DAPS: Intelligent Delay-Aware Packet Scheduling For Multipath Transport

Nicolas Kuhn,[†,‡] Emmanuel Lochin,[†]
Ahlem Mifdaoui[†]
[†]Université de Toulouse; ISAE, LAAS-CNRS, France
[†]{first.last}@isae.fr

Golam Sarwar[‡,⋆], Olivier Mehani[‡], Roksana Boreli[‡,⋆]
[‡] NICTA Australia
[⋆] University of New South Wales, Australia
[‡]{first.last}@nicta.com.au

***Abstract*—The increasing heterogeneity and asymmetry in wireless network environments makes QoS guarantees in terms of delays and throughput a challenging task. In this paper, we study a novel scheduling algorithm for multipath transport called Delay Aware Packet Scheduling (DAPS) which aims to reduce the receiver's buffer blocking time considered as a main parameter to enhance the QoS in wireless environments. We develop an analytical model of maximum receiver's buffer blocking time and extend the DAPS algorithm considering implementation issues. Performance evaluations based on ns-2 simulations highlight the enhanced QoS that DAPS can provide. With reference to the classical multipath transport protocol CMT-SCTP, we observe a significant reductions of the receiver's buffer occupancy, down by 77%, and the application delay, down by 63%.**

***Keywords*-Multipath Transport Protocol; CMT-SCTP; Delay-Aware Packet Scheduling**

I. Introduction and Motivation

With recent advances in mobile computing, an increasingly large population of users are relying on their smart-phones or tablets as primary means of accessing online services. Such devices include heterogeneous wireless network access (commonly 3/4G and Wi-Fi), and motivate the research and industry effort in multipath transport protocols, that can better utilise the available (multiple) network capacity. This includes the ongoing work on multipath and multi-homing capable versions of TCP considered at IETF [1], with MPTCP [2] also having industry support as demonstrated by the recently released implementation included in Apple iOS7.[1] An enhancement of the Stream Control Transport Protocol (SCTP) [3] for Concurrent Multipath Transfer (CMT-SCTP) [4] has also received research attention. Recent adoption of SCTP for WebRTC (a real-time multimedia communication protocol) let us foresee the use of SCTP for Web applications.

In the common multipath scenario where the links are asymmetric (*i.e.*, different delays and capacities), receiver's buffer blocking has been identified as a problem for both MPTCP [5] and CMT-SCTP [6]: out-of-order packets may occupy the entire receiver's buffer eventually stalling the whole transmission flow. Indeed, before transmitting newer data packets, both MPTCP and CMT-SCTP's congestion control mechanism will check if the receiver's buffer has enough storage for the transmitted data, while packets can only be taken out in order. The impact of this issue was quantified [7], [8] and there have been a number of proposals to overcome this problem.

Some proposed mitigation techniques rely on buffer management, such as increasing its size [5], or splitting it [7]. These solutions consider the trade-off between large buffer size and fixed capacity with limited buffer-size and are only applicable to scenarios where the communicating devices have enough memory; otherwise using smaller buffers results in a limited throughput. A second set of proposed solutions is based on specific retransmission policies [6], [9], [10], which may result in wasted resources on some paths.

In this paper, we develop a **model for maximum blocking time due to multipath delay imbalance** (the maximum amount of time a packet will wait in the receiver's buffer for in-order delivery to the upper layers) and **validate it with ns-2 simulations**. We investigate an alternative approach that uses scheduling of packets on the sender side, to match the paths asymmetry and avoid buffer blocking. We propose improvements to **Delay Aware Packet Scheduling (DAPS)** [11], which algorithm and integration deserve more explorations. We **implement and evaluate DAPS in ns-2**, and further adapt the scheduling of CMT-SCTP that selects packet sequences to be transmitted over each paths depending on their RTTs, in order to promote in order reception. We show that our proposal can **reduce the occupancy of the receiver's buffer for up to 77%, drive application delays down by around 63%** and **increase the cumulative throughput by 42%** for the selected simulation parameters. These results illustrate that DAPS is a very promising solution to overcome the receiver's buffer blocking issue.

The organisation of the rest of the paper is as follows. In Section II we model and validate the maximum receiver blocking time. Section III presents DAPS and Section IV evaluates the benefits that our scheduling can provide. We discuss implementation issues in Section V. Finally in Section VI we draw conclusion and propose future work.

II. Receiver Blocking on Asymmetric Links

Throughout this paper, we consider a scenario where two paths with different characteristics are available between a source and a destination. This is depicted in Figure 1. Most

[1]http://perso.uclouvain.be/olivier.bonaventure/blog/html/2013/09/18/mptcp.html

notably, we consider an imbalance in the round-trip times: the slow path has a higher RTT than the fast path, $r_s > r_f$. For ease of understanding, and without loss of generality, we also consider that the slow path has a lower capacity, $c_s < r_s$.

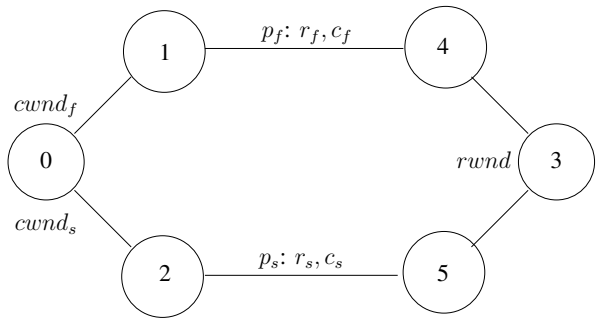

Figure 1: Two paths p_i (where i is s for "slow", or f for "fast") with different characteristics (RTT r_i [ms] , capacity c_i [Mbps]) are available between source and destination. The sender maintains one congestion window $cwnd_i$ per path; the receiver has a single buffer, and advertises only one window, $rwnd$.

In this section, we evaluate an upper bound of the blocking time, defined by the maximum amount of time a packet shall wait in the receiver's buffer for in-order delivery to the upper layers. We first summarise the standard round robin scheduling and congestion window updates used by both MPTCP [2] and CMT-SCTP [4]. In order to obtain the lowest maximum blocking time, we do not consider losses. This allows us to focus on the impact of the asymmetry of the links. We validate this model with ns-2 simulations.

A. Round-robin Scheduling and Congestion Window Updates

In both MPTCP and CMT-SCTP, a separate congestion window $cwnd_i$ is kept for each individual path p_i. The scheduler loops over the paths in a "blind" round robin fashion. For every path in sequence, it sends as much data as the congestion window for that path allows (taking data already in flight into account but not yet acknowledged, $unack_i$) that the receiver is able to store,[2] that is,

$$\min(cwnd_i - unack_i, rwnd - \sum_i unack_i). \qquad (1)$$

The scheduler then repeats this operation for the next path, and loops around to the first path when data has been sent on the last one.

Once there are as many bytes in flight as the receiver's buffer can store, *i.e.*, $rwnd - \sum_i unack_i = 0$, no more data can be sent, and the transmission is blocked. This limitation allows to prevent overflow of the receiver's buffer.

However it is only possible to deliver in-order packets to the application. On paths with different delays, packets sent in-order will not reach the receiver in sequence. They will therefore not be delivered to the upper layers, but will still be deducted from $rwnd$, resulting in lower throughput and higher application delay. We model and quantify this in the next section.

[2]For SCTP, the advertised receiver window is `a_rwnd`; this is what we call $rwnd$ here, in TCP fashion.

B. Asymmetric Links and Receiver's Buffer Blocking

To evaluate the impact of asymmetric links on multipath data transfer, we derive $T_{maxblock}$, the maximum time during which packets may be stored in the receiver's buffer before being delivered, in-order, to the application. We illustrate an extreme case in Figure 2: a first packet is sent on a slow path, and enough subsequent packets are sent on the fast path to fill the receiver's buffer. While the latter are received in a timely manner, the transfer is blocked, waiting for the first packet to be received, and eventually acknowledged.

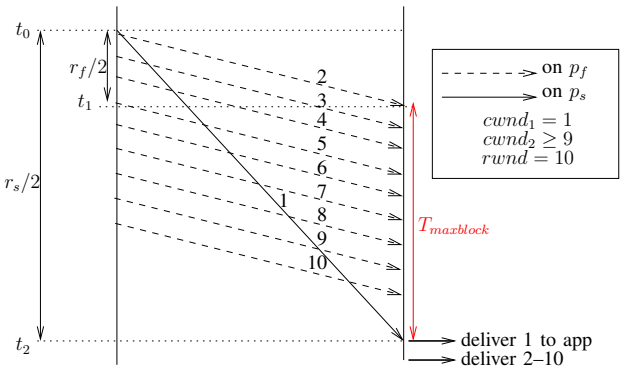

Figure 2: Example of a blocking state. Packet 1, delivered on the slow path delays the entire transfer, even though packets 2–10 have already arrived.

We denote by TSN_j, the sequence number of each packet, and by L, the size of a data packet, in bytes. We assume that the time to place one packet on the physical medium is $8L/c_i$.

Due to the $rwnd$-based flow control, the queuing delays that could be introduced are limited by the fact that the maximum amount of bytes in flight must not be more than $rwnd$. As a result, the model we propose considers that there are no packets in flight, following the scheme that introduces the larger maximum blocking time.

Figure 2 shows a sequence that triggers the maximum blocking time. This is observed when a packet with TSN_j is transmitted on the slow path (p_s which RTT verifies $r_s = \max_i r_i$) right after the transmission of packets TSN_{j+1}–TSN_{j+rwnd} on the fast path (p_f, $r_s = \min_i r_i$). The timing of events is as follows.

- from t_0 to t_2, transmission of TSN_1 on path p_s, transmission of TSN_2–TSN_{10};
- at $t_1 = r_f/2$, reception of TSN_2;
- at $t2 = r_s/2$, reception of TSN_1, delivery of the whole series of packets to the application.

The maximum forward transmission delay at the application layer level can be calculated as $T_{maxapp} = t_2 - t_0 = r_f +$

$8L/c_f + T_{maxblock}$ and the maximum blocking time as

$$T_{maxblock} = t_2 - t_1 = \frac{r_s}{2} + \frac{8L}{c_s} - \frac{r_f}{2} - \frac{8L}{c_f}. \quad (2)$$

C. *ns-2*-based Validation of the Model

We now validate our maximum blocking time model (2) by comparing the actual blocking time observed in *ns*-2 traces for the same parameters. The static simulations parameters are: $rwnd = 65$ kB, $L = 1500$ B, $r_f = 20$ ms and the size of the different queues is set by the rule-of-thumb $q_i = r_i c_i / 8L$; we vary the parameters of the slow path, r_s and c_s, as well as the capacity of the fast path, c_f. Table I presents the simulation cases and the corresponding results obtained with our model.

Table I: Simulation cases and corresponding model results.

Label (c_f;c_s/r_s)	c_f [Mbps]	c_s [Mbps]	r_s [ms]	$T_{maxblock}$ [ms]
(10;10/100)	10	10	100	40
(50;10/100)	50	10	100	41
(10;50/100)	10	50	100	39
(100;100/100)	10	10	100	40
(50;100/100)	50	15	100	39
(150;50/100)	15	50	100	40
(10;10/200)	10	10	200	90
(50;10/200)	50	10	200	91
(10;50/200)	10	50	200	89
(100;100/200)	10	10	200	90
(50;150/200)	50	15	200	89
(150;50/200)	15	50	200	90

Figure 3 shows a snapshot of the evolution of blocking time measured in *ns*-2 simulations for some simulation cases.

- For each case, we plot an horizontal line which corresponds to the result obtained with the model;
- For each IP packet, we measured the time they had to be stored in the receiver's buffer before in-order-delivery to the application.

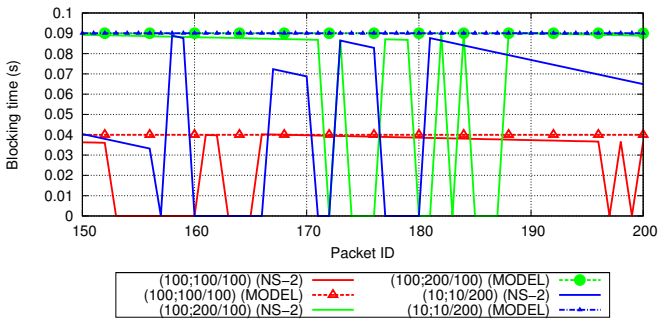

Figure 3: Blocking time as forecast by our model (dashed lines) and measured in *ns*-2 (full lines) for a subset of the cases of Table I

As Figure 3 illustrates, our proposed model of the maximum blocking time and accurately predicts the additional delay experienced by an application using a round-robin multipath transport over asymmetric paths.

Our results justify the need for a solution which reduces this blocking time. In Section III, we introduce advanced scheduling and demonstrate that we can indeed drastically reduce the blocking time, as well as the receiver's buffer occupancy. In the following section, we first review related techniques proposed to mitigate receiver buffer blocking in multipath transport.

D. Solutions to Address Buffer Blocking

The first solution is to provide a buffer large enough so that it may handle a given scenario. A simplistic calculation indicates a buffer with size $Rbuf$ equal to the combined bandwidth×delay product using the largest RTT of the paths could handle this problem, that is,

$$Rbuf_{min} = \sum_{i \in \{p_1,\ldots,p_n\}} c_i \times \max_{i \in \{p_1,\ldots,p_n\}} r_i. \quad (3)$$

This solution is however neither optimal nor scalable, as $Rbuf_{min}$ can quickly grow beyondmanageability. For example if we consider a scenario of two paths with 10 and 1 Mbps capacity, and 20 and 200 ms RTT, respectively, the required minimum buffer size would already be 275 kB to prevent blocking.

Other solutions based on buffer management techniques include a scheme where the receiver's buffer size is similarly adapted depending on the highest RTT of the available paths [5] and the proposal to dedicate parts of the receive buffer to specific paths [7].

Once the receiver's buffer is filled with out-of-order packets (*i.e.*, $rwnd = 0$), missing packets are retransmitted, even though they might actually be in flight over the longer delay paths. A number of retransmission-based mitigation techniques were therefore proposed to choose the best path for retransmission [6], [9], [10]. While these solutions allows a faster unblocking of the receiver buffer, they do so by unnecessary use of additional network resources, retransmitting packets which were not lost.

Rather than attempting to mitigate the consequences of a buffer blocking, we propose to correct the cause of the problem. We do so by introducing packet/path scheduling at the sender. Packets are sent (not necessarily in order) on a specific path, based on their TSN and path parameters, so that they are received in order by the receiver. We present, implement and evaluate this solution in the following sections.

III. Delay-Aware Packet Scheduling (DAPS)

To mitigate the problems associated with the blind round-robin packet transmission, we propose to introduce finer scheduling mechanisms, which specifically transmit a given packet on a given path. In this section, we first show how this scheduling is enabled in the packet-transmission logic—for clarity, we take CMT-SCTP as a specific example. We then present our Delay-Aware Packet Scheduling algorithm, and discuss operating conditions.

Algorithm 1 Blind round-robin

for each $p_i \in P$ **do** {Loop over paths}
  while $unack_i < cwnd_i$ **do**
    $TSN_j \leftarrow$ **getNextUnsentChunk**
    if $L \leq rwnd - \sum_i unack_i$ **then**
      transmit TSN_j on p_i
    end if
  end while
end for

Algorithm 2 Packet/path scheduling

$S \leftarrow$ **generateSchedule**
for each $s_j = (TSN_j, p_j) \in S$ **do** {Loop over schedule}
  while $unack_j < cwnd_j$ **do**

    if $L \leq rwnd - \sum_i unack_i$ **then**
      transmit TSN_j on p_j {Note: using p_j from s_j}
    end if
  end while
end for

Figure 4: Integrating packet/path scheduling involves considering packets in a specific order, rather than looping over paths

A. Introducing Packet/Path Scheduling

The main idea behind packet/path scheduling is not to transmit packets in monotonically increasing sequences on any available path, but to carefully choose which packet should be sent when and over which path, based on some characteristic of the associated paths.

In the following, we take $P = \{p_1, \ldots, p_n\}$ a set of paths. As before, each p_i has its associated RTT r_i, and the sender maintains $cwnd_i$. We also introduce a method, **generateSchedule**, which generates a schedule $S = \{s_1, \ldots, s_m\}$. Each element $s_j = (TSN_j, p_j)$ of S represents that packet TSN_j is to be transmitted on path p_j.

Figure 4 shows the difference between both approaches. Algorithm 1 shows CMT-SCTP's blind round-robin method, looping over paths and sending packets in sequence. Algorithm 2 shows how to introduce packet/path scheduling into this process.

B. Generating a Delay-Aware Schedule

This section details how **generateSchedule** computes S to implement delay-aware packet scheduling. The main idea behind DAPS is to aim for *in-order arrival at the receiver* to prevent its buffer from blocking. The following assumes that there are more than $cwnd_f$ packets to transmit, otherwise using solely the fast path would be sufficient.

Algorithm 3 shows how a delay-aware schedule is generated, based on the round-trip time of each path, r_i. In the specific case of two paths, as depicted earlier in Figure 1, the sequence number of the packets transmitted on the slow path p_s is determined depending on the ratio between the two RTTS, $\eta = \lfloor r_s/r_f \rfloor$, and $cwnd_f$, the congestion window on the fast path. The number of packets in flight on each path, $unack_i$, is also important and we include it in Algorithm 3 but we assume for simplicity it is 0 in this explanation.

We want the sender to transmit $max_f = \min(\eta, cwnd_f)$ packets on fast path p_f during the longest forward delay $r_s/2$, while the next $cwnd_s$ packets should be transmitted during the same amount of time on the slow path p_s, that is, transmit $TSN_j, \ldots, TSN_{j+max_f}$ on p_f and $TSN_{j+max_f+1}, \ldots, TSN_{j+max_f+cwnd_s}$ on p_s. Recalling the parameters of the scenario depicted in Figure 2, the first $\min(10, \geq 10) = 10$ packets (TSN_1–TSN_{10}) should be transmitted on p_f, and the following $cwnd_s = 1$ (TSN_{11}) on p_s.

Algorithm 3 Delay-aware **generateSchedule** for two paths

Require: $n_{pkts} > cwnd_f - unack_f$ {Too many packets for the fast path only}
Ensure: S is a packet/path schedule so packets are received in order
  $S \leftarrow \emptyset$
  $\eta \leftarrow \lfloor r_s/r_f \rfloor$ {Note: $r_s \geq r_f$}
  $max_f \leftarrow \min(\eta, cwnd_f - unack_f)$ {Maximum number of packets to send on the fast path}
  for $j = 1, \ldots, max_f$ **do** {Schedule for the fast path}
    $s_j \leftarrow$ (**getNextUnsentChunk**, p_f) {j^{th} chunk}
    Append s_j to S
  end for
  for $j = 1, \ldots, cwnd_s - unack_s$ **do** {Schedule for the slow path}
    $s_j \leftarrow$ (**getNextUnsentChunk**, p_s) {$(max_f + j)^{\text{th}}$ chunk}
    Append s_j to S
  end for
  return S

DAPS does not explicitly consider how much storage the receiver has available while computing S. Depending on $rwnd$, the transmitter may therefore not be able to transmit all the elements of S when executing Algorithm 2. The next execution of Algorithm 3 will generate a new schedule including all the packets that could not have been transmitted on the previous execution of Algorithm 2, and giving priority to the packets with the lowest TSN by sending them on the fast path. Most notably, this includes packets that need be retransmitted, regardless of which path they were initially sent.

C. Choosing to Use DAPS over Blind Round-robin

As we show in the next section, DAPS provides a better ordering of packet arrivals at the receiver. However it also requires more computation, as it needs to generate full schedules rather than simply send packets in sequence. While we never found DAPS to be less efficient than blind round-robin, we

identified cases where the latter was sufficient, or the former too costly to run frequently.

We derived the following criteria to decide when using DAPS is really beneficial. First of all, DAPS is designed for asymmetric paths, and performs similarly to blind round-robin when paths do not have very different delays. Therefore, DAPS provides a noticeable advantage only when

$$r_s >> r_f. \tag{4}$$

In addition, as a given delay-aware schedule is based on the values of both round-trip times and congestion windows of all paths, it is only valid if these values are stable during the full duration of the schedule. It is advisable to only use such a schedule when the following conditions on new samples of the RTTs and congestion windows are met for all paths.

$$R_i(t+1) \in [0.9 \times R_i(t); 1.1 \times R_i(t)], \text{ and} \tag{5}$$

$$cwnd_i(t+1) \in [cwnd_i(t); cwnd_i(t) \times (1+ \frac{cwnd_i(t+1) - cwnd_i(t)}{cwnd_i(t)})]. \tag{6}$$

This most notably excludes the slow-start phase, during which neither parameter is very stable.

IV. Performance of DAPS

We implemented DAPS within the CMT-SCTP ns-2 module. In this section, we assess the benefits of our delay-aware packet scheduling. We evaluate the performance of DAPS by simulating the network presented in Figure 1 in ns-2, and consider various cases for the slow path, presented in Table II, with fixed parameters for the fast path ($c_f = 1$ Mbps and $r_f = 20$ ms). We also vary the receive buffer to be smaller or larger than the rule-of-thumb of (3).

Table II: Simulation cases for evaluation of DAPS.

Label (c_s/r_s)	c_s [Mbps]	r_s [ms]	$Rbuf$ [kB]		$Rbuf_{min}$ [kB] (3)
(2/100)	2	100	35	<	37.5
(2/100)	2	100	500	>	37.5
(1/200)	1	200	45	<	50
(1/200)	1	200	500	>	50
(2/200)	2	200	45	<	75
(2/200)	2	200	500	>	75

Several metrics are of interest. Figure 5 shows the average throughput of the slow path in the considered cases, using either CMT-SCTP's blind round-robin, or DAPS. In Figure 6, we show a snapshot of the cummulative throughput on both paths, for the specific case $(c_s; r_s)$=(2 Mbps;100 ms). In Figure 7, we plot the average application delay, *i.e.*, the time between the transmission of a packet on the path and the in-order delivery to the destination application. Finally, Figures 8 and 9 are snapshots of the number of packets in the receiver's buffer, for values smaller (or larger) than $Rbuf_{min}$ from (3).

First, for $c_s = 2$ Mbps and $Rbuf < Rbuf_{min}$, we observe that the capacity of path p_s cannot be fully exploited

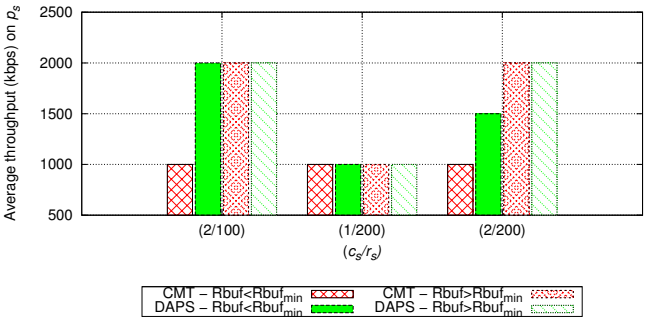

Figure 5: Average usage of the slow path. Unlike blind round-robin, for cases when $Rbuf < Rbuf_{min}$, DAPS is able to utilise the full capacity of the slow path.

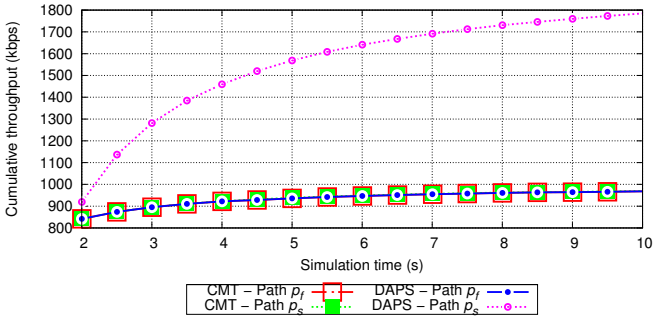

Figure 6: Cumulative throughput; case (1/20;2/100), $Rbuf =$ 35 kB$< Rbuf_{min}$.

with CMT-SCTP, while the scheduling proposed in DAPS enables improved use of this capacity (Figure 5). For the case presented in Figure 6, we measure an increase of 42% of the cumulative throughput of both paths (*i.e.*, $\frac{968+1784}{968+969} = 1.42$). This can be explained by the reduced number of packets in the receiver's buffer. Indeed, with CMT-SCTP, there are, on average, 10 packets waiting to be delivered in-order to the application, while DAPS has 2, which therefore allows the sender to transmit more packets sooner.

Second, Figures 8 and 9 illustrate DAPS's reduction of buffer occupancy. For example, when $c_s = 2$ Mbps and

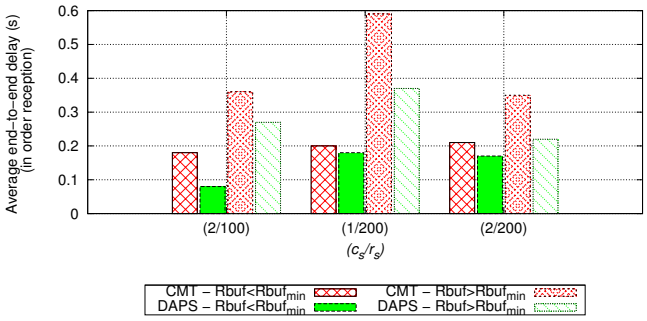

Figure 7: Average application level transmission delay. DAPS allows for packets to be delivered in-order earlier to the application.

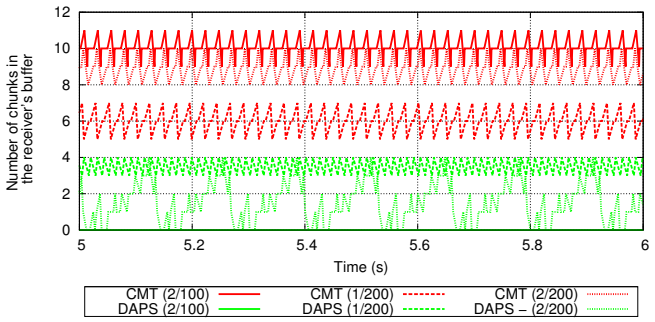

Figure 8: Occupancy of a small receiver's buffer

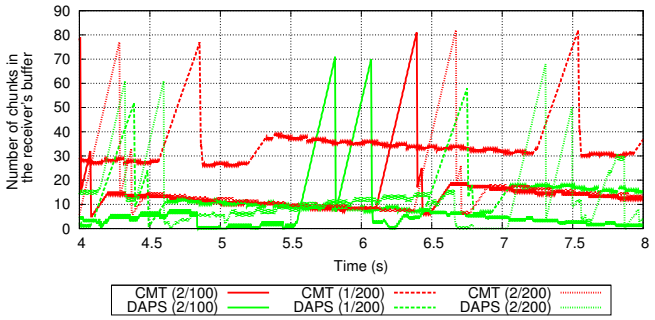

Figure 9: Occupancy of a large receiver's buffer

$r_s = 200\,\text{ms}$, the buffer occupancy is reduced by 77% (from 9 to 2). This explains the results in Figure 7 where the delivery-to-application delay is consistently shorter with DAPS: our scheduling enables better ordering of packet arrivals and thereby optimises the end-to-end transmission time. In particular, in the case where $Rbuf > Rbuf_{min}$ and ($c_s = 1\,\text{Mbps}/r_s = 200\,\text{ms}$), the end-to-end transmission delay decreases by 63%.

Finally, it is worth noting that increasing the receive buffer, as per the rule-of-thumb (3) has an adverse property of increasing the delivery-to-application delay regardless of the packet scheduling. However, even then, DAPS experiences a lower delay than basic CMT-SCTP.

The results presented in this section strongly support the introduction of advanced packet scheduling techniques for multipath transport protocols. We show that our delay-aware packet/path scheduling allows to reduce buffer occupancy, increase packet delivery to the application, and better use the capacity on all available paths.

V. Implementation considerations

The previous section shows example improvements achievable with DAPS on asymmetric links. However we still need to address a number of practical issues that relate to integrating DAPS within CMT-SCTP (or other multipath transport such as MPTCP).

First, we have assumed that there is a sufficiently large difference in paths, to ensure that DAPS will provide a gain compared to standard round-robin scheduling. Quantifying this difference, that could be used as a trigger for introducing DAPS, is a subject for future study.

Then, DAPS scheduling introduces a processing overhead, as it generates packet schedules rather than just taking the following in-sequence packets for transmission. This needs to be characterised and evaluated, in regards to the overhead compared to the round-robin scheme.

Finally, DAPS relies on estimations of end-to-end round-trip times on each of the paths, but RTT estimations by the transport protocol may not be very accurate. Although the improvement resulting from the use of DAPS will increase with the increased path imbalance and, with increased imbalance, the difference (even for incorrect estimations) may be less relevant, the impact of RTT estimation error needs to be evaluated and is also planed for future study.

VI. Conclusion and Future Work

We argued for enhanced packet scheduling mechanisms, and presented a Delay Aware Packet Scheduling (DAPS) to improve the performance of multipath transport protocols on asymmetric links. We evaluated the performance of this mechanism when used in CMT-SCTP for example parameters and showed that our contribution enables a lower occupancy of the receiver's buffer, resulting in an improved utilisation of aggregate link capacity and a reduced end-to-end transmission delay.

In future work, we plan to implement DAPS in FreeBSD's CMT-SCTP stack and in Linux implementation of MPTCP to evaluate the performance gain in realistic network conditions and address the related practical issues.

References

[1] Multipath TCP (mptcp). IETF. [Online]. Available: http://datatracker.ietf.org/wg/mptcp/charter/

[2] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," RFC 6824, 2013.

[3] R. R. Stewart, "Stream control transmission protocol," RFC 4960, 2007.

[4] J. Iyengar, P. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths," *IEEE/ACM Transactions on Networking*, 2006.

[5] C. Raiciu, C. Paasch, B. S., A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? Designing and implementing a deployable multipath TCP," in *USENIX NSDI*, 2012.

[6] J. Iyengar, P. Amer, and R. Stewart, "Retransmission policies for concurrent multipath transfer using SCTP multihoming," in *12th IEEE ICON. Proceedings.*, 2004.

[7] H. Adhari, T. Dreibholz, M. Becke, E. Rathgeb, and M. Tüxen, "Evaluation of Concurrent Multipath Transfer over Dissimilar Paths," in *IEEE WAINA*, 2011.

[8] G. Sarwar, R. Boreli, E. Lochin, and A. Mifdaoui, "Performance evaluation of multipath transport protocol in heterogeneous network environments," in *ISCIT*, 2012.

[9] J. Liu, H. Zou, J. Dou, and Y. Gao, "Rethinking retransmission policy in concurrent multipath transfer," in *IIHMSP*, 2008.

[10] Y. Qiao, E. Fallon, J. Murphy, L. Murphy, and A. Hanley, "Path selection of SCTP fast retransmission in multi-homed wireless environments," in *Wireless and Mobile Networking*, ser. IFIP. Springer US, 2008.

[11] 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 *IEEE WAINA*, 2013.