoupled BBR [18] congestion control with an enhanced version of the DAPS scheduler [34]. This version uses a latency-aware congestion control and a state-of-the-art scheduler aimed at reducing delivery delay and reordering at the receiver, which can exploit BBR's estimate of the path capacity to improve its decisions.
- Uncoupled BBR with the STTF scheduler [36], which schedules packets on the fastest path (considering currently estimated capacity and queued packets).
- Uncoupled BBR with the DEMS [38] scheduler, using redundancy to speed up block delivery time. This is the most meaningful comparison for HOP, as DEMS can use FEC to protect data blocks and prevent small capacity drops on one path from affecting the overall transmission.

As the receiver buffer sizes are set to large values in the simulation, head-of-line blocking is not an issue, so using uncoupled congestion control provides a better performance than coupled algorithms. Furthermore, we argue that the service enabled by QoS-oriented transport will be relevant in *intelligent* future networks, which can be expected to have some form of slicing mechanism to separate different classes of traffic. Consequently, we leave the study of the fairness of the protocol towards best-effort TCP flows to future work.

In order to test the protocols in a controlled setting, we generate synthetic scenarios with fast-varying capacity. In each scenario, the capacity of the bottlenecks changes every 25 ms:

- In the *balanced* scenario, the capacity distribution for the two paths is the same. Samples are Gaussian and Independent and Identically Distributed (IID), with an average of 40 Mb/s and a standard deviation of 8 Mb/s;
- In the *unbalanced* scenario, capacity samples are Gaussian and IID with a standard deviation of 8 Mb/s, but the mean is different for the two paths. One path has a mean capacity of 32 Mb/s, while the other's is 48 Mb/s.
- In the *highly unbalanced* scenario, capacity samples are still Gaussian and IID with a standard deviation of 8 Mb/s, but the means of the two distributions are very different: the first path has a mean capacity of 64 Mb/s, while the second's is just 16 Mb/s. This large difference presents a more challenging scenario, as the scheduler will need to send enough packets on the low-capacity path to maintain an accurate capacity estimation.
- In the *RTT unbalanced* scenario, the capacity is distributed as in the balanced scenario, but instead of having a minimum RTT of 25 ms, the two paths have respective minimum RTTs of 20 ms and 30 ms.

We also added a theoretical bound to the synthetic trace simulations: by using the capacity distribution, we calculated the maximum achievable reliability for the given data block size in case of no queuing and perfect scheduling with full repetition. If we consider ideal conditions with no queue at the bottlenecks and set a 98% reliability and a 35 ms maximum latency, the maximum achievable goodput when respecting the application constraints is about 51 Mb/s. We remark that the capacity volatility in these scenarios is significant, and corresponds to a highly congested wireless network. Furthermore, since this bound does not take queuing into account, it can be very loose if the scheduler causes self-queuing delays by sending too many packets on a path. In order to evaluate SKIP, we also used the JUMP scheduler, feeding it with the correct capacity distribution. As the scheduler is optimal, this is an upper bound that considers queuing.

We also captured three live capacity traces and used them to drive the simulations (with the same parameters used in the synthetic trace-based simulations).

- A WiGig trace with an experimental setup, in a non-line of sight condition with multiple reflections;
- An 802.11n trace on the 5.75 GHz band in an office with several other access points and active cross-traffic;
- An LTE trace captured on a commercial network while driving in Newark, NJ.

These traces have a higher average capacity than the synthetic ones, but the WiGig trace has a strong risk of sudden drops, while the LTE and 802.11n traces have a high variance.

A. JUMP stability analysis

JUMP can provide reliability as long as the objective is achievable, but quickly becomes unstable if it is not, as the additional FEC packets increase the probability of creating a queue and lead to a reliability collapse. To avoid this issue, it would be beneficial to set a lower reliability: when the network cannot cope with the application's demands, JUMP tends to add more FEC, making the system more unstable. While this is still the correct behavior for the protocol, as HOP can report to the application that its demands are unsustainable, the design of a limiting mechanism to make the scheme fail more gracefully is an interesting avenue for future research. We expect LEAP to be immune from reliability collapse, as it is limited by the stricter stability constraint in (8).

In order to show the instability problem, we consider a measure of packet stability in the balanced synthetic scenario, giving JUMP the exact capacity distribution. If we assume the process is time-invariant and IID, the state of the system is given by the vector $\mathbf{q}(t) \in \mathbb{R}^m$, whose elements represent the queuing delay on each path. We denote the schedule that JUMP derives in any state as $\mathbf{s}(\mathbf{q}(t))$, and derive the value of $q_i(t)$ by applying (23):

$$q_i(t, C_i) = \max\left(0, q_i\left(t - T_b + \frac{\tau_{\min}}{2}\right). + \frac{L s_i(\mathbf{q}(t - T_b))}{C_i} - T_b + \frac{\tau_{\min}}{2}\right), \tag{35}$$

assuming that the capacity C_i is constant over a period T_b. Given a schedule $\mathbf{s}(\mathbf{q}(t))$, we now define the associated coding rate $R_C(\mathbf{q}(t))$ as

$$R_C(\mathbf{q}(t)) = \frac{k}{\sum_{i=1}^{m} s_i(\mathbf{q}(t))}. \tag{36}$$

Using the conditions in (27), it naturally follows that a longer queue results in a lower coding rate. In turn, a larger number of redundancy packets increase the probability of having a longer queue, resulting in a positive feedback loop. We can define a stability criterion:

$$S(\mathbf{q}(t), \mathbf{q}(t + Tb)) = u\left(R_C(\mathbf{q}(t)) - R_C(\mathbf{q}(t + T_b))\right). \tag{37}$$

If the redundancy does not increase, the system stays in a state that is at least as favorable as the previous one. Naturally, stability is not certain, since capacity is a stochastic process, but we can compute the stability probability $P_S(\mathbf{q}(t))$:

$$P_S(\mathbf{q}(t)) = \int_{(\mathbb{R}^+)^m} S(\mathbf{q}(t), \mathbf{q}(t + T_b, \mathbf{C})) \prod_{i=1}^{m} \varphi_i(C_i)\, d\mathbf{C}, \tag{38}$$

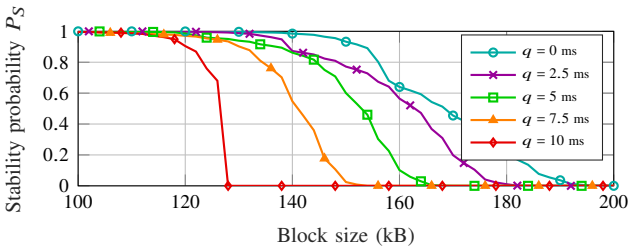

Fig. 3: Stability for different values of initial queuing.

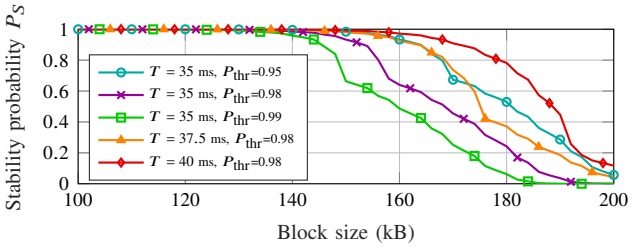

Fig. 4: Stability for different values of the deadline and reliability.

where $\varphi_i(x) = \varphi\left(\frac{x-\mu_i}{\sigma_i}\right)$. Since the coding rate function is not simple, we perform the integration numerically. Figure 3 shows the stability probability as a function of the block size for different values of the initial queuing time (which was the same for both paths), confirming that a longer initial queue indeed increases instability. The resulting positive feedback loop can sharply decrease reliability, as each increase in the queue takes the system to a more unstable state, making further increases more likely. Figure 4 shows the stability probability for different values of the deadline T and the reliability requirement P_{thr}, confirming the intuitive principle that tighter deadlines and higher reliability requirements make the system more unstable by reducing the margin of error in noisy channels.

B. Simulation results: synthetic traces

Figure 5 shows the performance of the protocols in the balanced scenario as a function of the application block size. The plot shows the average performance over 10 simulations, each with a duration of 30 seconds, with error bars representing the 95% confidence interval. It is easy to see that HOP is the only protocol that can guarantee that data will be delivered on time with the required reliability. As expected, the DEMS scheduler can use capacity estimation to slightly outperform DAPS and STTF, and they all significantly outperform the simple LowRTT scheduler, which is not shown in the figure, as it has a failure probability higher than 0.1 even with a block size of just 80 kB. However, none of these schedulers is able to guarantee 98% reliability, even for small block sizes, while HOP can guarantee 98% reliability with 145 kB blocks, which correspond to a goodput of 46.4 Mb/s, more than 90% of the theoretically achievable goodput. In this case, the scheduler's performance is very close to the ideal JUMP's, showing that SKIP works reliably. The confidence interval confirms that the results are not an artifact, but hold across several traces.

It is interesting to note that HOP is very close to the theoretical bound as long as the objective is achievable, but quickly degrades after that, as the additional FEC packets

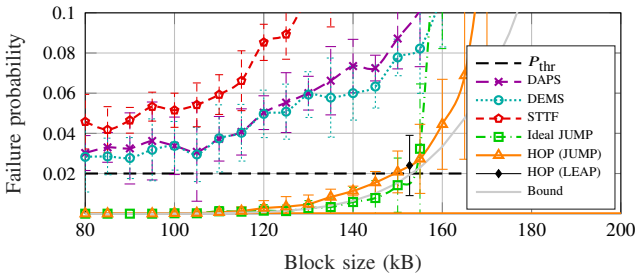

Fig. 5: Performance of the protocols in the *balanced* scenario.

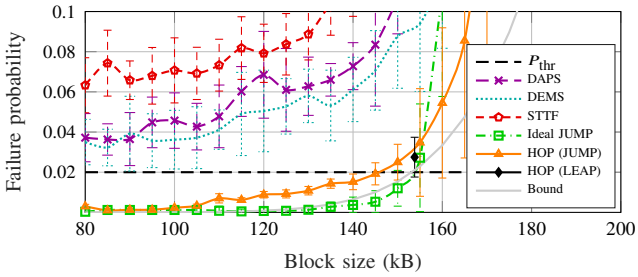

Fig. 6: Performance of the protocols in the *unbalanced* scenario.

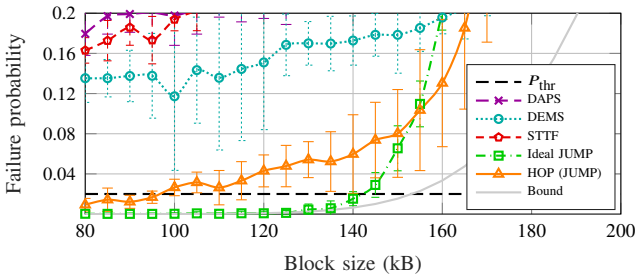

Fig. 7: Performance of the protocols in the *highly unbalanced* scenario.

increase the probability of creating a queue and lead to a reliability collapse. To avoid this issue, it would be beneficial to set a lower reliability, avoiding the reliability collapse described in the previous section. The design of a limiting mechanism to make the scheme fail more gracefully is an interesting avenue for future research. This is confirmed by looking at the number of redundancy packets injected by HOP: for blocks up to 145 kB, the redundancy overhead hovers around 5% of the block size, while it rapidly grows to over 10% if the blocks are larger than 150 kB. However, the redundancy overhead is limited to less than 20% of the goodput in all scenarios.

The situation is more complex in the unbalanced scenario. As Fig. 6 shows, all realistic protocols have more failures, as the schedulers and congestion control schemes struggle to cope with the imbalance. As in the previous case, LowRTT is not even shown in the figure, as its failure rate is always above 0.1. The difference between DAPS and DEMS is slightly larger, but both have more failures, and STTF performs worse than either, as it just considers expected delivery time without trying to balance the traffic on the two paths. The HOP protocol can guarantee, within the 95% confidence interval, that 98% of blocks will be on time for block sizes up to 140 kB (with a maximum goodput of 44.8 Mb/s, slightly over 85% of the theoretical bound). Note that, while the scheduler with perfect capacity estimation is not affected, HOP's performance

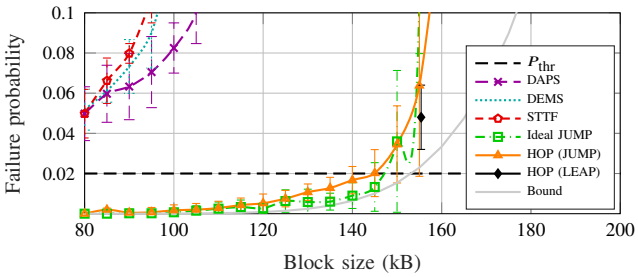

Fig. 8: Performance of the protocols in the *RTT unbalanced* scenario.

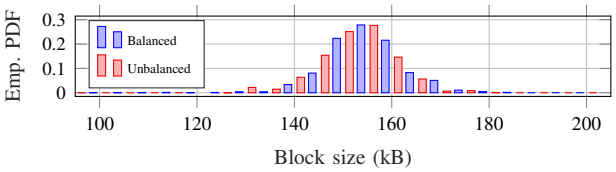

Fig. 9: Block size histogram of the LEAP scheduler in the balanced and unbalanced scenarios.

is slightly degraded by the imperfections in the capacity estimation. At HOP's limit, DAPS and DEMS perform about 3 to 4 times worse in this scenario. Furthermore, HOP is self-correcting, as errors in the capacity estimation can never lead to a delay far beyond the latency bound: the stability limits in the scheduler ensure it, as it will reduce the number of scheduled packets on a path as soon as a queue builds up.

The third scenario we consider is the *highly unbalanced* one shown in Fig. 7, with a strongly dominant link; in this case, even HOP is hardly able to track the lower-capacity channel, which is an order of magnitude smaller than the other. This is probably due to interactions between the scheduler and the SKIP estimation mechanism, as the scheduler with ideal inputs still performs in the same way as for the previous channels. However, the gain from using our QoS-oriented scheme is even clearer in this scenario: DAPS, STTF, and DEMS are unable to maintain a failure probability below 0.1 even with 80 kB blocks, and HOP has 4 times fewer failures than DEMS and 6 times fewer than DAPS and STTF. As above, LowRTT is not even shown in the plot, as its failure rate is always higher than 0.2.

In the *RTT unbalanced* scenario, the performance gains of HOP are even more evident: while it is slightly affected by having two paths with different RTTs, and it has a steeper increase in the failure probability if the blocks are larger than 140 kB, it is still very close to the optimal bound until that size. On the other hand, the three legacy algorithms completely fail to have a failure probability below 0.1 if the block sizes are larger than 100 kB. In this case, HOP can have 10 to 20 times fewer failures than traditional schedulers.

Finally, we present the results for the LEAP scheduler, with a guard parameter $\alpha = 1.25$. In the balanced, unbalanced, and RTT unbalanced scenarios, LEAP had slightly more failures than the 2% threshold, but it maintained an average block size between 150 kB and 160 kB. This operating point is consistent with the performance of the ideal JUMP scheduler, and better than the real JUMP curve, but LEAP is still slightly too aggressive. The full histogram of block sizes in the balanced and unbalanced scenario is shown in Fig. 9: as the two curves

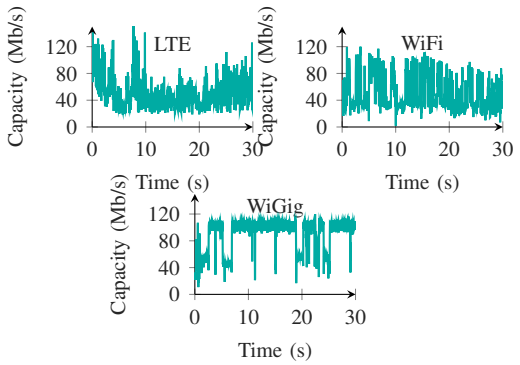

Fig. 10: Capacity traces used in the realistic simulations.

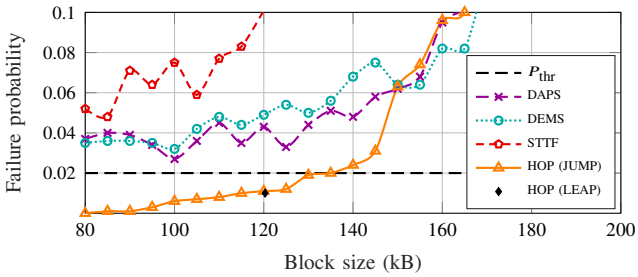

Fig. 11: Performance of the protocols in the WiGig/WiFi scenario.

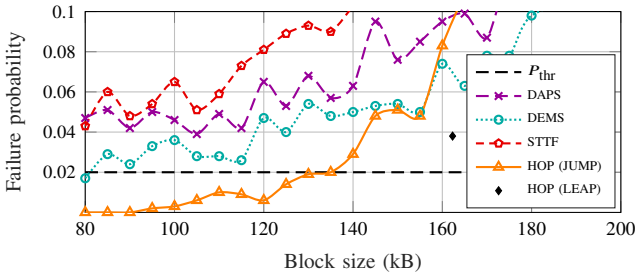

Fig. 12: Performance of the protocols in the LTE/WiGig scenario.

are similar, we confirm that HOP can adapt to several scenarios with either scheduler, as long as the imbalance between the paths is not too big. As predicted, LEAP does not suffer from reliability collapse, dynamically adjusting the goodput to the capacity. In the highly unbalanced scenario, LEAP's performance is not shown, as its average block size is lower than 80 kB. In this extreme case, the default value $\alpha = 1.25$ is too large, and SKIP underestimates the capacity.

C. Simulation results: live traces

We now show simulations performed over three live traces over different Radio Access Technologies (RATs). The three traces are shown in Fig. 10, and have a higher average capacity than the synthetic ones, but they also have stronger dips and a less regular pattern: the WiGig trace has the highest capacity, but it often has sharp drops in the capacity due to blockages.

Figure 11 shows the protocol's performance in the WiGig/WiFi scenario. In this case, as in all the real trace experiments, only one 30 s simulation was run, due to the limited availability of capacity traces. The scenario is not significantly more challenging than the unbalanced one, and the protocols even perform slightly better. As for the synthetic scenarios, the three legacy schemes are far from the required 98% reliability even for small block sizes, while HOP can for blocks of up to 130 kB. In general, HOP's failure rate is less than half of DAPS's, until the quick performance degradation when the QoS constraints are unsustainable. DEMS and particularly STTF perform worse than DAPS in this scenario, as the DEMS FEC adjustment seems to suffer when the two paths have a large capacity difference, while STTF cannot compensate for quick changes in the capacity. DEMS outperforms DAPS for all block sizes in the LTE/WiGig scenario, as shown in Fig. 12, while STTF still has a far higher error than either DAPS or DEMS. However, HOP is still the only one to reach a 98% reliability, with less than half of DEMS's failures.

The WiFi/LTE scenario is the most challenging, as shown in Fig. 13: in this case, it is hard to achieve reliability, and the reliable low-latency throughput is much lower. However, HOP still significantly outperforms DEMS, STTF, and DAPS for smaller blocks. The performance degradation after the required reliability becomes unreachable makes it underperform for large blocks, but the legacy algorithms also have a high failure rate in this case: applications should adjust their demands accordingly to avoid this outcome by lowering their reliability requirements, increasing the maximum latency, or using compression to reduce the block size.

The LEAP scheduler confirms the results of the synthetic traces: in the WiGig/WiFi scenario, it operates close to the JUMP operating point with a failure probability of approximately 1%, while in the WiGig/LTE scenario its failure rate is close to 4%, with a far higher goodput than JUMP manages at the same rate. In the WiFi/LTE scenario, the LEAP failure rate is 4%, but the average block size for is just 75 kB. LEAP's relative unreliability can be solved by adapting its setting to the observed failure rate. In all practical scenarios, HOP can deliver a far higher reliability than even optimized MPTCP versions. It is the only protocol that achieves 98% reliability (or even more than 99%, albeit with a lower throughput) and, more importantly, controls the trade-off between latency, reliability and throughput. In this way, it provides an entirely new service, letting applications set their QoS requirements and informing them when they cannot be met.

Overall, HOP manages to significantly outperform existing approaches in all scenarios, suffering only when the imbalance between the two paths is extremely large, but still managing to achieve far better performance. This is due to its ability to dynamically manage FEC, striking a balance between protecting the transmission from capacity fluctuation on either path and maintaining a short standing queue by minimizing the load on the connections. SKIP might be the critical issue when the protocol misses its reliability goal, as a lower capacity would perhaps require a slower estimation of the path capacity. If fewer packets are used to compute the RTT gradient, the noisier result increases the uncertainty of the estimate, making the system likelier to add more redundancy and cause self-queuing delay. However, we remark again that the simplifying assumptions made in the design hold remarkably well under realistic conditions, as we showed in the last plots: HOP can reduce the number of late blocks by an order of magnitude in most considered scenarios. The dynamic adaptation of the protocol can correct these imperfections, providing flexibility

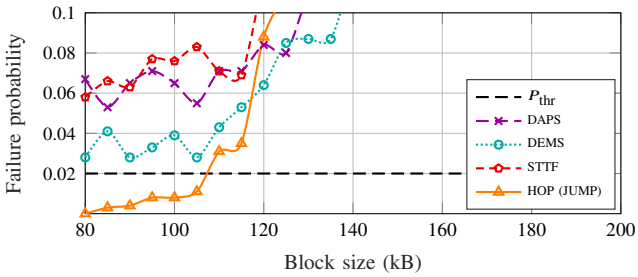

Fig. 13: Performance of the protocols in the WiFi/LTE scenario.

to highly different network environments.

VII. Limitations and Future Work

While the results presented in this work demonstrate HOP's effectiveness at providing QoS in highly dynamic wireless scenarios, maintaining a low latency even for throughput-intense applications, the current version of the protocol still has some limitations that need to be addressed. While our protocol can outperform legacy solutions for high-throughput real-time applications, it is not a one-size-fits-all solution to congestion and end-to-end delay, and its operation parameters must be kept in mind.

Firstly, its reliance on private buffers means that it might have significant fairness issues if it is deployed in a legacy network with shared buffers, and its interactions with the numerous standard TCP versions currently deployed [26] need to be better understood before doing so. In any case, the design of the protocol for low-latency traffic means that its deployment is not advised in scenarios with long RTTs. The deployment of HOP in the wild would also require careful study of the interactions with middleboxes and performance enhancement strategies at the lower layers, some of which are enabled on several networks. The use of the protocol for applications with non-uniform traffic flows, such as encoded video, also requires further study, and a future version of the protocol might exploit the temporal patterns to improve its adaptability.

Secondly, handovers and changes in one or both routes over which HOP operates are an interesting issue in cellular networks, particularly when mobility is involved: while the protocol has not been tested in these kinds of scenarios, we expect SKIP to be able to recover gracefully thanks to its ability to handle sudden changes in the link statistics. Further evaluation on this point will be the focus of future works.

Finally, we remark that HOP is meant for longer flows, such as video or AR data, which span multiple seconds. The SKIP mechanism operates in a similar way to TCP's slow start in its initial phase, and although it could always reach steady-state performance within 2 seconds in our simulation (and reach the channel capacity in less than 1 second), it was not designed for short-lived flows such as web browsing or machine-type traffic, and its performance in those scenarios has not been tested. While such an extension of HOP would be interesting, it is outside the scope of the current work.

VIII. Conclusion

In this work, we presented HOP, a QoS-oriented multipath transport protocol that can meet explicit reliability and latency constraints for throughput-intensive applications. We showed that our protocol far outperforms the state of the art in terms of reliability in trace-based simulations, both with synthetic and real wireless network capacity traces. Furthermore, the HOP protocol can work at different points in the trade-off between latency and reliability, meeting the demands of applications when the network can support them.

The protocol opens several possible avenues of future research: firstly, while the Kalman filter can track capacity effectively, more complex models may be able to more accurately follow the dynamics of the network and improve the performance of the overall protocol. The implementation of a less bursty version of SKIP, which would need to use more advanced filters to deal with the send rate limit, would also be interesting, as would its combination with BBR, whose paced delivery mechanism is friendlier to other flows than HOP's bursty sending pattern, but less reliable as a result. Its use for other types of applications, which could have variable block sizes and inter-block times, would also be interesting. Finally, a wider field of future research is the design of a controller protocol, which dynamically adjusts the QoS constraints to the network scenario, ensuring the best possible service.

References

[1] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, "What will 5G be?" *IEEE J. on Sel. Areas in Comm.*, vol. 32, no. 6, pp. 1065–1082, Jun. 2014.

[2] C. J. Hansen, "WiGiG: Multi-gigabit wireless communications in the 60 GHz band," *IEEE Wireless Comm.*, vol. 18, no. 6, Dec. 2011.

[3] S. Ferlin, T. Dreibholz, and Ö. Alay, "Multi-path transport over heterogeneous wireless networks: Does it really pay off?" in *Global Comm. Conf. (GLOBECOM)*. IEEE, Dec. 2014, pp. 4807–4813.

[4] S. Kucera, M. Buddikhot, and K. Fahmi, "Multi-path data communications," Patent WO 2019/06 368 A1, Apr., 2019.

[5] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The QUIC transport protocol: Design and Internet-scale deployment," in *Conf. of the SIG on Data Comm. (SIGCOMM)*. ACM, Aug. 2017, pp. 183–196.

[6] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, and C. Paasch, "TCP extensions for multipath operation with multiple addresses," IETF, RFC 6824, Jan. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6824.txt

[7] 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 *9th Conf. on Net. Sys. Design and Impl. (NSDI)*. USENIX, Apr. 2012, pp. 29–29.

[8] B. Hesmans, F. Duchene, C. Paasch, G. Detal, and O. Bonaventure, "Are TCP extensions middlebox-proof?" in *Workshop on Hot topics in Middleboxes and Net. Func. Virtualization (HotMiddlebox)*. ACM, Dec. 2013, pp. 37–42.

[9] M. Scharf and A. Ford, "Multipath TCP (MPTCP) application interface considerations," IETF, RFC 6897, Mar. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6897.txt

[10] O. Bonaventure, C. Paasch, and G. Detal, "Use cases and operational experience with Multipath TCP," IETF, RFC 8041, Jan. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8041.txt

[11] B. Han, F. Qian, and L. Ji, "When should we surf the mobile web using both WiFi and cellular?" in *5th SIGCOMM Workshop on All Things Cellular: Op., Appl. and Challenges (AllThingsCellular)*. ACM, Oct. 2016, pp. 7–12.

[12] M. Polese, R. Jana, and M. Zorzi, "TCP and MP-TCP in 5G mmWave networks," *IEEE Internet Comp.*, vol. 21, no. 5, pp. 12–19, Sep. 2017.

[13] M. Li, A. Lukyanenko, S. Tarkoma, Y. Cui, and A. Ylä-Jääski, "Tolerating path heterogeneity in Multipath TCP with bounded receive buffers," in *SIGMETRICS Perf. Eval. Rev.*, vol. 41, no. 1. ACM, Jun. 2013, pp. 375–376.

[14] K. Nguyen, G. P. Villardi, M. G. Kibria, K. Ishizu, F. Kojima, and H. Shinbo, "An enhancement of Multipath TCP performance in lossy wireless net." in *41st Conf. on Local Comp. Net. Workshops (LCN Workshops)*. IEEE, Nov. 2016, pp. 187–191.

[15] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Op. Sys. Review*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[16] A. Veres and M. Boda, "The chaotic nature of TCP congestion control," in *19th Ann. Joint Conf. of the IEEE Comp. and Comm. Soc. (INFOCOM)*, vol. 3. IEEE, Mar. 2000, pp. 1715–1723.

[17] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *ACM Queue*, vol. 9, no. 11, p. 40, Apr. 2011.

[18] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *ACM Queue*, vol. 14, no. 5, p. 50, Oct. 2016.

[19] M. Becke, T. Dreibholz, H. Adhari, and E. P. Rathgeb, "On the fairness of transport protocols in a multi-path environment," in *Int. Conf. on Comm. (ICC)*. IEEE, Jun. 2012, pp. 2666–2672.

[20] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "MPTCP is not Pareto-optimal: performance issues and a possible solution," *IEEE/ACM Trans. on Net.*, vol. 21, no. 5, pp. 1651–1665, Oct. 2013.

[21] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for Multipath TCP." in *8th Symp. on Net. Sys. Design and Impl. (NSDI)*, vol. 11. USENIX, Mar. 2011, pp. 99–112.

[22] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet," *IEEE/ACM Trans. on Net.*, vol. 14, no. 6, pp. 1260–1271, Dec. 2006.

[23] A. Walid, J. Hwang, Q. Peng, and S. Low, "Balanced Linked Adaptation congestion control algorithm for MPTCP," IETF, Working Draft, Jul. 2014. [Online]. Available: http://www.ietf.org/internet-drafts/draft-walid-mptcp-congestion-control-00.txt

[24] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of Multipath TCP performance over wireless networks," in *SIGCOMM Internet Measurement Conf. (IMC)*. ACM, Oct. 2013, pp. 455–468.

[25] M. Xu, Y. Cao, and E. Dong, "Delay-based congestion control for MPTCP," IETF, Working Draft, Jan. 2017. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-xu-mptcp-congestion-control-05

[26] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, and B. Leong, "The great Internet TCP congestion control census," in *SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*. ACM, Jun., pp. 59–60.

[27] T. Zhu, X. Qin, L. Chen, X. Chen, and G. Wei, "wBBR: A bottleneck estimation-based congestion control for multipath TCP," in *88th Vehicular Tech. Conf. (VTC-Fall)*. IEEE, Aug. 2018.

[28] J. Hwang and J. Yoo, "Packet scheduling for Multipath TCP," in *7th Int. Conf. on Ubiquitous and Future Net. (ICUFN)*. IEEE, Jul. 2015, pp. 177–179.

[29] D. Ni, K. Xue, P. Hong, and S. Shen, "Fine-grained forward prediction based dynamic packet scheduling mechanism for multipath TCP in lossy networks," in *23rd Int. Conf. on Comp. Comm. and Net. (ICCCN)*. IEEE, Aug. 2014, pp. 1–7.

[30] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath TCP schedulers," in *SIGCOMM Workshop on Capacity Sharing (CSWS)*. ACM, Aug. 2014, pp. 27–32.

[31] K. W. Choi, Y. S. Cho, J. W. Lee, S. M. Cho, J. Choi *et al.*, "Optimal load balancing scheduler for MPTCP-based bandwidth aggregation in heterogeneous wireless environments," *Comp. Comm.*, vol. 112, pp. 116–130, Nov. 2017.

[32] E. Dong, M. Xu, X. Fu, and Y. Cao, "LAMPS: A loss aware scheduler for Multipath TCP over highly lossy net." in *42nd Conf. on Local Comp. Net. (LCN)*. IEEE, Oct. 2017, pp. 1–9.

[33] H. Shi, Y. Cui, X. Wang, Y. Hu, M. Dai, F. Wang, and K. Zheng, "STMS: Improving MPTCP throughput under Heterogeneous Net." in *Ann. Tech. Conf. (ATC)*. USENIX, Jul. 2018, pp. 719–730.

[34] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *Int. Conf. on Comm. (ICC)*. IEEE, Jun. 2014, pp. 1222–1227.

[35] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "FMTCP: A fountain code-based Multipath Transmission Control Protocol," *IEEE/ACM Trans. on Net.*, vol. 23, no. 2, pp. 465–478, Apr. 2015.

[36] P. Hurtig, K.-J. Grinnemo, A. Brunstrom, S. Ferlin, Ö. Alay, and N. Kuhn, "Low-latency scheduling in MPTCP," *IEEE/ACM Trans. on Net.*, vol. 27, no. 1, pp. 302–315, Dec. 2018.

[37] A. Garcia-Saavedra, M. Karzand, and D. J. Leith, "Low delay random linear coding and scheduling over multiple interfaces," *IEEE Trans. on Mobile Comp.*, vol. 16, no. 11, pp. 3100–3114, Mar. 2017.

[38] Y. E. Guo, A. Nikravesh, Z. M. Mao, F. Qian, and S. Sen, "Accelerating multipath transport through balanced subflow completion," in *23rd Int. Conf. on Mobile Comp. and Net. (MobiCom)*. ACM, Oct. 2017, pp. 141–153.

[39] F. Chiariotti, S. Kucera, A. Zanella, and H. Claussen, "Analysis and design of a latency control protocol for multi-path data delivery with pre-defined QoS guarantees," *IEEE/ACM Trans. on Net.*, vol. 27, no. 3, pp. 1165–1178, Jun. 2019.

[40] G. Sarwar, P.-U. Tournoux, R. Boreli, and E. Lochin, "eCMT-SCTP: Improving performance of multipath SCTP with erasure coding over lossy links," in *38th Ann. Conf. on Local Comp. Net.* IEEE, Oct. 2013, pp. 476–483.

[41] M. Wang, C. Xu, X. Chen, H. Hao, L. Zhong, and D. O. Wu, "Design of multipath transmission control for information-centric Internet of Things: A distributed stochastic optimization framework," *IEEE Internet of Things J.*, vol. 6, no. 6, pp. 9475–9488, Jul. 2019.

[42] M. Polese, F. Chiariotti, E. Bonetto, F. Rigotto, A. Zanella, and M. Zorzi, "A survey on recent advances in transport layer protocols," *IEEE Comm. Surveys and Tut.*, vol. 24, no. 4, pp. 3584–3608, Aug. 2019.

[43] N. Kuhn, E. Lochin, F. Michel, and M. Welzl, "Coding and congestion control in transport," IETF NWCRG, IETF Draft, Oct. 2020. [Online]. Available: https://tools.ietf.org/id/draft-irtf-nwcrg-coding-and-congestion-04.html

[44] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Net. slicing for 5G with SDN/NFV: Concepts, architectures, and challenges," *IEEE Comm. Mag.*, vol. 55, no. 5, pp. 80–87, May 2017.

[45] Q. Wang, J. Alcaraz-Calero, R. Ricart-Sanchez, M. B. Weiss, A. Gavras, N. Nikaein, X. Vasilakos, B. Giacomo, G. Pietro, M. Roddy *et al.*, "Enable advanced QoS-aware network slicing in 5G networks for slice-based media use cases," *IEEE Transactions on Broadcasting*, vol. 65, no. 2, pp. 444–453, Mar. 2019.

[46] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM SIGCOMM Comp. Comm. Review*, vol. 27, no. 2, pp. 24–36, Apr. 1997.

[47] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *10th Symp. on Net. Sys. Design and Impl. (NSDI)*. USENIX, Apr. 2013, pp. 459–471.

[48] H. Emmons and M. Pinedo, "Scheduling stochastic jobs with due dates on parallel machines," *Eur. J. of Op. Res.*, vol. 47, no. 1, pp. 49–55, Jul. 1990.

[49] A. Allahverdi and J. Mittenthal, "Scheduling on M parallel machines subject to random breakdowns to minimize expected mean flow time," *Naval Res. Logistics (NRL)*, vol. 41, no. 5, pp. 677–682, Aug. 1994.

[50] J. Jaffe, "Flow control power is nondecentralizable," *IEEE Trans. on Comm.*, vol. 29, no. 9, pp. 1301–1306, Sep. 1981.

[51] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

[52] C. Price, "An analysis of the divergence problem in the Kalman filter," *IEEE Trans. on Aut. Control*, vol. 13, no. 6, pp. 699–702, Dec. 1968.

[53] R. Mehra, "On the identification of variances and adaptive Kalman filtering," *IEEE Trans. on Aut. Control*, vol. 15, no. 2, pp. 175–184, Apr. 1970.

[54] W. Gao, J. Li, G. Zhou, and Q. Li, "Adaptive Kalman filtering with recursive noise estimator for integrated SINS/DVL systems," *The J. of Nav.*, vol. 68, no. 1, pp. 142–161, Jan. 2015.

[55] H.-J. Kim, "On the distribution and its properties of the sum of a normal and a doubly truncated normal," *Comm. for Stat. Appl. and Methods*, vol. 13, no. 2, pp. 255–266, Aug. 2006.

[56] H. Joe, "Approximations to multivariate normal rectangle probabilities based on conditional expectations," *J. of the American Stat. Ass.*, vol. 90, no. 431, pp. 957–964, Sep. 1995.

[57] B. Allik, C. Miller, M. J. Piovoso, and R. Zurakowski, "The Tobit Kalman filter: an estimator for censored measurements," *IEEE Trans. on Control Sys. Tech.*, vol. 24, no. 1, pp. 365–371, Jun. 2015.

Dr. Federico Chiariotti [S'15 M'19] is currently a post-doctoral researcher at the Department of Electronic Systems, Aalborg University, Denmark. He

received his Ph.D. in information engineering in 2019 from the University of Padova, Italy. He received the bachelor's and master's degrees in telecommunication engineering from the University of Padova, in 2013 and 2015, respectively. In 2017 and 2018, he was a Research Intern with Nokia Bell Labs, Dublin. He has authored over 40 published papers on wireless networks and the use of artificial intelligence techniques to improve their performance. He was a recipient of the Best Paper Award at several conferences, including the 2020 IEEE INFOCOM WCNEE Workshop. His current research interests include network applications of machine learning, transport layer protocols, Smart Cities, bike sharing system optimization, Age of Information, and adaptive video streaming.

Prof. Andrea Zanella [S'98-M'01-SM'13] is a Full Professor at the Department of Information Engineering (DEI), University of Padova, Italy. He received the Laurea degree in Computer Engineering in 1998 from the same University and the PhD in 2001. During 2000, he spent 9 months with Prof. Mario Gerla's research team at the University of California, Los Angeles (UCLA). Andrea Zanella is one of the coordinators of the SIGnals and NETworking (SIGNET) research lab. His long-established research activities are in the fields of protocol design, optimization, and performance evaluation of wired and wireless networks. He has been serving as Technical Area Editor for the IEEE Internet of Things Journal, and Associate Editor for the IEEE Transactions on Cognitive Communications and Networking, IEEE Communications Surveys and Tutorials, and Digital Communications and Networks.

Dr. Stepan Kucera [SM'13] received Ph.D. degree in Informatics from the Graduate School of Informatics, Kyoto University, Kyoto, Japan, in 2008. Currently, he serves as Senior 5G Expert and 3GPP RAN2 delegate at Nokia Munich in Germany where he is responsible for driving research and development of advanced 5G NR concepts as well as their 3GPP standardization, mainly in areas of industrial IoT, V2X sidelink and enhanced positioning. In the past decade, he developed at Nokia innovative networking technologies that became the basis for new commercial products and have been demonstrated to top-tier business customers worldwide and in industry shows. Impactful examples include multi-connectivity with user-defined latency/reliability guarantees, zero-touch machine-designed self-optimization for large-scale networks, light-based Gigabit mobile access, optimal interference management and medium access for heterogeneous networks. He is the recipient of multiple professional awards, among others *Nokia Top Inventor 2018*, *Nokia UK&I Top Inventor of All Times*, *Hummies Gold Award for Best Human-Competitive Design 2019*, and *Irish Laboratory Scientist of the Year Award 2018*. He also served as work package leader, primary beneficiary/investigator, board member in large-scale research projects totalling 15+ Mil. EUR (e.g., MINTS, ENLIGHTEM, ELIOT), as well as founded and managed advanced research projects with six major universities in EU and Japan. He has filed 50+ patents, published 70+ book chapters, transactions, and conference papers in peer-reviewed ACM/IEEE venues. He is a Senior IEEE Member and actively serves on technical boards of major ACM/IEEE journals and conferences.

Dr. Kariem Fahmi is currently a software architect at Nokia, Ireland. He received his PhD in Computer Science in 2021 from the Trinity College Dublin, Ireland. He received his bachelor's in Computer Science from Qatar University in 2015. During his PhD, he collaborated closely with Nokia Bell Labs on projects related to wireless multi-connectivity, real time communication and quality of service, and received a certificate of out-standing-achievement for his contribution to Bell Labs multi-connectivity innovation. His PhD work became the foundation of new Nokia commercial products in the area of wireless vehicle-to-ground communication. He has filed over 10 patents related to computer networks. His research interests include wireless multi-connectivity, real-time communication and vehicle-to-ground communication.

Dr. Holger Claussen is Head of the Wireless Communications Laboratory at Tyndall National Institute where he is building up research teams in the area of RF, Access, Protocols, AI, and Quantum Systems to invent the future of Wireless Communication Networks. Previously he led the Wireless Communications Department of Nokia Bell Labs located in Ireland and the US. In this role, he and his team innovated in all areas related to future evolution, deployment, and operation of wireless networks to enable exponential growth in mobile data traffic and reliable low latency communications. His research in this domain has been commercialized in Nokia's (formerly Alcatel-Lucent's) Small Cell product portfolio and continues to have significant impact. He received the 2014 World Technology Award in the individual category Communications Technologies for innovative work of "the greatest likely long-term significance". Prior to this, Holger directed research in the areas of self-managing networks to enable the first large scale femtocell deployments. Holger joined Bell Labs in 2004, where he began his research in the areas of network optimization, cellular architectures, and improving energy efficiency of networks. Holger received his Ph.D. degree in signal processing for digital communications from the University of Edinburgh, United Kingdom in 2004. He is author of the book "Small Cell Networks", more than 130 journal and conference publications, 78 granted patent families, and 46 filed patent applications pending. He is Fellow of the World Technology Network, senior member of the IEEE, and member of the IET.

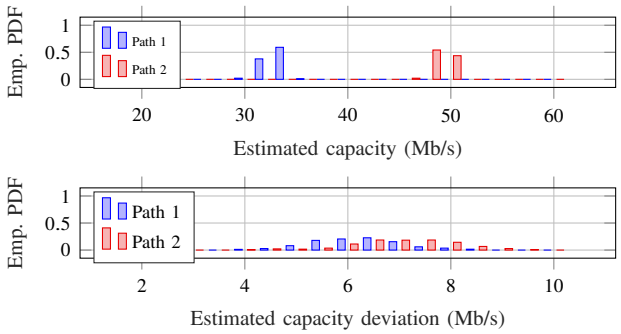

Fig. 14: SKIP estimate quality in the unbalanced case using the JUMP scheduler.

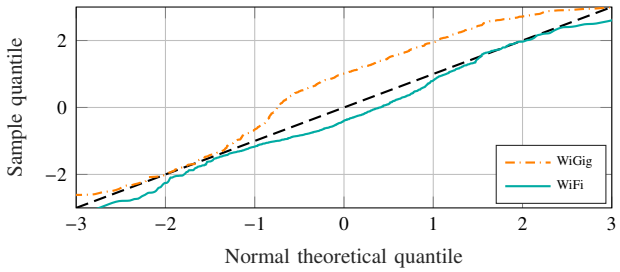

Fig. 15: Q-Q plot of the SKIP estimate quality in the WiGig/WiFi case using the JUMP scheduler with 140 kB blocks.

Appendix

In this Appendix, we go into the functioning of the SKIP mechanism in more depth and draw some practical considerations on its use in real-world scenarios. First, we compute the analytical form of the capacity CDF, for which we used a non-truncated approximation in the practical implementation of the algorithm.

We define the standard deviations of C_1 and $C_{2,\dots,k}$ as σ_1 and $\sigma_{2,\dots,k}$, respectively:

$$\sigma_1 = \sqrt{\mathrm{Var}(C_1)} = \sqrt{(\ell+1)Q + R} \tag{39}$$

$$\sigma_{2,\dots,k} = \sqrt{\mathrm{Var}(C_{2,\dots,k})} = \sqrt{\frac{(k+\ell)(k-1)Q}{2} + R} \tag{40}$$

The sum of the capacities over the k steps is then $C_1 + C_{2,\dots,k}$, or the sum of a doubly truncated Gaussian variable and a truncated (from below) Gaussian variable, which is restricted to the interval $[0, \infty)$. We define the following auxiliary variables:

$$Z_1 = \Phi\left(\frac{C_{\max} - \hat{x}_{1|0}}{\sigma_1}\right) - \Phi\left(-\frac{\hat{x}_{1|0}}{\sigma_1}\right) \tag{41}$$

$$Z_2 = 1 - \Phi\left(\frac{(1-k)\hat{x}_{2|1}}{\sigma_{2,\dots,k}}\right) \tag{42}$$

$$Y_1 = \frac{(x - (k-1)\hat{x}_{2|1})\sigma_1^2 + \hat{x}_{1|0}\sigma_{2,\dots,k}^2}{\sigma_1\sigma_{2,\dots,k}\sqrt{\sigma_1^2 + \sigma_{2,\dots,k}^2}} \tag{43}$$

$$Y_2 = \frac{-(k-1)\hat{x}_{2|1}\sigma_1^2 + (\hat{x}_{1|0} - x)\sigma_{2,\dots,k}^2}{\sigma_1\sigma_{2,\dots,k}\sqrt{\sigma_1^2 + \sigma_{2,\dots,k}^2}} \tag{44}$$

$$Y_3 = \frac{(x - C_{\max} - (k-1)\hat{x}_{2|1})\sigma_1^2 + (\hat{x}_{1|0} - C_{\max})\sigma_{2,\dots,k}^2}{\sigma_1\sigma_{2,\dots,k}\sqrt{\sigma_1^2 + \sigma_{2,\dots,k}^2}} \tag{45}$$

$$W = \sqrt{\frac{2}{\pi}} e^{-\frac{(-x+\hat{x}_{1|0}+(k-1)\hat{x}_{2|1})^2}{k(k+1)Q+2R}}. \tag{46}$$

The piecewise PDF of the sum of the capacities over the k steps is given by

$$p_{C_{1,\dots,k}}(x) = \begin{cases} 0 & x < 0 \\ \frac{2W(\varphi(Y_1) - \varphi(Y_2))}{4Z_1Z_2\sqrt{\frac{k(k+\ell+1)Q}{2}+R}} & x \in [0, C_{\max}) \\ \frac{2W(\varphi(Y_1) - \varphi(Y_3))}{4Z_1Z_2\sqrt{\frac{k(k+\ell+1)Q}{2}+R}} & x \geq C_{\max} \end{cases} \tag{47}$$

We then look at SKIP's tracking performance in the synthetic and realistic traces, looking at how well SKIP does at estimating capacity.

Figure 14 shows the quality of SKIP's estimate in the unbalanced scenario, for a block size of 120 kB: while SKIP is slightly optimistic, estimating a slightly higher average capacity with a slightly lower standard deviation than the real distribution, its accuracy is sufficient for HOP to outperform the other schemes significantly. While the mechanism slightly underestimates variance, it still performs very well in the scenario that matches its capacity model. In a more realistic case, we tried to look at the Q-Q plot of the real capacity against the normalized estimate. If we look at the Kalman update rule in (17), the normalized capacity sample is given by

$$C_n(t) = \frac{C_t - x_t}{\Sigma_{\mathrm{AKF}}}. \tag{48}$$

If the SKIP estimate were perfect, the distribution of $C_n(t)$ would be normal, as the capacity samples would follow the estimated distribution perfectly. Figure 15 shows the Q-Q plot for the capacity samples in the WiGig/WiFi scenario with 140 kB blocks. We removed the capacity deviations of more than 3σ, as these sharp capacity changes are explicitly dealt with by the drop and step mode settings explained in Sec. V-A. The two curves for the two paths are pretty close to the perfect estimate, although the WiFi channel is slightly overestimated. The WiGig curve exhibits a slight rightward skew, which can be explained by its large right tail, as shown in Fig. 10.

These results show that the SKIP capacity mechanism, while not perfect, can provide a pretty good approximation of the capacity distribution in real links, while maintaining a relatively simple model. More complex models would be much more unstable, requiring careful parameter tuning and a different setting for each technology. The use of learning models is a possible avenue of future work, as they would provide a more tailored estimate of future capacity. Another possible improvement would be to exploit cross-layer information, if the node at the bottleneck can make it available: in general, the modularity of HOP means that these innovations can be implemented with no modifications to the scheduler, and even on only part of the paths.

In order to avoid fairness issues due to the bursty nature of HOP, HOP requires the use of separate buffers for each user. While this is always the case in modern cellular networks, separate buffers are not a requirement in 802.11 routers. In order to verify that most recent models implement this policy,

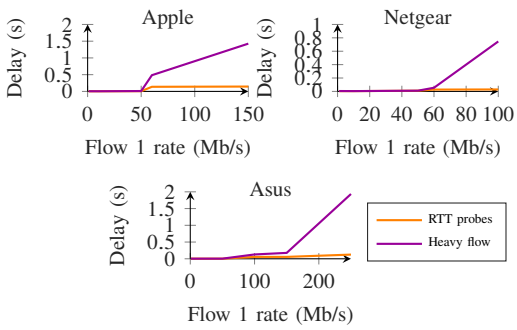

Fig. 16: Results of the buffer analysis on the three tested router models.

we tested three models of home routers from different brands: an Asus Wireless-AC2900 Dual Band, a Netgear Nighthawk x10, and an Apple Airport Extreme 802.11ac (1st generation). We connected a server to each of the Access Points (APs), using two different Ethernet connections with different addresses. Then, we started a high-throughput UDP flow to a client connected to the WiFi through the first interface, varying its sending rate to create a queue. We also periodically sent one-packet UDP RTT probes to the client on the other Ethernet port; if the separate buffer hypothesis is correct, the changes in the heavy flow's send rate should not affect the RTT of the probes, since the packets are not queued together.

As Fig. 16 shows, the delay on both flows is very low as long as the data rate is lower than the wireless link capacity. When the first heavy-throughput flow starts experiencing congestion, its delay sharply rises, but the delay of the probes is almost unchanged. As the first flow's rate increases, the probes' delay slightly increases, but this is most likely due to contention issues on the Medium Access layer, as sharing a buffer would cause the two flows' RTTs to have a similar pattern and increase at the same rate. The results are very similar for all three models, confirming our hypothesis.