ICCRG Working Group
Internet-Draft
Intended status: Experimental
Expires: September 10, 2020

O. Bonaventure
M. Piraux
Q. De Coninck
UCLouvain
M. Baerts
Tessares
C. Paasch
Apple
M. Amend
Deutsche Telekom

March 09, 2020

# Multipath schedulers
draft-bonaventure-iccrg-schedulers-00

## Abstract

This document proposes a series of abstract packet schedulers for multipath transport protocols equipped with a congestion controller.

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

The Internet was designed under the implicit assumption that hosts are equipped with a single network interface while routers are equipped with several ones. Under this assumption, an Internet host is usually identified by the IP address of its network interface.

This assumption does not hold anymore today for two reasons. First, a growing fraction of the Internet hosts are equipped with several network interfaces, usually through different datalink networks. These multihomed hosts are reachable via different IP addresses. Second, a growing fraction of the hosts that are attached through a single network interface are dual-stack and are thus reachable over both IPv4 and IPv6.

Several Internet transport protocols have been extended to leverage the different paths that are exposed on such hosts: Multipath TCP [RFC6824], the load sharing extensions to SCTP [I-D.tuexen-tsvwg-sctp-multipath], Multipath DCCP [I-D.amend-tsvwg-multipath-dccp] and Multipath QUIC [I-D.deconinck-quic-multipath]. These multipath transport protocols differ in the way they are organized and exchange control information and user data. However, they all include algorithms to handle three problems that any multipath transport protocol needs to solve:

- Congestion controller
- Path manager
- Packet scheduler

o Packet re-assembly

From a congestion control viewpoint, the main concern for a multipath transport protocol is that a multipath connection should not be unfair to single-path transport connections that share a common bottleneck. This problem can be solved by coupling the congestion windows of the different paths. The solution proposed in [RFC6356] is applicable to any transport protocol. Beside providing fairness, congestion control can also be a valuable input for different kind of traffic distribution algorithm within a packet scheduler. Typically metrics like RTT and available capacity can be derived.

A multipath transport protocol uses different flows during the lifetime of a connection. The Path Manager contains the logic that regulates the creation/deletion of these flows. This logic usually depends on the requirements of the application that uses the multipath transport. Some applications use multipath in failover situations. In this case, the connection can use one path and the path manager can create another path when the primary one fails. An application that wishes to share its load among different paths can request the path manager to establish different paths in order to simultaneously use them during the connection. Many path managers have been proposed in the literature [CONEXT15], but these are outside the scope of this document.

The packet scheduler is the generic term for the algorithm that selects the path that will be used to transmit each packet on a multipath connection. This logic is obviously only useful when there are at least two active paths for a given multipath transport connection. A variety of packet schedulers have been proposed in the literature [ACMCS14] and implemented in multipath transport protocols. Experience with multipath transport protocols shows that the packet scheduler can have a huge impact on the performance achieved by such protocols.

Packet re-assembly or re-ordering in multipath transport has the functionality to equalize the effect of packet scheduling across paths with different characteristics and restore the original packet order to a certain extent. Obviously, packet re-assembly is the counterpart of packet scheduling and located at the far end of the multipath transport. However, packet scheduling schemes exists which render the re-assembly superfluous or lowering at least its effort.

In this document, we document a series of multipath packet schedulers that are known to provide performance that matches well the requirements of specific applications. To describe these packet schedulers, we assume an abstract transport that is briefly presented in Section 2. In Section 3 we describe the challenges and constraints around a multipath scheduler. Finally, we describe the different schedulers in Section 4. To keep the description as simple and intuitive as possible, we assume here multipath connections that are composed of two paths, a frequent deployment scenario for multipath transport. This does not restrict the proposed schedulers to using only two paths. Implementations are encouraged to support more than 2 paths. We leave the discussion on how to adapt these abstract schedulers to concrete multipath transport protocols in future drafts.

## 2. An abstract multipath transport protocol

For simplicity, we assume a multipath transport protocol which can send packets over different paths. Some protocols such as Multipath TCP [RFC6824] support active and backup paths. We do not assume this in this document and leave the impact of these active/backup paths in specific documents.

Furthermore, we assume that there are exactly two active paths for the presentation of the packet schedulers. We consider that a path is active as long as it supports the transmission of packets. Meaning, A Multipath TCP subflow TCP segment with the FIN or RST flags set is not considered as an active path. Other constraints are possible on whether or not a path is active. These are specific to the scheduler and vary depending on the goal of the scheduler. An example of these is that when a path has experienced a certain number N of retransmission timeouts, the path can be considered inactive.

We assume that the transport protocol maintains one congestion controller per path as in [RFC6356]. We do not assume a specific congestion controller, but assume that it can be queried by the packet scheduler to verify whether a packet of length l would be blocked or not by the congestion control scheme. A window-based congestion controller such as [RFC6356] can block a packet from being transmitted for some time when its congestion window is full. The same applies to a rate-based congestion controller although the latter could indicate when the packet could be accepted while the former cannot.

We assume that the multipath transport protocol maintains some state at the connection level and at the path level. On both level, the multipath transport protocol will maintain send and receive windows, and a Maximum Segment Size that is negotiated at connection establishment.

It may also contain some information that is specific to the application (e.g. total amount of data sent or received) and information about non-active flows. At the path level, we expect that the multipath transport protocol will maintain an accurate estimation of the round-trip-time over that path, possibly a send/receive window, per path MTU information, the state of the congestion controller, and optionally information that is specific to the application or the packet scheduler (e.g. priority for one path over another one).

## 3. Packet scheduling challenges

Packet scheduling tries to balance different quality of service goals with different constraints of the paths. The balance depends on which of the goals or constraints is the primary factor for the experience the application is aiming for. In the following we list these goals and constraints and conclude by how they can influence each other.

Each path can be subject to a different cost when transmitting data. For example, a path can introduce a per-byte monetary cost for the transmission (e.g., metered cellular link). Another cost can be the power consumption when transmitting or receiving data. These costs are imposing restrictions on when a path can be used compared to the lower-cost path.

A goal for many applications is to reduce the latency of their transaction. With multiple paths, each path can have a significantly different latency compared to the other paths. It is thus crucial to schedule the traffic on a path such that the latency requirements of the application are satisfied.

Achieving high throughput is another goal of many applications. Streaming applications often require a minimum bit rate to sustain playback. The scheduler should try to achieve this bit rate to allow for a flawless streaming experience. Beyond that, adaptive streaming requires also a more stable throughput experience to ensure that the bit rate of the video stream is consistent. When sending traffic over multiple paths the bit rate can experience more variance and thus the scheduler for such a streaming application needs to take precautions to ensure a smooth experience.

Finally, transport protocols impose a receive-window that signals to the sender how much data the application is willing to receive. When the paths have a large latency difference, a multipath transport can quickly become receive-window limited. This limitation comes from the fact that a packet might have been sent on a high-latency path. If the transport imposes in-order delivery of the data, the receiver needs to wait to receive this packet over the high-latency path before providing it to the application. The sender will thus become receive-window limited and may end up under-utilizing the low-latency path. This can become a major challenge when trying to achieve high throughput.

All of these quality of service goals and constraints need to be balanced against each other. A scheduler might decide to trade latency for higher throughput. Or reduce the throughput with the goal of reducing the cost.

## 4. Packet schedulers

The packet scheduler is executed every time a packet needs to be transmitted by the multipath transport protocol. A packet scheduler can consider three different types of packets:

- o packets that carry new user data
- o packets that carry previously transmitted user data
- o packets that only carry control information (e.g., acknowledgements, address advertisements)

In Multipath TCP, the packet scheduler is only used for packets that carry data. Multipath TCP will typically return acknowledgements on the same path as the one over which data packets were received. For Multipath QUIC, the situation is different since Multipath QUIC can acknowledge over one path data that was previously received over another path. In Multipath TCP, this is only partially possible. The subflow level acknowledgements must be sent on the subflow where the data was received while the data-level acknowledgements can be sent over any subflow.

This document uses the Python language to represent multipath schedulers. A multipath scheduler is represented as a Python function. This function takes the length of the next packet to schedule as argument and returns the path on which it will be send. A path is represented as a Python class with the following attributes:

- o srtt: The smoothed RTT of the path [RFC6298].
- o cc_state: The state of the congestion controller, i.e. either slow_start, congestion_avoidance or recovery.
- o blocked(l): A function indicating whether a packet of length l would be rejected by the congestion controller.

The schedulers presented can be executed in a simulator [MultipathSim] implementing the abstract multipath protocol presented in Section 2. It can be used to simulate a file transfer between a client and a server over multiple paths.

### 4.1. Round-Robin

We use the Round-Robin scheduler as a simple example to illustrate how a packet scheduler can be specified, but we do not recommend its usage. Experiments with Multipath TCP [ACMCS14] indicate that it does not provide good performance.

This packet scheduler uses one additional state at the connection level: last_path. This stores the identifier of the last path that was used to send a packet. We assume that the paths are identified by an integer. The scheduler is defined by the code shown in Figure 1.

```
class RoundRobin(Scheduler):
    """ Chooses an available path in a round-robin manner. """
    last_path: Optional[Path] = None

    def schedule(self, packet_len: int):
        if self.last_path in self.paths:
            next_idx = self.paths.index(self.last_path) + 1
        else:
            next_idx = 0
        sorted_paths = self.paths[next_idx:] + self.paths[:next_idx]
        for p in sorted_paths:
            if not p.blocked(packet_len):
                self.last_path = p
                return p
```

Figure 1: A simple Round Robin scheduler

This scheduler does not distinguish between the different types of packets. It iterates over the available paths and sends over the ones whose congestion window is open.

### 4.2. Strict Priority

The Strict Priority scheduler's aim is to select paths based on a priority list. Some paths might go through networks that are more expensive to use than others. Then the idea is to select the path with the highest priority if it is available before looking at others by priority. This scheduler is described by the code shown in Figure 2.

```
class StrictPriority(Scheduler):
    """ Chooses the first available path in a priority list of paths. """

    def schedule(self, packet_len: int):
        for p in sorted(self.paths, key=lambda p: p.priority, reverse=True):
            if not p.blocked(packet_len):
                return p
```

Figure 2: A simple Strict Priority scheduler

This scheduler can face performance issues if, compared to others, paths with high priority accept a lot of data but delivered packets with a high latency. When the path is experiencing bufferbloat, the receiver has to store packets for a long time in its buffers to ensure an in-order delivery. It is then recommended to cover these cases in the scheduler implementation with the help of the congestion control algorithm.

## 4.3. Delay Threshold

The Delay Threshold scheduler selects the first available path with a smoothed round-trip-time below a certain threshold. The goal is to keep the RTT of the multipath connection to a small value and avoid having the whole connection impacted by "bad" paths. A prototype is shown in Figure 3.

```
@dataclass
class DelayThreshold(Scheduler):
    """ Chooses the first available path below a certain delay threshold. """
    threshold: float

    def schedule(self, packet_len: int):
        for p in self.paths:
            if p.srtt < self.threshold and not p.blocked(packet_len):
                return p
```

Figure 3: A simple Delay Threshold scheduler

This kind of protection can of course be added to other existing schedulers.

## 4.4. Lowest round-trip-time first

The Lowest round-trip-time first scheduler’s goal is to minimize latency for short flows while at the same time achieving high throughput for long flows [ACMCS14]. To handle the latency differences across the paths when being limited by the receive-window, this scheduler deploys a fast reinjection mechanism to quickly recover from the head-of-line blocking.

At each round, the scheduler iterates over the list of paths that are eligible for transmission. To decide whether or not a path is eligible, a few conditions need to be satisfied:

- The congestion window needs to provide enough space for the segment

- The path is not in fast-recovery or experiencing retransmission timeouts

Among all the eligible paths, the scheduler will choose the path with the lowest RTT and transmit the segment with the new data on that path. Figure 4 illustrates a simple lowest RTT scheduler which does not include fast reinjections.

```
class LowestRTTFirst(Scheduler):
    """ Chooses the first available path with the lowest RTT. """

    def schedule(self, packet_len: int):
        # Sort paths by increasing SRTT
        for p in sorted(self.paths, key=lambda path: path.srtt):
            if not p.blocked(packet_len) \
               and p.cc_state != 'recovery':
                return p
```

Figure 4: A simple Lowest RTT First scheduler

To handle head-of-line blocking situations when the paths have a large delay difference the scheduler uses a strategy of opportunistic retransmission and path penalization as described in [NSDI12].

Opportunistic retransmission kicks in whenever a path is eligible for transmission but the receive-window advertised by the receiver prevents the sender from transmitting new data. In that case the sender can transmit previously transmitted data over the eligible path. To overcome the head-of-line blocking the sender will thus transmit the packet at the head of the transmission queue over this faster path (if it hasn't been transmitted on this particular path yet). This packet has thus a chance to quickly reach the receiver and fill the hole created by the head-of-line blocking.

Whenever the previously mentioned mechanism kicks in, it is and indication that the path's round-trip-time is too high to allow the path with the lower RTT to fully use its capacity. We thus should reduce the transmission rate on this path. This mechanism is called penalization and is achieved by dividing the congestion window by 2.

[comment:] ## Out-of-order transmission for in-order arrival

ASCII figure

Figure 5: A simple figure

## 5. Informative References

[ACMCS14] Paasch, C., Ferlin, S., Alay, O., and O. Bonaventure, "Experimental Evaluation of Multipath TCP Schedulers", Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop , n.d..

[CONEXT15] Hesmans, B., Detal, G., Barre, S., Bauduin, R., and O. Bonaventure, "SMAPP : Towards Smart Multipath TCP-enabled APPlications", CoNEXT '15: Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies , n.d..

[I-D.amend-tsvwg-multipath-dccp] Amend, M., Bogenfeld, E., Brunstrom, A., Kassler, A., and V. Rakocevic, "DCCP Extensions for Multipath Operation with Multiple Addresses", draft-amend-tsvwg-multipath-dccp-03 (work in progress), November 2019.

[I-D.deconinck-quic-multipath] Coninck, Q. and O. Bonaventure, "Multipath Extensions for QUIC (MP-QUIC)", draft-deconinck-quic-multipath-04 (work in progress), March 2020.

[I-D.tuexen-tsvwg-sctp-multipath] Amer, P., Becke, M., Dreibholz, T., Ekiz, N., Iyengar, J., Natarajan, P., Stewart, R., and M. Tuexen, "Load Sharing for the Stream Control Transmission Protocol (SCTP)", draft-tuexen-tsvwg-sctp-multipath-19 (work in progress), January 2020.

[MultipathSim] Piraux, M., "Multipath simulator for the IETF draft Multipath schedulers", n.d., <https://github.com/obonaventure/draft-schedulers/blob/scheduler_simulator.py>.

[NSDI12] Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP", 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12) , n.d..

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[RFC6356] Raiciu, C., Handley, M., and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols", RFC 6356, DOI 10.17487/RFC6356, October 2011, <https://www.rfc-editor.org/info/rfc6356>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Authors' Addresses

Olivier Bonaventure
UCLouvain

Email: Olivier.Bonaventure@uclouvain.be

Maxime Piraux
UCLouvain

Email: Maxime.Piraux@uclouvain.be

Quentin De Coninck
UCLouvain

Email: quentin.deconinck@uclouvain.be

Matthieu Baerts
Tessares

Email: Matthieu.Baerts@tessares.net

Christoph Paasch
Apple

Email: cpaasch@apple.com

Markus Amend
Deutsche Telekom

Email: markus.amend@telekom.de