A Programming Model for Application-defined Multipath TCP Scheduling

ACM/IFIP/USENIX Middleware 2017
Alexander Frömmgen

Application

Load MPTCP Scheduler

Network Stack

Userland
Kernel

Sending Queue

Programmable Scheduler

Send Data

Subflows

Alexander Frömmgen
Amr Rizk
Tobias Erbshäußer
Max Weller
Boris Koldehofe
Alejandro Buchmann
Ralf Steinmetz
Multipath TCP in a Nutshell

Server
Sending Queue

Scheduler

Subflow 1 WiFi

Subflow 2 LTE

Client
Application

Receive Queue

\[ p_6 \quad p_5 \]

\[ p_2 \quad p_1 \]
Multipath TCP in a Nutshell

1 Congestion Control
   Fairness on Joint Paths
   Server Sending Queue
   $p_6$ $p_5$
   Scheduler

2 Path Management
   Trigger Subflow Establishment
   Client
   Application
   Receive Queue
   $p_2$ $p_1$

3 TCP Options for MPTCP
   Control Information, e.g.,
   Data Sequence Numbers

4 Scheduler
   Map Packets on Subflows

Subflow 1 WiFi
Subflow 2 LTE
MPTCP Packet

Multipath TCP Scheduling

RTT = 40ms
BW = ...
P_IN_FLY = 0
CWND = 10

RTT = 10ms
BW = ...
P_IN_FLY = 0
CWND = 10

Images: https://www.youtube.com/watch?v=U2T1_VrTcoo

KOM – Multimedia Communications Lab
Multipath TCP Scheduling

Intuition: Schedule on Subflow with minimum round-trip time (RTT).

Packet 2
Packet 1
Scheduler

RTT = 40ms
BW = ...
P_IN_FLY = 0
CWND = 10

RTT = 10ms
BW = ...
P_IN_FLY = 0
CWND = 10

Subflow 2 LTE
Subflow 1 WiFi
Client

Images: https://www.youtube.com/watch?v=U2T1_VrTcoo
Multipath TCP Scheduling

Schedule on Subflow with minimum round-trip time (RTT).

Packet 1
- RTT = 10ms
- BW = ...
- P_IN_FLY = 10
- CWND = 10

Packet 11
- RTT = 40ms
- BW = ...
- P_IN_FLY = 0
- CWND = 10

Subflow 1 WiFi
Subflow 2 LTE

Scheduler
Client

Images: https://www.youtube.com/watch?v=U2T1_VrTcoo
Schedule on Subflow with minimum round-trip time (RTT).

which is not saturated (Congestion window larger than packets in flight).
Multipath TCP Scheduling
Motivating Example

Server
Sending Queue

\[ p_6 \quad p_5 \]

Scheduler

\[ \text{~RTT 40ms} \]

Subflow 2 LTE

\[ \text{~RTT 10ms} \]

Subflow 1 WiFi

Client
Application

\[ p_2 \quad p_1 \]

Receive Queue

Constant Bitrate Stream

Home Network WiFi and LTE
Germany

![Graph showing throughput over time for WiFi and LTE subflows.](image)
Multipath TCP Scheduling
Motivating Example

Server
Sending Queue

Scheduler

~RTT 40ms
Subflow 2 LTE

~RTT 10ms
Subflow 1 WiFi

Client
Application

Receive Queue

Constant Bitrate Stream
Home Network WiFi and LTE Germany

throughput [kB/s]

4 MB Stream

1 MB Stream

~RTT 40ms
~RTT 10ms

0 2 4 6 8 10 12 14

time [s]
## Multipath TCP Scheduler Overview

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Pref.</th>
<th>Available in Linux Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min RTT (default)</td>
<td>General Purpose</td>
<td>Binary</td>
<td>✔️</td>
</tr>
<tr>
<td>Round-Robin</td>
<td>Academic</td>
<td>Binary</td>
<td>✔️</td>
</tr>
<tr>
<td>Redundant</td>
<td>Thin Flows</td>
<td>Binary</td>
<td>✔️</td>
</tr>
<tr>
<td>Compensate Loss</td>
<td>Short Datacenter Flows</td>
<td>?</td>
<td>No</td>
</tr>
<tr>
<td>Video and Energy</td>
<td></td>
<td>?</td>
<td>No</td>
</tr>
<tr>
<td>DASH Video</td>
<td></td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

...
Multipath TCP Scheduler Overview
Great Concepts without MPTCP Implementation

The reference FIFO scheduler. We compare our cross-layer scheduler and the optimal solution to the reference FIFO scheduler, which mimics the current implementation of MPTCP without any cross-layer scheduler. The video cross-layer scheduler can be implemented either at the transport layer or at the application layer. In the following of the paper, we consider an implementation at the application level because it is more convenient. Here are some details.


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.


Research Questions

How can we systematically \textit{specify} and \textit{execute} MPTCP schedulers?

How can we \textit{enable application-defined} MPTCP scheduling?
Part I: Towards a Programmable Scheduler in the Network Stack
Model of the Scheduler Environment

Userland

Kernel

Application

Sending Queues

Scheduler

*operation, e.g., POP

Subflows
Model of the Scheduler Environment

Userland

Kernel

Sending Queue Q

Packets in Flight QU

Reinjection Queue RQ

Sending Queues

Application

Scheduler

Subflow sbf1

Subflow sbf2

Subflows

*p* operation, e.g., POP

0x08
Specify Multipath TCP Schedulers

**Requirements**

- Expressiveness
- Isolation
- Timely Execution

**Design Decisions**

1. Modelled Elements as Entities: Set of Subflows with their properties, Queues of Packets with their Properties
2. Declarative Packet and Subflow Selection (Filter, Min, Max)
3. No Recursion, No Functions, Limited Loops
4. Variables with Single Assignment, Implicit and Static Type System
5. No, One, or Multiple Packets per Scheduler Execution
Systematically Specify MPTCP Schedulers
Domain Specific Specification Language

Example: Preference-aware RTT-sensitive Scheduler

```
VAR sbfCandis = SUBFLOWS.FILTER(
  sbf => sbf.CWND > sbf.SKBS_IN_FLIGHT + sbf.QUEUED
  AND !sbf.TSQ_THROTTLED AND !sbf.LOSSY);

VAR backSbf = sbfCandis.FILTER(
  sbf => sbf.IS_BACKUP).MIN(sbf => sbf.RTT);

VAR nonBackSbf = sbfCandis.FILTER(
  sbf => !sbf.IS_BACKUP).MIN(sbf => sbf.RTT);

IF (nonBackSbf.RTT_MS > 100 AND backSbf.RTT_MS < 80) {
  backSbf.PUSH(Q.POP());
} ELSE {
  nonBackSbf.PUSH(Q.POP());
}
```
Systematically Specify MPTCP Schedulers
Domain Specific Specification Language

Example: Preference-aware RTT-sensitive Scheduler

1 VAR sbfCandis = SUBFLOWS.FILTER(
2 sbf => sbf.CWND > sbf.SKBS_IN_FLIGHT + sbf.QUEUED
3 AND !sbf.TSQ_THROTTLED AND !sbf.LOSSY);
4
5 VAR backSbf = sbfCandis.FILTER(
6 sbf => sbf.IS_BACKUP).MIN(sbf => sbf.RTT);
7 VAR nonBackSbf = sbfCandis.FILTER(
8 sbf => !sbf.IS_BACKUP).MIN(sbf => sbf.RTT);
9
10 IF (nonBackSbf.RTT_MS > R1 AND backSbf.RTT_MS < R2) {
11 backSbf.PUSH(Q.POP());
12 } ELSE {
13 nonBackSbf.PUSH(Q.POP());
14 }
Systematically Specify and Execute MPTCP Schedulers

Specified schedulers are executable in the Linux Kernel.
Application-aware Scheduling in Multi-Tenancy Cloud Environments

import socket
from progmp import ProgMp
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('10.0.0.2', 8080))
try:
    ProgMp.loadScheduler("python_api_example")
    ProgMp.setScheduler(s, "python_api_example")
except:
    print "Scheduler loading error."
ProgMp.setRegister(s, ProgMp.R1(), 50)
s.send("Multipath is awesome!")
Abstraction vs. Overhead

![Chart showing relative execution time per scheduled packet for different MinRTT configurations.]

- MinRTT Default
- MinRTT Interpreter
- MinRTT eBPF
Abstraction vs. Overhead

The runtime environment induces a small overhead, which is acceptable for most application scenarios.
## Part II: Design of Novel Multipath TCP Schedulers

<table>
<thead>
<tr>
<th></th>
<th>Preference-aware</th>
<th>Application-aware</th>
<th>Executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round-trip Time-aware</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Constant Bitrate Stream Scheduling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Redundant Scheduling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HTTP-aware Scheduling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>More in Paper and Under Review*</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

A Close Look at Redundant Schedulers

Sender

<table>
<thead>
<tr>
<th>Q</th>
<th>In Flight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq 6</td>
<td>Seq 2</td>
</tr>
<tr>
<td>Seq 7</td>
<td>Seq 3</td>
</tr>
<tr>
<td></td>
<td>Seq 4</td>
</tr>
<tr>
<td></td>
<td>Seq 5</td>
</tr>
</tbody>
</table>

Subflow 1 (RTT 50ms)

- Ack 2
- Seq 4
- Seq 3

Subflow 2 (RTT 150ms)

- Seq 5
- Seq 3

Receiver

- Seq 2

Should we send the fresh packet Seq 6 or the old packet Seq 5 when the acknowledgement Ack 2 arrives at the sender?
A Close Look at Redundant Schedulers

Should we send the fresh packet or the old packet when the acknowledgment arrives at the sender?

```c
FOREACH(VAR sbf IN sbfCandidates) {
    VAR skb = QU.FILTER(s => !s.SENT_ON(sbf).TOP);
    IF(skb != NULL) {
        sbf.PUSH(skb);
    } ELSE {
        sbf.PUSH(Q.POP());
    }
}
```

Prefer Per Subflow Fresh Packets

```c
IF(!sbfCandidates.EMPTY) {
    FOREACH(VAR sbf IN sbfCandidates) {
        sbf.PUSH(Q.TOP);
    }
    DROP(Q.POP());
}
```

Prefer Global Fresh Packets

ProgMP enables rapid specification and evaluation of schedulers.
A Close Look at Redundant Schedulers

ProgMP enables novel redundant schedulers, which outperform established approaches.
Side Note: How can we systematically compare and evaluate scheduler design decisions?

- Web Frontend to Manage Exp.
  - Add Config. and Env. Variations
  - Add Protocols and Algorithms
  - Fix / Improve Implementation

- Scalable Experiment Execution

- Inspect Results: Interactive Data Analysis and Exploration
  - Single Experiment Result
  - Environ. x Repet. Configurations

Iterative Refinements in the Research Process
Side Note: How can we systematically compare and evaluate scheduler design decisions?

A Framework for the Management, the Scalable Execution and Interactive Analysis of Extensive Network Experiments

https://maci-research.net
HTTP/2-aware Scheduling

ProgMP enables HTTP/2-aware Scheduling.
HTTP/2-aware Scheduling

Note that the large variance is partly caused by a high variance of the underlying real world environment.
Conclusion

We presented **the first programming model for Multipath TCP scheduling**.

- **Specification** and **execution** of MPTCP schedulers
- **Application-defined** MPTCP scheduling

We **proposed** and **evaluated** sophisticated novel schedulers.

- RTT-aware scheduler
- Constant bitrate schedulers
- Flavors of redundant schedulers
- HTTP-aware scheduler
- ...

We presented the first programming model for Multipath TCP scheduling.
Conclusion

We presented the first programming model for Multipath TCP scheduling. We proposed and evaluated sophisticated novel schedulers.

Try it! https://progmp.net
Questions
Multipath TCP in iOS 11
Interactive Mode

Low latency for low-volume interactive flows
Wi-Fi and cellular
Available in an upcoming Beta

http://www.tessares.net/highlights-from-advances-in-networking-part-1/
Configure the scheduler:

We have a modular scheduler infrastructure. At compile-time you can select the schedulers that should be compiled in.

At run-time you can select one of the compiled schedulers through the `sysctl net.mptcp.mptcp_scheduler`. You have the choice between:

- 'default': This scheduler is the default one. It will first send data on subflows with the lowest RTT until their congestion-window is full. Then, it will start transmitting on the subflows with the next higher RTT.
- 'roundrobin': This scheduler will transmit traffic in a round-robin fashion. It is configurable, how many consecutive segments should be sent with the tunable "num_segments" in the `sysfs` (default 1). Additionally, you can set the boolean tunable "cwnd_limited", to specify whether the scheduler tries to fill the congestion window on all subflows (true) or whether it prefers to leave open space in the congestion window (false) to achieve real round-robin (even if the subflows have very different capacities) (defaults to true). In case you are unsure, never ever enable this scheduler. Its performance is bad and it is only interesting for academic/testing purposes. The default scheduler is the best known up to today.
- 'redundant': This scheduler will try to transmit the traffic on all available subflows in a redundant way. It is useful when one wants to achieve the lowest possible latency by sacrificing the bandwidth.

**New in v0.92**: Alternatively, you can select the scheduler through the socket-option `MPTCP_SCHEDULER` (defined as 43) by doing:
Enabling per Application-aware Scheduling in Multi-Tenancy Cloud Environments

- Application, e.g., Webserver
- Application B
- Python Application
- JRE

Kernel Userspace

- Socket API
- Ext. API
- Isolated Scheduler ...

Application

- Parser / Optimizer / Repository
- Set Packet Property
- Set Register
- Choose Scheduler
- Registers
- Packet Scheduler
- Proc fs

Extended API

- proc fs

Isolated Scheduler Runtime Environment

- Isolated Scheduler