Application-defined Multipath TCP Scheduling

Frühjahrstreffen der GI Fachgruppen BS und KuVS
Amr Rizk

A version of this work was published at ACM/USENIX Middleware 2017
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
     - $p_2$, $p_1$
   - Receive Queue

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

4. **Scheduler**
   - Map Packets to Subflows
Multipath TCP Scheduling

**Server**
- Packet 1
- Scheduler

**Client**

**Subflow 1 WiFi**
- RTT = 10ms
- BW = ...
- P_IN_FLY = 0
- CWND = 10

**Subflow 2 LTE**
- RTT = 30ms
- BW = ...
- P_IN_FLY = 0
- CWND = 10

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

Intuition: Schedule on Subflow with minimum round-trip time (\textit{minRTT}).

Packet 2

Packet 1

Scheduler

Subflow 2 LTE

\begin{itemize}
  \item RTT = 30ms
  \item BW = ...
  \item P\_IN\_FLY = 0
  \item CWND = 10
\end{itemize}

Subflow 1 WiFi

\begin{itemize}
  \item RTT = 10ms
  \item BW = ...
  \item P\_IN\_FLY = 0
  \item CWND = 10
\end{itemize}

Client

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

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

- Subflow 2 LTE:
  - RTT = 30ms
  - BW = ...
  - P_IN_FLY = 0
  - CWND = 10

- Subflow 1 WiFi:
  - RTT = 10ms
  - BW = ...
  - P_IN_FLY = 10
  - CWND = 10

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

Schedule on Subflow with minimum round-trip time \(\text{minRTT}\).

which is not saturated (Congestion window > packets in flight).

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

Constant Bitrate Stream
Home Network WiFi and LTE Germany

Server
Sending Queue

\( p_5 \)
\( p_6 \)
Scheduler

\( \sim \text{RTT 30ms} \)
Subflow 2 LTE

\( \sim \text{RTT 10ms} \)
Subflow 1 WiFi

Client
Application
\( p_1 \)
\( p_2 \)
Receive Queue

\( \sim \text{RTT 30ms} \)

4 MB Stream

1 MB Stream

throughput [kB/s]

0 1000 2000 3000 4000

0 2 4 6 8 10 12 14
time [s]

Server
Sending Queue

\( p_5 \)
\( p_6 \)
Scheduler
Multipath TCP Scheduling
Real-world Experiment

Server
Sending Queue

\[ p_6 \quad p_5 \]

Scheduler

\[ \text{~RTT 30ms} \]
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

4 MB Stream

1 MB Stream

throughput [kB/s]

0 1000 2000 3000 4000

0 2 4 6 8 10 12 14
time [s]
# Multipath TCP Scheduler Overview

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Preferences</th>
<th>Available in Linux (Implementation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min RTT <em>(Default)</em></td>
<td>General Purpose</td>
<td>Binary</td>
<td>✓</td>
</tr>
<tr>
<td>Round-Robin</td>
<td>General Purpose</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>Energy preserving</td>
<td></td>
<td>?</td>
<td>No</td>
</tr>
<tr>
<td>DASH Video</td>
<td></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

The 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.


Goals (of this talk)

- Systematically **specify** and **execute** MPTCP schedulers
- Enable **application-defined** MPTCP scheduling
Part I: Towards a Programmable Scheduler in the Network Stack

Userland
Kernel

Application
Send Data

Network Stack

Sending Queue
Programmable Scheduler

Load MPTCP Scheduler
Subflows
Model of the Scheduler Environment

- Application
- Userland
- Kernel
- Sending Queues
- Scheduler
- Subflows

*operation, e.g., POP

PUSH

op*
Model of the Scheduler Environment

Userland
 Kernel

Sending Queue Q

Packets in Flight QU

Reinjection Queue RQ

Sending Queues

op*

Scheduler

*operation, e.g., POP

Application

PUSH

Subflow sbf1

Queue Q_{sbf1} p_1

Properties: rtt, cwnd

Subflow sbf2

Queue Q_{sbf2}

Prop.: rtt, cwnd, …

PUSH

PUSH

PUSH

op*

op*

p_1 p_3 p_2

p_1

Packets in Flight QU
Specifying Multipath TCP Schedulers

Requirements

Expressiveness  Timely Execution  Isolation

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
Back to the Motivating Example

Constant Bitrate Stream
Home Network WiFi and LTE Germany

How to use the Backup Flow?

~RTT 30ms
Subflow 2 LTE

~RTT 10ms

Server
Sending Queue

Client
Application

0 2 4 6 8 10 12 14

0 1000 2000 3000 4000

throughput [kB/s]
time [s]

WiFi LTE Default Default + backup

4 MB Stream

1 MB Stream
Example: Preference-aware **RTT-sensitive** Scheduler

```plaintext
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());
}
```
Example: Preference-aware **RTT-sensitive** Scheduler

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

Specified schedulers are executable in the Linux Kernel
Abstraction vs. Overhead

![Bar Chart]

Relative Execution Time per Scheduled Packet

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

link saturation at 1 Gbit/s
Abstraction vs. Overhead

The runtime environment induces an overhead, which is acceptable for most application scenarios.

link saturation at 1 Gbit/s
### Part II: Design of Novel Multipath TCP Schedulers

<table>
<thead>
<tr>
<th>Scheduling Type</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/2-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

When the acknowledgement \textcolor{red}{\textbf{Ack 2}} arrives at the sender:

\(\rightarrow\) should we send the fresh packet \textcolor{green}{\textbf{Seq 6}} or the old packet \textcolor{green}{\textbf{Seq 5}}
A Close Look at Redundant Schedulers

```
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 Global Fresh Packets

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

Prefer Per Subflow Fresh Packets
A Close Look at Redundant Schedulers

Prefer Per Subflow Fresh Packets

1. FOREACH(VAR sbf IN sbfCandidates) {
2.   VAR skb = QU.FILTER(s => !s.SENT_ON(sbf).TOP);
3.   IF(skb != NULL) {
4.     sbf.PUSH(skb);
5.   } ELSE {
6.     sbf.PUSH(Q.POP());
7.   }
8. }

Prefer Global Fresh Packets

1. IF(!sbfCandidates.EMPTY) {
2.   FOREACH(VAR sbf IN sbfCandidates) {
3.     sbf.PUSH(Q.TOP);
4. }

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

ProgMP enables novel redundant schedulers, which outperform established approaches.
HTTP/2-aware Scheduling

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

[Diagram showing content streams and scheduling options]

Left to right boxplots: Default, Single Path, HTTP/2-Aware

- Dependencies End
- Initial Page End
- Preferences
Conclusion

We presented a programming model for Multipath TCP scheduling

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

We proposed and evaluated novel MPTCP schedulers

- RTT-aware scheduler
- Constant bitrate schedulers
- Flavors of redundant schedulers
- HTTP/2-aware scheduler
- …
Conclusion

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

https://progmp.net
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

Inspect Results: Interactive Data Analysis and Exploration

Scalable Experiment Execution

Exp. Study

Config. x Exp.

Env.

Exp.

Single Experiment Result

Iterative Refinements
How can we systematically compare and evaluate scheduler design decisions?

Web Frontend to Manage Exp.

Add Config. and Env. Variations

Add Protocols

Scalable Experiment Execution

Exp. Study

Inspect Results: Interactive Data Analysis and Exploration

Single Experiment Result

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

https://maci-research.net

Iterative Refinements