Differentiated Pacing on Multiple Paths to Improve One-Way Delay Estimations

Matthieu Coudron, Stefano Secci, Guy Pujolle
Sorbonne Universites, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France.
Email: firstname.lastname@upmc.fr

***Abstract*—Several works in the literature show that accurate estimations of the actual One-Way Delays (OWD) could improve the performance of various network protocol, such as Transport Control Protocol (TCP) throughput. With the emergence of multipath transport protocols like Multipath TCP or the Stream Control Transport Protocol (SCTP), the potential impact can be even higher. Indeed, as multipath transport protocols send data concurrently on heterogeneous paths, the knowledge of corresponding OWDs can greatly help mitigating packet arrival disorder. Theoretically, clock synchronization protocols between endpoints could ensure OWD knowledge, but their efficiency at the Internet scale is debatable. In practice, TCP uses the Round Trip Time (RTT) to take into account congestion or to compute retransmission timeouts, and the OWD is assumed to be half the RTT. However, numerous studies show that a majority of Internet connections experience latency asymmetry. In this paper, we propose a technique based on differential pacing over multiple paths to obtain an estimation of the difference in OWDs between the different paths, motivated by its strong utility for multipath transport protocols such as MPTCP and SCTP. Simulations show which are the interesting scenarios of application.**

I. Introduction

Accurate estimations of One-Way Delays (OWDs) in networks could improve network utilization and in particular user's quality of experience (QoE): applications such as voice-over-IP or video streaming depend more on the forward time delay than the reverse one, for example. As another example, the TCP vegas [1] family of congestion control algorithms rely on Round-Trip Time (RTT) inflation to detect congestion but is unable to distinguish between forward or reverse path inflation. Including congestion control also on acknowledgments (ACK) and not only on data packets would likely increase TCP performance as noted by [2].

Among the most promising evolution at the transport layer we can cite the Stream Control Transport Protocol (SCTP [3]) and Multipath TCP (MPTCP [4]), the latter experiencing a broad deployment recently. The two protocols differ in terms of incremental deployability features, but both can significantly increase resiliency, security and throughput. In particular, throughput increase, though an important driver for multipath transport protocols, is harder to achieve than it seems. Indeed, sending packets along heterogeneous paths (e.g., paths that differ in latencies, loss rates, etc) can result in head of line blocking, which decreases throughput, to the extent that sometimes MPTCP, e.g., would achieve even less than a legacy TCP connection [5]. The OWD knowledge - or rather the ΔOWD between subflows - is then even more crucial in multipath communications than in single path communications, knowing the forward ΔOWD allows to reduce in-arrival packet disorder, thus reducing buffer requirements or increasing throughput. In [6], a multipath UDP scheduler leads to a 30% increase in throughput when taking into account ΔOWD. A recent MPTCP scheduler [7] achieves a similar gain via modifying both the sender and the receiver.

As for the reverse ΔOWD, knowing which is the fastest reverse path lets the multipath transport protocol the possibility to acknowledge packets on the fastest path. Coupled with non-renegable selective acknowledgments, this allows to free sooner the send buffer in order to send fresh data faster [8].

While the benefits of knowing ΔOWDs look interesting, propositions to retrieve the information rely either on clock synchronization or some form of cooperation, none of them being standardized. With these constraints in mind, we elaborate an alternative to estimate the difference in OWDs between subflows of two end hosts.

The rest of the paper is organized as follows. Section II describes related work on OWD estimation. Section III precisely describes our solution. Finally, in section IV we expose and discuss simulation results.

II. Related work

In this section we present OWD and ΔOWD online estimation techniques for the single path case first, followed by techniques specific to multipath protocols. Offline OWD estimation techniques (i.e. where transit times are corrected a posteriori, as in [9]) also exist but they require cooperation from the network, or clock synchronization between end hosts. Thus we only consider in the following online techniques, able to dynamically adjust scheduling parameters.

From a practical point of view, the presented techniques may better apply to Wide Area Network (WAN) communications than to Local Area Network (LAN) communications (or intra-datacenter communications) as LAN smaller RTTs require more precision.

A. Clock synchronization in packet switched networks

The dissemination of clock synchronization information in packet-switching networks can be achieved through different protocols, with different precision levels. The most adopted solutions being the Global Positioning System (GPS) [10], the Precision Time Protocol (PTP) [11] and the Network Time Protocol (NTP) [12].

A GPS terminal can infer the clock by correlating the positioning signals from a constellation of satellites, each satellite embarking several atomic clocks, reaching a precision skew of a few nanoseconds only. GPS receivers are precise but expansive, thus not all computers can be equipped with one: time has to be distributed.

PTP [11] is able to distribute this time in packet-switched networks through continuous offset correction between a grandmaster clock and hierarchy of master-slave clocks. With sub-microsecond accuracy, this approach allows reaching very high precision, suitable for 4G base station synchronization.

Similarly NTP relies on a hierarchy of clocks, the higher the stratum, the lower the precision, which is in the order of a milliseconds at the endhost. NTP assumes that OWDs are half the RTT, which is an approximation considered as too coarse by many studies [13] [14]. As noted in [15], when applied to Internet connections this approximation may lead to noticeable errors as the internet does not guarantee symmetric routing. Even when routing is symmetric, there can still be noticeable differences between OWDs: because directions may have different characteristics (loss rates, capacities) due to policies (e.g., different resource reservation levels in cellular networks) or physical constraints (e.g., asymmetric bandwidth in ADSL networks).

B. TCP variations

Some variations of TCP such as TCP Vegas [1] try to infer the congestion level of the network from the evolution of the RTT. Two ways exist to retrieve the RTT for a TCP sender:

1) Compute the time needed to send a full window and receive the matching acknowledgments.
2) Use the TCP Timestamp option [16].

The TCP timestamp option can be used for RTT measurement. Its usage must be negotiated between the end-hosts during the connection establishment. Once negotiated, each host records the time at which it created a packet into this very packet. Upon reception of that packet, the receiver puts its own timestamp along the received one and returns the packet. Upon reception of the acknowledgment, the TCP sender retrieves the timestamp it previously sent and subtracts its value to the current node time.

There is little one host can deduce from the remote host timestamps since the standard only guarantees the remote TCP clock to be 'monotone-increasing' [16]. Alternatively, propositions exist to negotiate the clock skew during the connection establishment, as presented in [17]. This would allow each host to interpret the timestamp of the remote host. Knowledge of the receiver skew allows the sender to correct the remote timestamp so that the duration matches its local skew. It also gives an indication on the possible precision. The major drawback of TCP variations relying on RTT estimations is that TCP takes a coarse decision based on the maximum level of congestion between the forward and backward paths.

Choi et al. [18] modify the roles of the fields in the timestamp option: timestamps are replaced with the actual RTT values measured by the different nodes. Providing both hosts relayed RTTs to their peer since the beginning of the communication, the OWDs can be computed analytically with a precision depending on the value guessed for the first OWD. Gurewitez and al. [19] take another approach that requires cooperation from the network for the forwarding aspect, but no clock synchronization. To remove the uncertainty about an untrusted remote clock, a node sends probes through different paths in the network (there must be some mechanism to enforce these paths, e.g., source routing, traffic engineering) that must come back to the initial sender. The different values obtained through the use of the probes add constraints over the values of the different OWDs. When enough constraints have been harvested to solve the system of equations, the algorithm tries to determine the OWDs that yield the least square error.

C. Multipath control techniques

For a multipath transport protocol such as MPTCP, multiple connection 'subflows' can be opened between the end-hosts, regardless of the number of interfaces. Having an estimation of the difference between subflow OWDs (noted as $\Delta^F OWD$ for the forward delta OWDs, and $\Delta^B OWD$ for the backward delta OWDs in the rest of the article) would already allow some improvements. Moreover, in case of low traffic, this knowledge should be needed by delay-sensitive applications to send packets on the fastest path (provided loss ratios are similar).

As for the $\Delta^B OWD$, the faster an ACK reaches the sender, the faster the sender can free space in its send buffer and send new data packets. The knowledge of the fastest reverse path would thus let a multipath transport protocol such as MPTCP the possibility to acknowledge packets on the fastest path, as explained in [20].

Relying on [17], authors in [21] assume clock skew synchronization based on the TCP timestamp option. The sender, instead of discarding receiver timestamps, saves them; in this way, it can deduce the difference in arrival time at the receiver between packets that followed different paths. The difference in backward and forward delay can easily be deduced afterwards by the sender.

In [22], Zhou et al. propose to deduce the OWDs of two subflows from the size of the receiver input buffer. The upside is that it does not require any receiver modification or network cooperation but it assumes constant transmission rates and no loss. It also requires several measurements to get an accurate value.

III. Proposed OWD estimator

The primary objective of our estimator is to provide a practical estimation of forward $\Delta OWDs$ to a multipath transport protocol. The desirable OWD estimator shall thus:

- not require clock synchronization nor network cooperation; timing should be done on the sender's clock exclusively;
- be able to deliver an estimation within a few RTTs since most connections are short-lived.

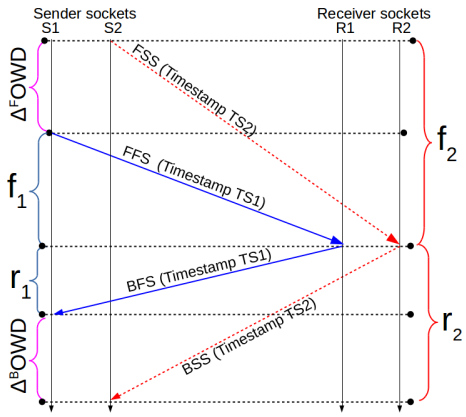


Fig. 1. Illustration of used notations for two subflows.

Briefly, the OWD estimator algorithm we propose is such that once it judges its $\tilde{\Delta}^F OWD$ estimation as accurate enough, it is able to identify the backward slow path and provide improved estimations of OWDs, as compared to halving the RTT.

In the following, we first present a few notations related to our delay model, then we present our algorithm.

A. Delay model

This section exposes a few notations for a baseline 2-subflow case. We make no assumption about the number of interfaces of the host (mono or multihomed) or the physical paths followed by the subflows (i.e., partially or fully disjoint). The disjoint case seems popular though, as it is likely to exhibit a higher difference between the OWDs, as it is the case for smartphones with both cellular and wifi interfaces.

Let $i \in \{1, 2\}$ be the subflow index, f_i and r_i be respectively the forward and reverse transfer delay on subflow i.

The following notations are illustrated in Figure 1, let the OWD of the Forward Fast Subflow (FFS), the Forward Slow Subflow (FSS), the Backward Fast Subflow (BFS) and the Backward Slow Subflow (BSS) be respectively:

$$f_{FFS} = \min_i(f_i) \tag{1}$$
$$f_{FSS} = \max_i(f_i) \tag{2}$$
$$r_{BFS} = \min_i(r_i) \tag{3}$$
$$r_{BSS} = \max_i(r_i) \tag{4}$$

It is worth noting that the Forward Fast Subflow (FFS) can be a BSS, i.e., a subflow is not necessary the shortest in both forward and backward directions. Obviously, the RTT of subflow i can be computed as:

$$RTT_i = f_i + \text{processing delay} + r_i \tag{5}$$

f_i and r_i can be decomposed into a deterministic and a stochastic part. The deterministic part corresponds to the time needed for a bit to propagate through its medium (supposing the route does not change during the connection) while the stochastic part refers to generic queuing delays. The processing delay refers to the time taken at each end-host for the packet to be actually sent or received at the application layer. We assume that the processing delay is negligible in (5) and thus we neglect it in the following. By analogy with TCP, this means that the Nagle algorithm [23] (which prevents TCP from sending many small packets) and delayed ACKs [24] (TCP receiver waits a certain amount of received packets or a timeout before acknowledging packets) should be disabled to keep the processing delay negligible. The sending buffer should also be empty for the same reason, except if hardware timestamping is in use. Duplicate acknowledgments caused by probes should not trigger retransmissions either (it is already true in MPTCP). Packet loss of any packet during a round can be detected and results from the round dismissed. The forward and backward delta delay are then defined as:

$$\Delta^F OWD = f_{FSS} - f_{FFS} \tag{6}$$
$$\Delta^B OWD = r_{BSS} - r_{BFS} \tag{7}$$

Both values are computed so that they are not negative.

The sender can easily retrieve the RTT, but it does not know f_i or r_i and traditionally assumes that $f_i = r_i = RTT_i/2$. In the following paragraph, we describe our estimator algorithm allowing to alleviate this issue.

B. Algorithm

As mentioned above, we aim to rely only on the sender's clock and require no cooperation with network elements. Before describing the algorithm, we need to highlight the following assumptions:

1) The clock precision is at least one magnitude higher than the maximum RTT.
2) The hosts can send packets at a locally precise determined time.
3) The server acknowledges every received packet immediately.
4) Timestamps can be embedded into packets

1) ensures that we can measure accurately enough the time we want to estimate. 2) may be more or less feasible, but is required to pace packet emission. 3) is assumed for convenience, but the algorithm could work with this additional constraint at the expense of precision. 4) ensures the host can identify each probe, to detect and discard rounds where probes arrive in disorder or are lost.

The first objective of the algorithm is to compute $\tilde{\Delta}^F OWD$, then it can compute a reverse $\tilde{\Delta}^B OWD$ value and finally provide an estimation for OWDs. The key idea to deduce the $\Delta^F OWD$ is to forcefully create and detect head of line blocking at the receiver from the sender side, as depicted in Figure 2 (the receive buffer is represented between square brackets). This is done by sending a packet on the FSS, the sender then waits for the currently estimated $\tilde{\Delta}^F OWD$ and sends a train of probes on the FFS until these probes match either the head of line blocking pattern in Figure 3(c) or that in Figure 3(d), i.e. packets sent on the FFS frame the

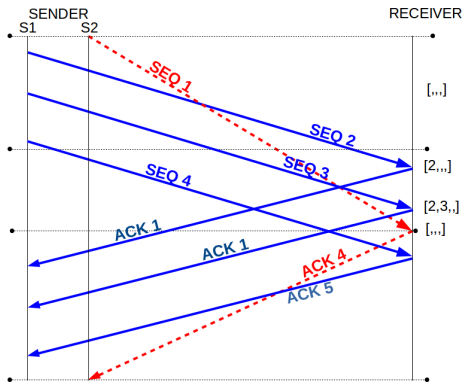
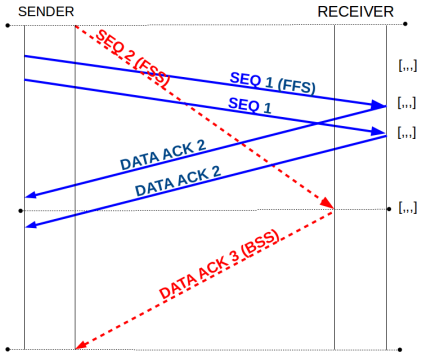


Fig. 2. An example of head of line blocking.

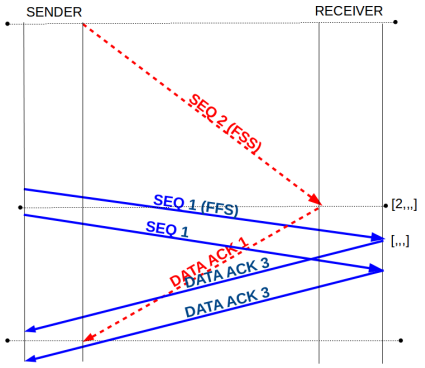
packet arrival on the FSS. Every probe embed its own unique timestamp in order to identify itself.

More precisely, the algorithm runs in two steps:

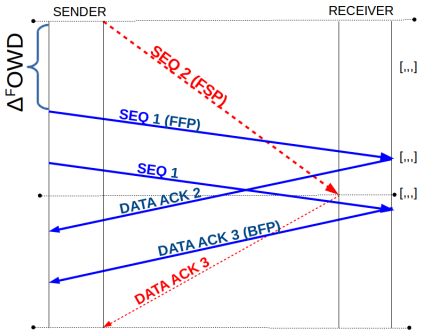
1) the sender needs to determine which subflow is the FFS. To do so, it sends two packets (one on each subflow) with consecutive sequence numbers at the same time. The server acknowledges each packet with the highest in-order sequence number. Thus it is possible for the sender to deduce from the ACKs which packet arrived first at the remote host. Note that the FFS can be the BSS, i.e., the first packet to come back to the sender is not necessarily the one that arrived first at the remote node. This step is quite straightforward and can be obtained in one RTT as described in the algorithm 1 line 2.
2) the sender then computes $\tilde{\Delta}^F OWD$. This is an iterative process that can run as long as FFS and FSS remain the same, e.g., the algorithm could restart if RTT varies too much. To achieve this, the sender sends a packet with sequence number $X+1$ on the FSS, then sends probes (two for instance in Figure 3) with sequence number X on the FFS with a delay close to the current estimation of $\Delta^F OWD$. There are four different patterns of packet arrival order at the receiver, all of them being visible on Figure 3. Depending on the case, the current $\tilde{\Delta}^F OWD$ is updated accordingly:
   - On Figure 3(a), all probes arrive before the packet on the FSS. The sender can deduce it from the head of line blocking exhibited by the acknowledgments of the probes. If the packet on the slow subflow arrives before any of the probes on the FFS, then the server would acknowledge that probe with the sequence number three. The sender can deduce from this that the current delay before sending the probe is underestimated.
   - Contrary to the previous case, in Figure 3(b) all probes arrive after the packet on the FSS. The sender receives an out-of-order ACK on the FSS. This means that all probes arrived after the packet on the FSS, i.e., the current $\tilde{\Delta}^F OWD$ is too high and



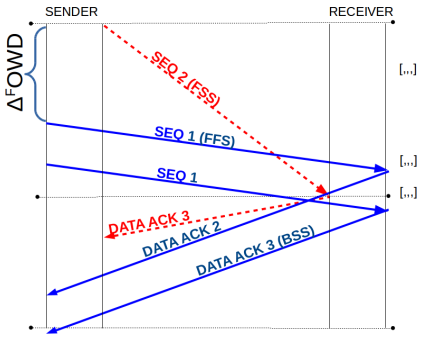
(a) Early probing: $\tilde{\Delta}^F OWD$ is too low.



(b) Late probing: $\tilde{\Delta}^F OWD$ is too high.



(c) Identical forward and backward fast subflows. The algorithm has converged to a valid $\tilde{\Delta}^F OWD$.



(d) Different forward and backward fast subflows.The algorithm has converged to a valid $\tilde{\Delta}^F OWD$.

Fig. 3. Exhaustive list of possibilities.

should be decreased.

- In Figure 3(c) are the cases that allow to deduce more precisely the $\Delta^F OWD$, i.e., probes framing the arrival of the packet on the FSS. The sender identifies these cases when it receives successive probes with different ACK numbers. To compute the forward delay of the FFS, we consider that the ACKs on each path were concurrently sent by the server as shown on the similar Figure 1 - so the estimations change as follows:

$$\tilde{f}_{FFS} = \frac{RTT_{FFS}}{2} \tag{8a}$$

$$\tilde{\Delta}^F OWD = TS2 - TS1 \tag{8b}$$

  where TS2 and TS1 are the timestamps exposed in Figure 1. Reverse OWDs are then deduced from the RTTs as shown later.
- Figure 3(d) case differs from Figure 3(c) case in that the FFS is the BSS. To compute the forward delay of the FFS, we consider that the ACKs on each path were concurrently sent by the server so the FFS OWD estimation becomes:

$$\tilde{f}_{FFS} = \frac{RTT_{FSS} - \tilde{\Delta}^F OWD}{2} \tag{9}$$

To summarize, when probes frame the packet arrival on the FSS, the value $\tilde{\Delta}^F OWD$ may be considered as correct and allows to compute estimations for the following values:

$$\tilde{f}_{FFS} = \min(RTT_{FFS}, RTT_{FSS} - \tilde{\Delta}^F OWD) \tag{10}$$

$$\tilde{f}_{FSS} = \tilde{\Delta}^F OWD + \tilde{f}_{FFS} \tag{11}$$

$$\tilde{r}_{BFS} = RTT_{FFS} - \tilde{f}_{FFS} \tag{12}$$

$$\tilde{r}_{BSS} = RTT_{FSS} - \tilde{f}_{FSS} \tag{13}$$

IV. Simulation results

We have chosen to implement the algorithm in a network simulator rather than doing real experiments because comparing delay estimations require very high precision values, we would have needed either synchronized clocks or complex tunneling to use the same host as client and server. Both are no easy task so we have implemented the algorithm in a slightly modified version of the network simulator ns3 [25] (version 3.20). Ns3 is a well-maintained open-source event-driven packet-level network simulator.

MPTCP being unavailable in vanilla ns3, we simulated its behavior via two custom applications - client and server - over UDP. The client application sends timestamped packets with a sequence number over the different subflows. Upon reception, the server generates a reply containing the received timestamp, plus the server own timestamp and the highest in-order sequence number it received. The server timestamps allow the computation of the real OWDs (i.e. these values are used for plotting but are ignored by the algorithm).

On the Internet, the asymmetry in OWDs can originate from several reasons, asymmetric bandwidths or asymmetric

Algorithm 1 Instance of the algorithm with three probes

$interval \leftarrow 3ms$ ▷ Time interval between probes on FFS

procedure FindForwardFastSubflow($seqNb$) ▷ Returns a tuple (FFS, FSS)
  SendPacket(1, $seqNb$)
  SendPacket(2, $seqNb + 1$)
  Wait for both acks
  if (Ack received on path 0) = $seqNb$ **then return** $(1, 2)$
  else return $(2, 1)$
  end if
end procedure

procedure SendPacket($pathId$, seq)
  Send sequence number seq on path $pathId$
end procedure

procedure StartEstimationRound(FFS, FSS, $deltaEstimation$)
  SendPacket(FSS, $roundLowestSeqNb + 1$, 0)
  Wait for $Max(deltaEstimation - interval, 0)$ ▷ Wait for a positive duration
  SendPacket(FFS, $roundLowestSeqNb$)
  Wait for $interval$
  SendPacket(FFS, $roundLowestSeqNb$)
  Wait for $interval$
  SendPacket(FFS, $roundLowestSeqNb$)
end procedure

Begin
  $roundLowestSeqNb \leftarrow 0$
  $\tilde{\Delta}^f OWD$ = 0 ▷ Start with an estimation of 0
  $FFS, FSS \leftarrow$ FindForwardFastSubflow($roundLowestSeqNb$)
  while no drastic change in RTT **do**
    $roundLowestSeqNb \leftarrow roundLowestSeqNb+2$
    StartEstimationRound(FFS, FSS, $\tilde{\Delta}^f OWD$)
    Wait for all acknowledgments
    ▷ In case of losses, a timeout restarts a new round
    $situation \leftarrow$ DeduceSituationFromAcks ▷ See Figure 3
    if $situation = Figure3(a)$ **then** ▷ Early probes
      $\tilde{\Delta}^f OWD \leftarrow \tilde{\Delta}^f OWD + interval$
    else if $situation = Figure3(b)$ **then** ▷ Late probes
      $\tilde{\Delta}^f OWD \leftarrow \tilde{\Delta}^f OWD - interval$
    else ▷ Probes on FFS framed packet arrival on FSS
      $\tilde{\Delta}^f OWD \leftarrow$ Average delay of the 2 framing probes
    end if
  end while
  Restart algorithm from the beginning
End

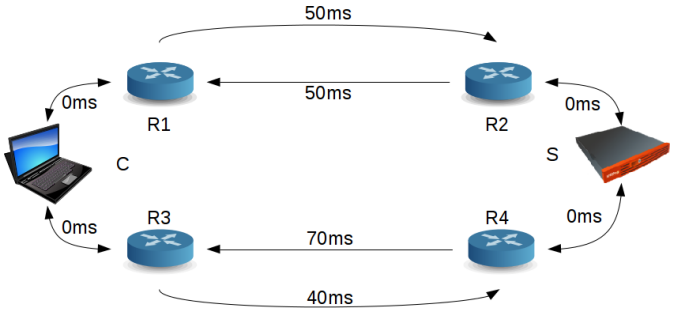


Fig. 4. Test topology with asymmetric paths.

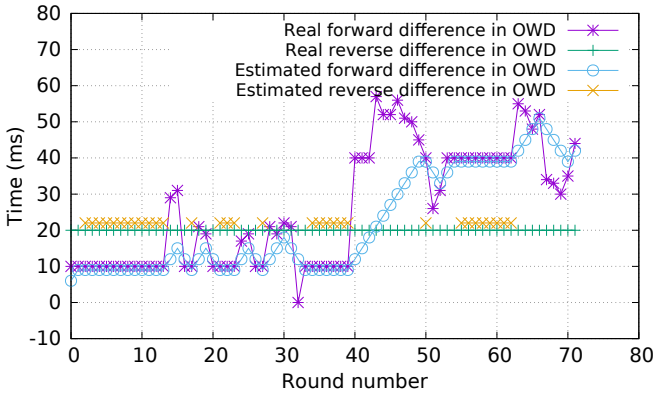


Fig. 5. Real and estimated ΔOWDs (forward and backward).

.

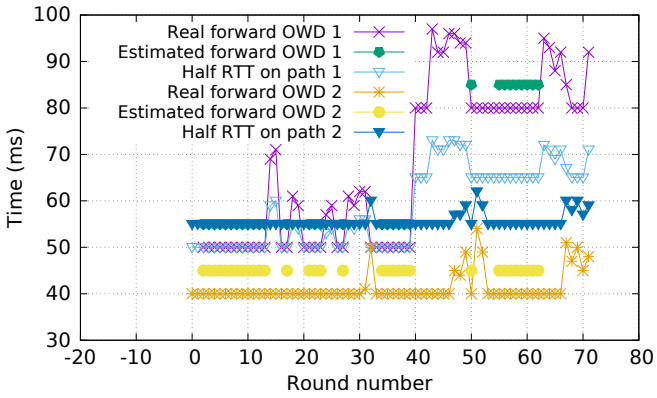


Fig. 6. Real and estimated forward OWDs.

propagation delays. We test both of them via binding one on/off TCP application on each client interface, to create jitter. The topology used is visible on Figure 4 and the queue size of the routers is set in packet unit. The source code of the simulation is available at [26].

A. Results

While Figure 5 displays $\tilde{\Delta}^F OWD$ at all times in order to better understand the estimation update process, it is worth noting that this estimation should be considered valid only when $\tilde{\Delta}^B OWD$ is plotted as well. This is particularly visible from rounds 40 to 50. Around round 40, we simulate a rerouting event by adding a 30ms delay from R1 to R2. The algorithm detects the situation in 3(a) and reacts according to the algorithm 1 line 31 and thus increases the $\tilde{\Delta}^F OWD$. Notice that no backward estimation is plotted at that time, which means the algorithm has not converged yet and that the current estimation sets only a minimum. In the range 40 to 50, where the FFS is different from the BFS, rounds last as much as $max(RTT_{FSS}, \tilde{\Delta}^F OWD + RTT_{FFS})$ which should be close to the RTT_1. Thus the algorithm converged to the new $\Delta^F OWD$ in ten RTTs. While our parameters adopted a conservative approach (starting with an estimation equal to zero, constant delay between probes), a dichotomic or cubic approach may allow for a faster convergence.

In Figure 6 the estimations are closer to the real OWD than halving the RTT - when they exist. For instance in round 60, the forward delay estimation on path one is 10ms more precise. Under low-jitter conditions, this should be true for all situations where the FFS is the same as the BFS, else it depends on the per-direction delay difference between subflows. The more difference there is, the more gain the algorithm should exhibit compared to halving the RTT.

B. Discussion

Our algorithm causes sporadic packet arrival disorder in order to deduce the $\Delta^F OWD$. This information then allows the sender to prevent constant packet arrival disorder. This may look contradictory at first but the packet arrival disorder provoked by the algorithm should occur less frequently then it would without the algorithm, thus proving beneficial in the long run. To mitigate the consequences of packet arrival disorder, one can envision the use of an MPTCP SACK option.

If we consider the adoption of such a scheme in MPTCP, the standard as well as its Linux implementation, integrating the described algorithm would require no compatibility breaking change since the linux implementation already acknowledges each packet. Pacing requires the sender to send packets according to a precise timing, which may be impaired by the numerous batching mechanisms in place at the different layers, be it Nagle algorithm or TCP segmentation offload. This may represent an additional concern.

V. Conclusion

More precision in the estimation of OWDs could improve the performance of various protocols. While single path protocols are limited in options to improve and use that estimation, multipath protocols that require an even more correct estimation hopefully provide additional possibilities. We devised a novel mechanism which has few requirements and tested it against asymmetric delays in the network simulator ns3. While the algorithm was applied to two subflows, it could run with more subflows, either running on appropriate pairs of subflows or sending concurrent probes on several subflows.

As future work, we intend to formalize the possible gain, design a scheduler accordingly to test it on a real network.

ACKNOWLEDGEMENT

This work was partially funded by the ANR LISP-Lab project (lisp-lab.org - Grant No: ANR-13-INFR-0009), the FUI 15 project RAVIR (http://www.ravir.io) and the EIT ICT-Labs Future Networking Services action line (eitictlabs.eu).

The authors would like to thank Tommaso Pecorella for the ns3 support.

REFERENCES

[1] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," in *Proceedings of the conference on Communications architectures, protocols and applications (SIGCOMM)*, 1994.

[2] M. Aylene and C. Weigle, "Investigating the Use of Synchronized Clocks in TCP Congestion Control," Ph.D. dissertation, University of North Carolina at Chapel Hill, 2003.

[3] R. Stewart, Q. Xie, K. Morneault, and Al., "Stream Control Transmission Protocol," *RFC2960*, 2000.

[4] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines," *RFC6824*, 2011.

[5] S. Ferlin and T. Dreibholz, "Multi-Path Transport over Heterogeneous Wireless Networks: Does it really pay off?" in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, 2014.

[6] D. Kaspar, "Multipath Aggregation of Heterogeneous Access Networks," Ph.D. dissertation, Faculty of Mathematics and Natural Sciences at the University of Oslo, 2012.

[7] F. Yang and P. Amer, "Using One-way Communication Delay for In-order Arrival MPTCP Scheduling," in *Proceedings of Chinacom*, 2014.

[8] ——, "Non-renegable Selective Acknowledgments (NR-SACKs) for MPTCP," in *27th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2013.

[9] V. Paxson, "On calibrating measurements of packet transit times," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, 1998, pp. 11–21.

[10] "Global positioning system standard positioning service performance standard 4," no. September, 2008. [Online]. Available: http://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf

[11] "Precision Time Protocol (IEEE 1588)." [Online]. Available: http://www.nist.gov/el/isd/ieee/ieee1588.cfm

[12] D. Mills, U. Delaware, J. Martin, J. Burbank, and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification," *RFC5905*, pp. 1–110, 2010.

[13] A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, and Z. M. Mao, "A measurement study of internet delay asymmetry," in *Lecture Notes in Computer Science*, vol. 4979 LNCS, 2008, pp. 182–191.

[14] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, "A measurement study on the impact of routing events on end-to-end internet path performance," *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2006.

[15] V. Paxson, "Measurements and analysis of end-to-end Internet dynamics," Ph.D. dissertation, University of California at Berkeley, 1998.

[16] V. Jacobson, R. Bradeb, and D. Borman, "TCP Extensions for High Performance," *RFC1323*, 1992.

[17] R. Scheffenegger, M. Kuehlewind, and B. Trammell, "Additional negotiation in the TCP Timestamp Option field during the TCP handshake," *draft-scheffenegger-tcpm-timestamp-negotiation-05*, 2013.

[18] J.-H. C. J.-H. Choi and C. Y. C. Yoo, "Analytic end-to-end estimation for the one-way delay and its variation," in *Second Consumer Communications and Networking Conference (CCNC)*. IEEE, 2005.

[19] O. Gurewitz and M. Sidi, "Estimating one-way delays from cyclic-path delay measurements," in *Conference on Computer Communications (INFOCOM)*. IEEE, 2001.

[20] E. P. Ribeiro and V. C. M. Leung, "Asymmetric path delay optimization in mobile multi-homed SCTP multimedia transport," *Proceedings of the 1st ACM workshop on Wireless multimedia networking and performance modeling - WMuNeP*, 2005.

[21] F. Song, H. Zhang, S. Zhang, F. Ramos, and J. Crowcroft, "An estimator of forward and backward delay for multipath transport," *Technical report, University of Cambridge*, no. 747, 2009.

[22] D. Zhou, H. Li, and J. Li, "Analysis of re-sequencing buffer overflow probability based on stochastic delay characteristics," *IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Sep. 2013.

[23] J. Nagle, "Congestion Control in IP/TCP Internetworks," *RFC896*, 1984.

[24] R. Braden, "Requirements for Internet Hosts – Communication Layers," *RFC1122*, 1989.

[25] "Ns3 official website." [Online]. Available: www.nsnam.org

[26] "Ns3 simulation source code." [Online]. Available: https://github.com/teto/pacing