My TCP ain’t your TCP - ain’t no TCP?

How new implementations speed up the internet ...
... and make engineers drink

Simon Lindermann
Miele & Cie KG / Freelancer
Demo Traces

https://cloud.local-area.network/index.php/s/7Ojmw9hhnDf9yBb

https://bit.ly/2Iae7A7 (Shortlink)
About me?

• Working and learning in IT since 2006
  • Employer: Miele Germany
  • Network Architect
  • Part time freelancer

Contact

@SimonLindermann
sl@local-area.network
Shark Pointer
LaserShark Pointer

What would an engineer do?
No revolution, but evolution

**Algorithms**
- TCP Tahoe
- TCP Reno
- TCP New Reno
- (TCP SACK)

**Components**
- Slow Start
- Additive Increase Multiplicative Decrease (AIMD)
- Fast Retransmit
- Fast Recovery
- Partial ACK
- Selective ACK

*Check out my talk from 2018 for this stuff!*
Signaling loss in the old days...

Sender

Packet[0]
Packet[1]
Packet[2]
Packet[3]
Packet[4]
Packet[5]

ACK[0]
ACK[0]
ACK[0]
ACK[0]
ACK[0]
ACK[2]

Receiver

ACK[0]
ACK[0]
ACK[3]
ACK[4]
ACK[5]

Packet[1]
Packet[2]
Packet[3]
Packet[4]
Packet[5]

3 DupACKs triggering first Fast Retransmit

Sender

ACK[3]
ACK[4]
ACK[5]

Receiver

Packet[3]
Packet[4]
Packet[5]

Timeout
TCP New Reno: Partial ACK

- Partial ACKs trigger Fast Retransmits of multiple lost segments
- Second lost segment gets retransmitted immediately after the first Partial ACK
RFC 2018

“TCP may experience poor performance when multiple packets are lost from one window of data.”

[...]

“The receiving TCP sends back SACK packets to the sender informing the sender of data that has been received. The sender can then retransmit only the missing data segments.”
TCP SACK

Sender

Receiver

Retransmission of the missing piece after 3 Dup SACKs
Demo

- Open Trace in your Wireshark:

  "sack_and_dsack.pcap"
OK, that’s the old stuff!

Let's think outside the box!
But now - Revolution!

Multipath TCP (MPTCP)

- RFC 6182 / 6824
- Latest implementations (suggested)
Motivation

Networks becoming multipath
- Link aggregation
- High availability

Redundant networks
- Mobile phones utilizing Wifi & 4G
- Multi-homed Servers
Flow distribution (1/2)

Server A to Server B

- Load balancing across links usually based on
  - MAC addresses
  - IP addresses

- Single TCP flow
Flow distribution (2/2)

Server A to Server B

- Load balancing across links usually based on
  - MAC addresses or
  - IP addresses

- 2nd TCP flow, same SRC/DST
TCP Statements

• TCP only uses a single path regardless of the network topology

• It’s always tied to a single SRC and DST address of client and server
  • If one or the other changes → Connection break!
Deployment goals of recent MPTCP implementation

• TCP needs to evolve and utilize multiple paths for the same data transport

• It MUST meet the following criteria:
  1) Support unmodified applications
  2) Work on today's networks
  3) Work whenever TCP works, or fallback to TCP
Easy peasy!

... or not?
MPTCP Session Setup

**Client**

- SYN + MP_CAPABLE + token 6767
- SYN / ACK + MP_CAPABLE + token 2525
- ACK + MP_CAPABLE + token 6767

**Server**

- Connection identifier
  - analog to ephemeral TCP ports
  - Used to associate multiple subflows
MPTCP Session Setup

Client

Subflow 1

Server

Connection identifier

• similar to ephemeral TCP ports
• Used to associate multiple subflows

SYN + MP_JOIN + token 6767

SYN / ACK + MP_JOIN + token 2525

ACK + MP_JOIN + token 6767
MPTCP Session Setup

Connection identifier
- similar to ephemeral TCP ports
- Used to associate multiple subflows
## The packet

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Header len</td>
<td>TOS</td>
<td>Total length</td>
</tr>
<tr>
<td>Identification</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>Source IP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header len</td>
<td>Reserved</td>
<td>Code bits</td>
<td>Window size</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# The packet

<table>
<thead>
<tr>
<th></th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Version</td>
<td>Header len</td>
<td>TOS</td>
<td>Total length</td>
</tr>
<tr>
<td>Identification</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>Header Checksum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source IP</td>
<td>Destination IP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ Number</td>
<td>ACK Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header len</td>
<td>Reserved</td>
<td>Code bits</td>
<td>Window size</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The packet

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Header len</td>
<td>TOS</td>
<td>Total length</td>
</tr>
<tr>
<td>Identification</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>Source IP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header len</td>
<td>Reserved</td>
<td>Code bits</td>
<td>Window size</td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
<td>Urgent pointer</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GOOD LUCK WITH THAT
Nasty middle boxes

Middle boxes can drop TCP options:

a) SYN MP_CAPABLE

Client

Server

MP status: disabled

MP status: disabled

SYN + MP_CAPABLE

SYN / ACK

ACK
Nasty middle boxes

Client

Server

Middle boxes can drop TCP options:
b) SYN / ACK MP_CAPABLE

MP status: disabled

MP status: enabled

SYN + MP_CAPABLE

SYN / ACK + MP_CAPABLE

ACK
Nasty middle boxes

Middle boxes can drop TCP options:

- c) ACK MP_CAPABLE
Nasty middle boxes

Client

- MP status: enabled

Server

- MP status: enabled

Summary

Don’t expect the path between two endpoints is always the same!

Have a fallback option available!

3 set MP options minimizes risks of middle box intervention!
• Live Demo
How to send Data?

With regular TCP
Application passes Data to MPTCP
How to send Data?

With regular TCP
Data will be send over a single connection in the order of appearance
How to send Data?

With regular TCP
ACKs are going back in the order of received data
How to send Data?

**MPTCP Strawman design**
Distribute Data evenly over all connections

No problem so far...

Client

Server
How to send Data?

MPTCP Strawman design

Oh no, a middle box!

- Path 2 did not see Data 1
- ACK 2 cumulatively acknowledges Data 1 & 2
How to send Data?

MPTCP Strawman design

Chances are middle box will:

a) Drop the ACK
b) Correct the ACK
c) RST the connection
The NOPES

- Don’t use TCP Sequence numbers across multiple flows
- Don’t expect the network to be as smart as your clients/servers
Instead

- Own sequence numbers for each subflow
  - Make MPTCP look like TCP with no gaps

- Additional data sequence numbers for MPTCP
  - Because reordering can still happen

- Additional data ACK for flow control
## MPTCP Header

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ Number</td>
<td>ACK Number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Header len</th>
<th>Reserved</th>
<th>Code bits</th>
<th>Window size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checksum</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data SEQ Number</th>
<th>Options</th>
<th>Data ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Urgent pointer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Unique to subflow**
Sending data across subflows

MPTCP data transmission
Sequence Numbers are unique per subflow

Data Sequence Numbers are shared across all subflows
How to react on packet loss?

MPTCP data transmission
Subflow 2 experiences packet loss
How to react on packet loss?

**MPTCP data transmission**
Data will be retransmitted on subflow 1

Client

![Diagram of MPTCP data transmission]

Server

SEQ 102 DATA 2  SEQ 101 DATA 3  SEQ 100 DATA 1

Subflow 1

Subflow 2

DATA
Demo

- Open Trace in your Wireshark:

  "mptcp_client01.pcap"
# MPTCP Options

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>MP_CAPABLE</td>
<td>Multipath Capable</td>
</tr>
<tr>
<td>0x1</td>
<td>MP_JOIN</td>
<td>Join Connection</td>
</tr>
<tr>
<td>0x2</td>
<td>DSS</td>
<td>Data Sequence Signal (Data ACK and data sequence mapping)</td>
</tr>
<tr>
<td>0x3</td>
<td>ADD_ADDR</td>
<td>Add Address</td>
</tr>
<tr>
<td>0x4</td>
<td>REMOVE_ADDR</td>
<td>Remove Address</td>
</tr>
<tr>
<td>0x5</td>
<td>MP_PRIO</td>
<td>Change Subflow Priority</td>
</tr>
<tr>
<td>0x6</td>
<td>MP_FAIL</td>
<td>Fallback</td>
</tr>
<tr>
<td>0x7</td>
<td>MP_FASTCLOSE</td>
<td>Fast Close</td>
</tr>
<tr>
<td>0xf</td>
<td>(PRIVATE)</td>
<td>Private Use within controlled testbeds</td>
</tr>
</tbody>
</table>
MPTCP Scheduler

Which path to use when?
Use the least congested path with the lowest RTT

However:
MPTCP throughput should never be less than best possible TCP throughput on a single path.
Main MPTCP Use Cases

- Mobile phone
  - Improved stability

- Datacenters
  - Improved throughput
Who is using it today?

- Linux kernel (MultipathTCP-Linux)
- Apple iOS and macOS
- Citrix load balancers
- FreeBSD (FreeBSD-MPTCP)
- Oracle Solaris
- ...
You might try!

- MPTCP doesn’t require the application to be adjusted
- You do not need to change your network
Want to play at home?

https://www.openmptcprouter.com/
References

- https://multipath-tcp.org/
- https://youtu.be/bwh5pr2uxgQ
- https://github.com/Neohapsis/mptcp-abuse
Thank you

Questions? Compliments? Wisdoms?

Please use the Guidebook App to provide feedback.

Contact

@SimonLindermann

sl@local-area.network