SOS: Stochastic Object-aware Scheduler for low delay communication over multiple wireless paths

Kariem Fahmi [1], Douglas J. Leith[1], Stepan Kucera[2], and Holger Claussen[2]

[1]School of Computer Science and Statistics Trinity College Dublin, Dublin 2, Ireland {*kfahmi,doug.leith*}@*tcd.ie*
[2]Nokia Bell Labs, Dublin 15, Ireland {*stepan.kucera,holger.claussen*}@*nokia-bell-labs.com*

***Abstract*—In this paper we consider the task of scheduling packet transmissions amongst multiple paths with uncertain, time-varying delay. We make the observation that the requirement is usually to transmit application layer objects (web pages, images, video frames etc) with low latency, and so it is the object delay rather than the per packet delay which is important. This has fundamental implications for multipath scheduler design. We introduce SOS (Stochastic Object-aware Scheduler), the first multipath scheduler that considers application layer object sizes and their relationship to link uncertainty. We show how to interface SOS with cwnd-based/ack-clocked congestion control (as usually used in TCP) and show up to 150% improvement in the 95% percentile and mean delay vs MPTCP/minRTT.**

I. Introduction

While much attention in 5G has been focused on the physical and link layers, it is increasingly being realized that a wider redesign of network protocols is also needed in order to meet 5G requirements. Transport protocols are of particular relevance, with ETSI recently setting up a working group to study next generation protocols for 5G [1]. Additionally, there have been parallel initiatives to evolve the transport layer such as Google QUIC [2] (currently being developed by the IETF), Coded TCP [3] and the Open Fast Path Alliance [4]. In this paper we consider next generation edge transport architectures of the type illustrated in Figure 1. Traffic to and from client stations is routed via a proxy located close to the network edge (e.g. within a cloudlet). This creates the freedom to implement new transport layer behavior over the path between proxy and clients, which in particular includes the use of multiple wireless paths at the network edge.

Multipath transport protocols have the potential to improve network performance dramatically by utilizing multiple interfaces simultaneously to increase capacity through aggregation, reduce latency and increase reliability. Currently, almost all smart phones are equipped with two radio interfaces (WiFi and Cellular) and the ubiquity of WiFi hot-spots and cellular coverage means that the opportunity for multipath aggregation is almost always there. While multiple commercial entities have begun to seize this opportunity [5], building an efficient, low latency multipath transfer mechanism remains highly challenging [6][7][8]. A primary reason for this is that the transmission delay along each path is typically uncertain and time-varying due to queueing, link layer retransmission, the action of congestion control, etc [9]. Packets sent along different paths therefore frequently arrive out of order and need to be buffered at the receiver to allow in-order delivery,

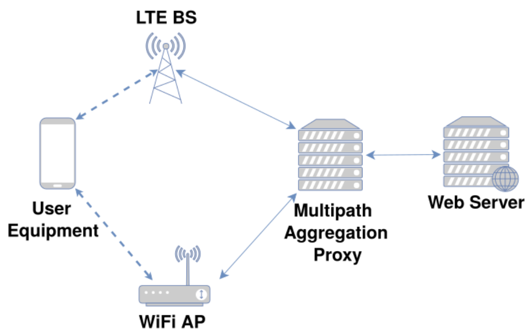


Fig. 1: Schematic of a cloudlet-based edge transport architecture.

leading to head of line blocking and introducing substantial delay. While head of line blocking is not exclusive to multipath, it is potentially much more of a performance bottleneck when multiple paths are used.

In this paper we consider the task of scheduling packet transmissions amongst multiple paths with uncertain, time-varying delay. We make the observation that the requirement is usually to transmit application layer objects (web pages, images, video frames etc) with low latency, and so it is the object delay rather than the per packet delay which is important. We define objects as the atomic unit of data that the application needs to receive fully before any action is taken. Depending on the application, an object can be smaller than a single packet (e.g. VOIP) or it can span hundreds of packets (e.g. UHD video). This observation fundamentally changes both scheduler design and the scope for making use of links with fluctuating delay. Firstly, in-order delivery of packets is no longer important but rather it is the time when all of the packets forming an object are received which is the key quantity of interest. Secondly, when the requirement is to transmit individual packets subject to a delay deadline then it is difficult to make use of a path with highly fluctuating delay since many packets will miss the deadline. However, when the requirement is to transmit an object consisting of many packets then statistical multiplexing means that it is indeed possible to use a path with fluctuating delay while ensuring low object delivery delay with high probability, as we show in more detail later.

Our main contributions are summarized as follows:

1. We propose SOS (Stochastic Object-aware Scheduler), a multi-path scheduler that takes in consideration the size of

the application layer object and the uncertainty of the link delay in order to deliver objects with a reliable bound on delay, making it suitable for latency-sensitive applications. We offer a low complexity implementation that can execute in $O(\log n)$ time for two links, where n is the number of packets in an object. To the best of our knowledge this is the first multipath scheduler that considers application layer object sizes and their relationship to link uncertainty.

2. We show how SOS can interface with cwnd-based congestion control, using Reno as an example, in order to optimize scheduling when an object partition is split across multiple cwnds. We implement this in a user-space network stack, evaluate its performance against MPTCP/minRTT under different link conditions and using different object sizes and show up to a 150% improvement in the 95% percentile and mean delay is achieved.

II. Preliminaries

A. Application Layer Objects

We begin by noting that the number of packets used to transmit an object is a direct proxy for the object size. The underlying mechanism is that an object of size N bytes, such as a video frame, is typically sent by an application using a TCP socket. The networking layer within the sender operating system then segments the object and transmits it over the network as $\lfloor N/MSS \rfloor + 1$ packets, where MSS is the network path maximum segment size. Some applications use UDP, in which case the segmentation is carried out by the application itself, but the focus in the present paper is on objects sent using TCP with its associated reliability and in-order delivery guarantees. Applications employ different sizes of objects/frames depending on the type of the application. For example, Fig 2 plots the measured distribution of HTTP object sizes (measured in packets) for three popular streaming web sites•. Observe that YouTube video object sizes can be as large as 2500 packets. Similarly, TLS (Transport Layer Security protocol) employs its own framing on top of TCP/UDP which can span up to 13 packets [10] in v1.2, with data unable to be decrypted until a frame is received completely. Compression algorithms such as gzip behave in a similar way.

These observations are pertinent because they mean that the relevant delay for applications is the time it takes to deliver an application object, rather than the time taken to deliver individual packets. Hence, for example, application latency may not be improved by in-order packet delivery (since what matters is that all of the packets forming an object are received, not their order of arrival) and so schedulers that minimize usage of variable-delay links that cause head of line blocking might be inadvertently hurting application latency by reducing aggregate capacity.

B. Variability of LTE and WiFi Path Delays

Fig. 3 plots example measurements of per packet delay on production WiFi and LTE paths, taken from [9]. These

•Data was collected using HAR files obtained from the Chrome web browser in March, 2018. The object sizes were inferred from the HTTP response content-length field. Youtube video measured is "Ghost Town 8k". NetFlix movie measured is "The Sinner" and a random Twitch stream active at the time was selected.

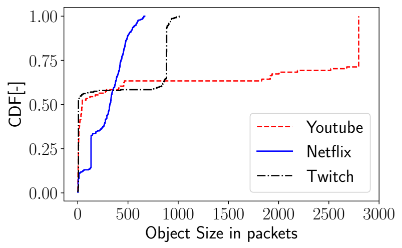


Fig. 2: HTTP object sizes for videos offered by 3 popular streaming websites (Youtube, Netflix and Twitch).

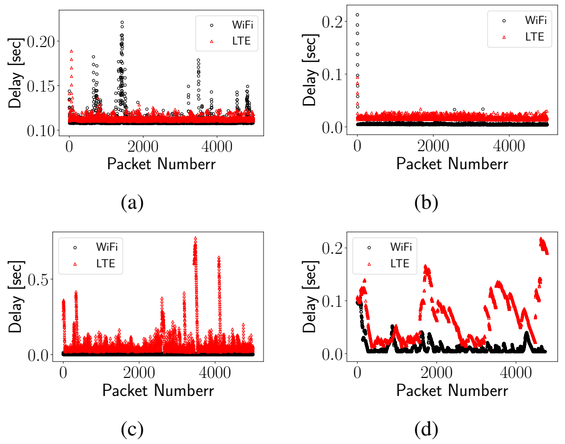


Fig. 3: Packet delay measured while transmitting 5000 UDP packets to a server from 4 different locations in Dublin, Ireland.

measurements are for UDP packets in order to eliminate the influence of TCP congestion control on the variability of the delay, and are measured between a UE with WiFi and LTE interfaces and a server, both located in the same city. It can be seen that a range of delay behaviours are observed, with the delay behaviour of WiFi and LTE often differing significantly. For example, in Fig. 3(a) it can be seen that the WiFi link is on average faster than the LTE link (has lower mean delay) but also has higher variability. Conversely in Figs. 3(c) and 3(d) the LTE path has much higher delay variability than WiFi. The magnitude of the delay fluctuations is also quite variable. For example in Fig. 3(b) the delay fluctuations are around 10-20ms whereas in Fig. 3(c) the LTE delay fluctuations can be as high as 600-800ms.

C. Multipath Schedulers: State of the Art

Perhaps the most well known multipath scheduler is the MPTCP default scheduler, minRTT [11]. This scheduler attempts to send packets out on the link with the lowest RTT (Round Trip Time) first until its cwnd (Congestion Window) is full and then moves to the next lowest RTT link, filling its cwnd. Once all cwnd are full, it will send packets on a link as soon as there is space in its cwnd. Additionally, the scheduler will re-inject packets that are causing HoL blocking on one link into another link and penalize the link that caused the HoL blocking by halving its cwnd [11]. In [12][13][14] and [15] various approaches are proposed for reducing out of order delivery in minRTT. In [12], a Blocking Estimation (BLEST) scheduler is introduced. BLEST addresses the issue of HoL blocking when one path is slower than the other, in which it is possible for the faster link to deliver multiple cwnds worth of packets in the time taken for the slower link to deliver one cwnd worth of packets which, using minRTT, leads to out of order delivery. BLEST estimates the amount of packets that can be delivered on the faster path while the slower path delivers one cwnd of packers. It also accounts for growth of the cwnd on the faster subflow during the transmission. In [13], a Delay Aware Packet scheduler (DAP) is introduced. DAP creates a schedule based on the Lowest Common Multiple (LCM) of forward delays on all paths. For example, if the forward delay on the slower link is 5 times that of the faster link, then DAP will transmit packets 1 to 5 on the faster link and packet 6 on the slow link. One issue with DAP is that once a schedule is created it will not be modified until completed, which causes it to be less reactive than other approaches. In [14], the Out-of-order Transmission for In-order Arrival Scheduler (OTIAS) is introduced. OTIAS will queue packets to paths regardless of whether they have free space in the cwnd. It schedules on a per-packet basis as soon as the packet arrives and creates a queue on each path.

Perhaps the closest work to the present paper in the sense that it takes explicit account of delay variability when making scheduling decisions is [16], which introduces Stochastic Earliest Deadline Path First (SEDPF). SEDPF models inter-packet delay as i.i.d. Gaussian random variables. Using the Gaussian assumption, it calculates the distribution of the total inter-packet delay of in-flight packets on each link with the new packet added. It then evaluates the maximum of multiple sets of Gaussian random variables, using an approximate closed form, each representing the delay of a packet being sent on a particular path. It then selects the set that had the lowest maximum, as that implies the lowest delay.

None of the above schedulers exploit knowledge of application layer objects and their impact on scheduling decisions, which is our focus in the present paper. With the exception of [16], existing schedulers also deal only indirectly with the time-varying and uncertain nature of path delay, namely by utilizing a smoothened average RTT and applying this to all packets sharing the same cwnd[§] when making scheduling decisions.

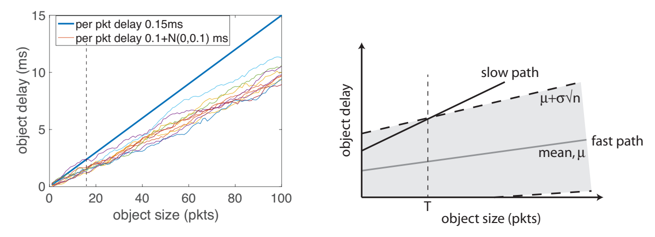
III. Scheduling Objects

A. QoS Over Paths With Time-Varying Delay

Consider two paths, one of which is slower but has consistent delay and the other which is faster but has highly fluctuating delay. When the requirement is to transmit individual packets subject to a delay deadline then it is difficult to make efficient use of the link with fluctuating delay since many packets will miss the deadline. However, when the requirement is to transmit an object consisting of multiple packets subject to an overall deadline then the situation changes fundamentally.

The reason why is illustrated in Figure 4. Figure 4(a) plots sample paths of the object transmission time on a fluctuating faster path vs the object size. It can be seen that

[§]The difference in delay between the head of the cwnd and the tail can, for example, be large when network buffers start to fill.



(a) (b)

Fig. 4: Illustrating time taken to transmit an object over a slower path with fixed delay and over a faster path with fluctuating delay. Plot (a) shows some example paths when the inter-packet delays on the faster path is normally distributed with standard deviation $\sigma = 0.1$ms. The shaded region in plot (b) indicates schematically the mean plus one standard deviation in transmission time for the faster path. The vertical dashed line on both plots indicates the object size above which the delay on the faster path is, with high probability, lower than that of the slower path.

the variance of the transmission time increases with the object size, as expected since the object transmission time is the sum $\sum_{i=1}^{n} T_i$ of the individual packet transmission times T_i and when these are i.i.d this has standard deviation $\sqrt{n}\sigma$ where σ is the standard deviation of the per packet times T_i. However, the transmission time on the slower path scales with n, see Figure 4(b). Hence, for a sufficiently large object size the transmission time will, with high probability, be lower on the faster path despite its highly fluctuating delay.

This basic observation has significant implications. In particular, it means that we can still make efficient use of wireless links with highly fluctuating inter-packet delay (e.g. mmWave, LiFi) while providing controlled quality of service, provided that when scheduling packets across paths we move from consideration of packets individually to consideration of objects (collections of packets subject to an overall delivery deadline).

B. Object Scheduler Design

In light of the above observation we would like to design a multipath packet scheduler that is cognisant of (i) objects within the packet stream and (ii) the impact of fluctuations in path delay on object transmission time.

Index the available paths by $j = 1, 2, \dots, m$ and let $T_k^{(j)}$ and $P^{(j)}$ denote the inter-packet delay and propagation delay, respectively, experienced by packet $k = 1, 2, \dots$ transmitted on path j ,where the inter-packet delay between two packets is defined as the time elapsed between the receipt of the last byte of the first packet and the the last byte of the second packet. Given an object consisting of n packets to be transmitted across the paths let $n^{(j)}$ denote the number of packets sent over path j, with $\sum_{j=1}^{m} n^{(j)} = n$, and $\vec{n} = [n^{(1)}, \dots, n^{(m)}]^T$. The transmission time of the object is then given by

$$D(\vec{n}) := \max_{j \in \{1,\dots,m\}} T^{(j)}(n^{(j)}) + P^{(j)} \quad (1)$$

where $T^{(j)}(n^{(j)}) = \sum_{k=1}^{n^{(j)}} T_k^{(j)}$. The difficulty in designing a scheduler is, of course, that the inter-packet delays $T_k^{(j)}$ are variable and unknown, and so also the aggregate delays $T^{(j)}(n^{(j)})$. While we might attempt to calculate the probability distribution of transmission time $D(\vec{n})$ for every partition $\vec{n}$ this quickly becomes computationally expensive for larger object sizes (so infeasible for real-time packet scheduling) plus in any case we usually lack full details of the distribution of the inter-packet delay on each path. We therefore adopt the following approximate approach and select parameters $w^{(j)} \geq 0$ such that

$$P(T^{(j)}(n^{(j)}) \geq T_U^{(j)}(n^{(j)})) \leq \epsilon^{(j)} := \epsilon/m \qquad (2)$$

where $\mu^{(j)} = E[T_k^{(j)}] \geq 0$, $T_U^{(j)}(n^{(j)}) = n^{(j)}\mu^{(j)} + \sqrt{n^{(j)}}w^{(j)}$ and ϵ is a design parameter which, unless otherwise stated, in the rest of the paper we select to be 0.05 , providing reliability at the 95% percentile. In particular, modelling the inter-packet delays $T_k^{(j)}$ as being i.i.d then using the Chernoff-Hoeffding bound we can solve for $w^{(j)}$ using

$$w^{(j)} = \sqrt{\frac{-\ln(\epsilon^{(j)})(a^{(j)} - b^{(j)})^2}{2}} \qquad (3)$$

where $a^{(j)}$ and $b^{(j)}$ are upper and lower bounds on the inter-packet delay, $a^{(j)} \leq T_k^{(j)} \leq b^{(j)}$. It follows from (2) that

$$P(D(\vec{n}) \geq D_U(\vec{n})) \leq 1 - (1 - \epsilon/m)^m \leq \epsilon \qquad (4)$$

where $D_U(\vec{n}) = \max_{j \in \{1,...,m\}} T_U^{(j)}(n^{(j)}) + P^{(j)})$. We now select a partition $\vec{n}$ that solves the following optimization problem,

$$\min_{\vec{n} \in \mathbb{N}^m} \quad D_U(\vec{n}) \quad s.t. \quad \sum_{j=1}^{m} n^{(j)} = n \qquad (5)$$

1) Receive Buffer Management: Packets are buffered at the receiver until they can be delivered in-order to the application so as to preserve TCP byte-stream semantics. This means that the receiver can become blocked when the receive buffer capacity is reached due to out-of-order delivery or outage on one or more paths. To reduce the occurrence of such blocking we size the receive window as $\sum_{j=0}^{m} \frac{D_U(\vec{n})}{\mu^{(j)}}$ by setting the socket buffer to the appropriate size for each connection. In addition, in the rare cases where blocking does occur we re-transmit the packets which fail to be delivered. Receiver buffer management can be implemented through adapting socket buffer sizes in a user-space network stack that implements SOS.[17]

2) Multiple Objects: When transmitting multiple objects at the same time, the expression for $T_U^{(j)}$ can be modified to account for the in-flight packets

3) Acquiring object sizes: Object information can be acquired for known protocols such as HTTP using decryption at the transport layer. If decryption is not possible, the scheduler can also consider TLS block sizes instead, since decryption cannot occur before the entire block is received. The scheduler can also be placed inside the browser (similarly to QUIC) so that it has access to the object sizes before encryption occurs. A potential approach that would involve modifying the operating system network stack is to add an option to the socket send function to include the object size information to the transport layer.

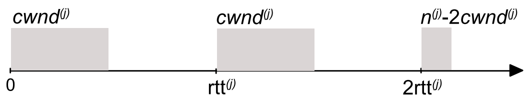


Fig. 5: Illustrating extra delay introduced by cwnd-based congestion control when the cwnd $cwnd^{(j)}$con link j is less than the number $n^{(j)}$ of packets to be sent over link j since only cwnd packets can be sent per RTT.

IV. Interfacing SOS with cwnd-based congestion control

In many cases, packets of an object partition sent over link j are not always committed to the link at once, implying that number of packets that can be in-flight (the cwnd) is smaller to $n^{(j)}$. This introduces extra delay as the $n^{(j)}$ packets will then need to be split across multiple cwnds, and so multiple RTTs. In order to accurately estimate the object partition delay when running on top of cwnd-based congesiton control SOS therefore has to explicitly consider the value of cwnd when making scheduling decisions. In this section we implement a cwnd-aware version of SOS using a user-space network stack. We compare its performance to that of MPTCP/minRTT across a range of network conditions and object sizes and measure the improvement in the 95% percentile and the mean delay of each object. Due to issues of space and implementation details, we leave the evaluation of SOS against other congestion-control aware algorithms to future work.

A. Cwnd-aware SOS

1) Delay Estimation: Cwnd-based congestion control restricts the number of packets that can be sent across a link to cwnd packets per RTT. Hence, when the cwnd $cwnd^{(j)}$ on link j is less that the number $n^{(j)}$ of packets to be sent over link j then it takes multiple RTTs to send the $n^{(j)}$ packets, see Figure 5. Denoting the cwnd on link j by $cwnd^{(j)}$ and the RTT by $rtt^{(j)}$ and assuming that $cwnd^{(j)}$ and $rtt^{(j)}$ stay roughly constant over the time taken to send an object, then the time taken to send $n^{(j)}$ packets over the link is

$$T^{(j)}(n^{(j)}) = N^{(j)}rtt^{(j)} + T^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)}) \qquad (6)$$

where $N^{(j)} = \lfloor n^{(j)}/cwnd^{(j)} \rfloor$ and $T^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$ is the time taken to transmit the $n^{(j)} - N^{(j)}cwnd^{(j)}$ packets which occupy only part of the link j cwnd.

We accommodate fluctuations in the round-trip time and $T^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$ as follows. Firstly, assuming that fluctuations in the round-trip time are i.i.d. then we can replace $N^{(j)}rtt^{(j)}$ by $\sum_{i=1}^{N^{(j)}} RTT_i^{(j)}$ where $RTT_i^{(j)}$ is a random variable equal to the round-trip time during the i'th cwnd taken to transmit the $n^{(j)}$ packets. We can then bound this value using the Chernoff-Hoefdding bound similarly to eqn (2) as

$$\begin{aligned} T_U^{(j)}(n^{(j)}) = N^{(j)}E[rtt^{(j)}] + w^{(j)}\sqrt{N^{(j)}} \\ + T_U^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)}) \end{aligned} \qquad (7)$$

where $w^{(j)}$ is as in eqn (3) with $a^{(j)} \leq RTT^{(j)} \leq b^{(j)}$, and $T_U^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$ is the estimated upper bound on $T^{(j)}(n^{(j)} - N^{(j)}cwnd^{(j)})$.

2) Estimating $E[rtt^{(j)}]$ and $E[T_U^{(j)}(n^{(j)} - N^{(j)})]$: We can estimate $E[rtt^{(j)}]$ as and follows

$$E[rtt^{(j)}] = E[\sum_{i=1}^{cwnd^{(j)}} T^{(j)} + rtt_{min}^{(j)}] \tag{8}$$

$$E[T_U^{(j)}(n^{(j)} - N^{(j)}] = E[\sum_{i=1}^{n^{(j)}-N^{(j)}} T^{(j)}] \tag{9}$$

Where $T^{(j)}$ is the IID random variable representing inter-packet delay and $rtt_{min}^{(j)}$ is minimum round-trip time on link j. The distribution of $T^{(j)}$ and thus $E[T^{(j)}]$ can be collected by observing the arrival time between acknowledgements, assuming that one acknowledgement is generated for each packet. However, this approach suffers from low accuracy due to ack compression, delayed acks, queues on the uplink and cumulative acks.

Instead by recording the transmission time of a packet, the number of packets in-flight when it was transmitted, and the time the acknowledgement for this packet was received we can estimate the distribution of $\sum_{i=1}^{n} T^{(j)} + rtt_{min}^{(j)}$ for various values of n, and we can use that to calculate both $E[rtt^{(j)}]$ and $E[T_U^{(j)}(n^{(j)} - N^{(j)})]$

In cases where we need the $E[\sum_{i=1}^{n} T^{(j)}]$ for previously un-encountered value of n, we can extrapolate as follows

$$E[\sum_{i=1}^{n} T^{(j)}] = \frac{n^{(j)}}{n^{(nearest)}} E[\sum_{i=1}^{n_{nearest}} T^{(j)}] \tag{10}$$

where $n^{nearest}$ is the highest value where $n_{nearest} \leq n$ and the distribution of $T^{(j)}(n)$ is known. If no such $n^{nearest}$ exists, we use the value of $rtt_{min}^{(j)}$.

Additionally, we remove old samples as new ones arrive in order to keep the estimate up to date and to prevent noise from skewing the system for long. In cases where no more samples exist for a link as a result of the link not being utilized for an extended period of time, possibly as a result of delay spikes causing the scheduler to ignore th elink, the value for any $T^{(j)}(n)$ will become $rtt_{min}^{(j)}$ causing the link to start being probed in case the delay profile has changed.

3) Re-Scheduling: Since transmission of the $n^{(j)}$ packets over link j proceeds in $N^{(j)}$ rounds each of cwnd packets, the opportunity exists to re-evaluate our schedule periodically and adjust the value of $n^{(j)}$ (so long as we do not reduce it below the number of packets already transmitted). In our tests we re-evaluate the schedule upon the receipt of an ack from any link, as this presents a new transmission opportunity. It is however possible to perform more sophisticated re-evaluation of the schedule at higher granularity based on the expected arrival time of an ack.

B. Performance Evaluation

1) Experimental Setup: We transmitted objects using two python scripts, one acting as a client sending a request for an object and the other acting as a server sending the object in response to a request. The delay was calculated as time between sending the request for an object and receiving the full object from the server. We allocated the maximum size of receive and send buffers for both the client and the server sockets in order to prevent the MPTCP penalization mechanism from under-utilizing a link. The server transmitted 5 different object sizes, namely 1, 100, 1000 and 3000 packets. Objects of each size being sent repeatedly over a long lived connection for 5 minutes. The reason we used one long-lived connection is to minimize the impact of the capacity estimation on the performance of the schedulers.

We used the Linux tc tool with its extension netem to emulate an LTE and a WiFi link on top of two 1Gbps ethernet cables. The first "LTE" link was configured to have static delay of 40ms and a rate of 40Mbps. The second "WiFi" link was configured to have Gaussian-distributed delay with 10ms mean and standard deviation of 0,10,50,100 and 200ms. The distribution is truncated in order to prevent negative-delay. The delay value is assigned to each packet and will cause all subsequent packets to be delayed by the same value at least in order to preserve in-order delivery.

As a baseline for comparison we used MPTCP/minRTT. The version of MPTCP used is that for the Linux kernel 4.9, with default configuration parameters and Reno congestion control.

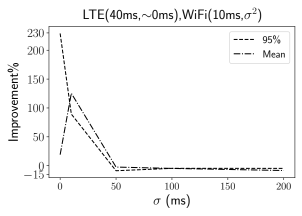
We implemented Cwnd-aware SOS in a user-space network stack

2) Results: Fig. 6(a) present measurements of the performance improvement of SOS over MPTCP/MinRTT when transmitting a single packet. We observe that MPTCP/MinRTT didn't always utilize the faster link when the standard deviation of the RTT is low, occasionally transmitting packets through the slower link which caused both the mean and 95% transmission delay to drop relative to that of SOS. However, for higher levels of RTT variance MPTCP/minRTT and SOS have similar delay performance (MPTCP/MinRTT being slightly better since the way SOS was implemented added a certain amount of fixed delay due to the usage of a virtual interface).

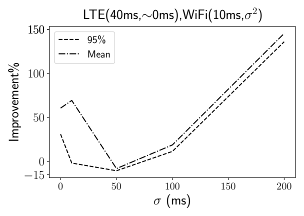
Fig. 6(b) shows the performance when transmitting 100 packets. When the faster link has no RTT variance, MPTCP/minRTT sends packets across the slower link leading to higher than necessary delay. As the level of RTT variance increases MPTCP/minRTT always sends some packets across the faster fluctuating link, although the number sent falls as the RTT variance increases, causing the object delay to increase as level of RTT variance is increased.

In Fig. 6(c) we show measurements for the transmission of 1000 packet objects. It can be seen that SOS initially out-performs MPTCP/minRTT but as the RTT variance increases the difference in performance reduces. This is because SOS reduces its usage of the fluctuating link more quickly than does MPTCP/minRTT, but eventually both schedulers avoid using the fluctuating link.

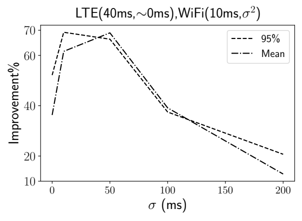
In Fig. 6(d), for 3000 packet objects, it can be seen that initially SOS and MPTCP/minRTT perform similarly as the object spans multiple cwnds which allows time for MPTCP/minRTT to adapt, but as the RTT variance increases, the relative performance of MPTCP/minRTT degrades.



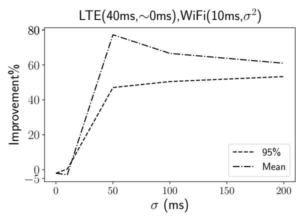
(a) 1 Packet



(b) 100 Packets



(c) 1000 Packets



(d) 3000 Packets

Fig. 6: Comparing performance of SOS against MPTCP/minRTT over two emulated paths, one WiFi and the other LTE. Plot shows improvement by SOS at 95th percentile and mean delay over 5 different object sizes while varying the standard deviation of the RTT variance on the WiFi path.

In summary, across a range of network conditions and object sizes our measurements indicate that SOS consistently outperforms MPTCP/minRTT.

V. Conclusions

In this paper we consider the task of scheduling packet transmissions amongst multiple paths with uncertain, time-varying delay. We make the observation that the requirement is usually to transmit application layer objects (web pages, images, video frames etc) with low latency, and so it is the object delay rather than the per packet delay which is important. This has fundamental implications for multipath scheduler design. We introduce SOS (Stochastic Object-aware Scheduler), the first multipath scheduler that considers application layer object sizes and their relationship to link uncertainty. We also show how SOS can be adapted to take account of cwnd-based congestion control and present performance measurements using MPTCP/minRTT as a baseline. These measurements show that across a range of network conditions and object sizes SOS consistently outperforms MPTCP/minRTT

References

[1] "Next Generation Protocols – Market Drivers and Key Scenarios," *European Telecommunications Standards Institute (ETSI)*, 2016. [Online]. Available: http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp17_Next_Generation_Protocols_v01.pdf

[2] A. Wilk, J. Iyengar, I. Swett, and R. Hamilton, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2." [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00

[3] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. J. Leith, and M. Médard, "Congestion control for coded transport layers," in *2014 IEEE International Conference on Communications (ICC)*, Jun. 2014, pp. 1228–1234.

[4] "Open Fast PAth," 2016. [Online]. Available: https://openfastpath.org/

[5] "Commercial usage of Multipath TCP — MPTCP." [Online]. Available: http://blog.multipath-tcp.org/blog/html/2015/12/25/commercial_usage_of_multipath_tcp.html [last accessed 01-August-2018].

[6] G. Sarwar, R. Boreli, E. Lochin, and A. Mifdaoui, "Performance evaluation of multipath transport protocol in heterogeneous network environments," in *2012 International Symposium on Communications and Information Technologies (ISCIT)*, Oct. 2012, pp. 985–990.

[7] A. Alheid, D. Kaleshi, and A. Doufexi, "Performance Evaluation of MPTCP in Indoor Heterogeneous Networks," in *Proceedings of the 2014 First International Conference on Systems Informatics, Modelling and Simulation*, ser. SIMS '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 213–218. [Online]. Available: http://dx.doi.org/10.1109/SIMS.2014.40

[8] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "A First Analysis of Multipath TCP on Smartphones," Mar. 2016, pp. 57–69.

[9] B. Partov and D. J. Leith, "Experimental Evaluation of Multi-path Schedulers for LTE/Wi-Fi Devices," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, ser. WiNTECH '16. New York, NY, USA: ACM, 2016, pp. 41–48. [Online]. Available: http://doi.acm.org/10.1145/2980159.2980169

[10] E. Rescorla <ekr@networkresonance.com>, "The Transport Layer Security (TLS) Protocol Version 1.2." [Online]. Available: https://tools.ietf.org/html/rfc5246

[11] 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 *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 29–29. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228338

[12] S. Ferlin, O. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, May 2016, pp. 431–439.

[13] G. Sarwar, R. Boreli, E. Lochin, A. Mifdaoui, and G. Smith, "Mitigating Receiver's Buffer Blocking by Delay Aware Packet Scheduling in Multipath Data Transfer," in *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, Mar. 2013, pp. 1119–1124.

[14] F. Yang, Q. Wang, and P. D. Amer, "Out-of-Order Transmission for In-Order Arrival Scheduling for Multipath TCP," in *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, May 2014, pp. 749–752.

[15] K. Chebrolu and R. Rao, "Communication using multiple wireless interfaces," in *2002 IEEE Wireless Communications and Networking Conference Record. WCNC 2002 (Cat. No.02TH8609)*, vol. 1, Mar. 2002, pp. 327–331 vol.1.

[16] A. Garcia-Saavedra, M. Karzand, and D. J. Leith, "Low Delay Random Linear Coding and Scheduling Over Multiple Interfaces," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3100–3114, Nov. 2017.

[17] K. Fahmi and S. Kucera, "Multiple path transmission of data," Patent PCT/EP2017/073 192.