MPTCP: Design and Deployment

Day 11
Use of Multipath TCP in iOS 7

• Multipath TCP in iOS 7
  – Primary TCP connection over WiFi
  – Backup TCP connection over cellular data

• Enables fail-over

• Improves performance

• Currently only being used between phone and Siri servers
Outline

• TCP Background

• MPTCP Design

• MPTCP Deployment issues
TCP--Revisited
What is the Point of TCP?

• Effectively share the network?
  – Goal: Fairness and vague notion of equality
  – Ideal: If N connections, each should get 1/N of BW
    • So, 10MB with 2 connections, each gets 5MB

• How to determine 1/N in a distributed manner?
  – Each connection probes for available BW by sending more packets
  – When a loss occurs then sso slow down
  – If no losses send more packets
TCP Revisited

• TCP 3-way handshake
TCP Revisited

- TCP 3-way handshake

- Flow-Control/Congestion-Control
  - AIMD
TCP Revisited

• TCP 3-way handshake

• Flow-Control/Congestion-Control

• Connection Tear-down

FIN-Flag

ACK/FIN-Flag

I’m done talking

Got it, good bye
Congestion Control: AMID

- Every RTT increase your Window by 1 Pkt
  - \[ W = W + 1 \]
- However if you receive a loss
  - Cut your window by half: \[ w = w/2 \]
  
  - You aggressively decrease!!
Congestion Control

- Every RTT increase your Window by 1 Pkt
  - \( W = W + 1 \)
- However if you receive a loss
  - Cut your window by half: \( w = \frac{w}{2} \)

  - You aggressively decrease!!
Congestion Control

- Every RTT increase your Window by 1 Pkt
  - $W = W + 1$
- However if you receive a loss
  - Cut your window by half: $w = w/2$
  - You aggressively decrease!!
TCP Congestion Control: AIMD (Additive Increase Multiplicative Decrease)

• Every RTT increase your Window by 1 Pkt
  – \( W = W + 1 \)

• However if you receive a loss
  – Cut your window by half: \( w = w/2 \)
  – You aggressively decrease!!

\[ \text{RENO Probing} \]
Side-effects

• TCP is fundamentally unfair
  – Every things happens after an RTT
  – Flows with smaller RTT outperform flows with larger RTT

• TCP is slow
  – It increase one packet at a time
  – Takes a while to reach capacity for links with large BW

• TCP is inefficient
  – Devices have multiple paths (Data center, mobile)
  – TCP only uses only path
Outline

- TCP Background
- MPTCP Design
- MPTCP Deployment issues
MPTCP

• Takes advantage of existence of multiple paths
  – Examples:
    • Data center network: as many as 4 paths exists
    • Mobile networks: as many as 2 paths (3G, Wifi)
MPTCP- Constraints

– Must be Fair to TCP: Must perform as well as a connection with same loss rate
  • Increase the window of subflow on same path by 1/N, where N is number of subflows on a path.

– Perform at-least as well as TCP

– Must send traffic on the efficient path
  • Send traffic on path proportion to the loss rate of path
  • Send probe paths on poor paths to detect change in conditions

– Resilient to sub-path failure; If at-least one path exists then connection must persist
  • The above solutions helps out here
Implementing MPTCP

• How do sender/receiver agree to use MPTCP?

• How do you start new subflows?

• How do you ensure flow-control and reliable delivery?
  – Sequence numbers and ACKs

• How do you end the connection?
How do you Extend TCP?

• TCP header
  – Ten mandatory fields
  – Optional extension field (usually during handshake)
<table>
<thead>
<tr>
<th>Bit 0</th>
<th>Bit 15</th>
<th>Bit 16</th>
<th>Bit 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>IHL</td>
<td>TOS</td>
<td>ECN</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>Destination IP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence Number</td>
<td>Acknowledgment Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header Length</td>
<td>Reserved</td>
<td>Code bits</td>
<td>Receive Window</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>
How do you Extend TCP?

• TCP header
  – Ten mandatory fields
  – Optional extension field (usually during handshake)

• How should Options be treated?
  – If Unknown options
    • Ignored by receiving Host/Routers
    • At Router packets should not be changed
Negotiating MPTCP

• Ideal: Use TCP Options

• Problem: Network Middleboxes
  – MB: strip all TCP options from 10% of paths
  – MB: strip TCP options on SYN/ACK for 5% of paths

• Solution:
  – Fall back to normal TCP if options are stripped
  – If options are only stripped on SYN/ACK the problematic:
    • Receiver thinks MPTCP
    • Sender think fallback to TCP
  • Receiver should keep sending MPTCP options on subsequent ACKs
TCP Revisited

• TCP 3-way handshake

Hello: I use MPTCP

Hello: I too use MPTCP

Let’s talk: in MPTCP
Negotiating MPTCP

- Ideal: Use TCP Options

- Problem: Network Middleboxes
  - MB: strip all TCP options from 10% of paths
  - MB: strip TCP options on SYN/ACK for 5% of paths

- Solution:
  - Fall back to normal TCP if options are stripped
  - If options are only stripped on SYN/ACK the problematic:
    - Receiver thinks MPTCP
    - Sender think fallback to TCP
    - Receiver should keep sending MPTCP options on subsequent ACKs
TCP Revisited

• TCP 3-way handshake

Hello:
I use MPTCP
TCP Revisited

- TCP 3-way handshake
TCP Revisited

• TCP 3-way handshake
TCP Revisited

• TCP 3-way handshake
TCP Revisited

• TCP 3-way handshake
Negotiating MPTCP

- Ideal: Use TCP Options

- Problem: Network Middleboxes
  - MB: strip all TCP options from 10% of paths
  - MB: strip TCP options on SYN/ACK for 5% of paths

- Solution:
  - Fall back to normal TCP if options are stripped
  - If options are only stripped on SYN/ACK the problematic:
    - Receiver thinks MPTCP
    - Sender think fallback to TCP
    - Receiver should keep sending MPTCP options on subsequent ACKs
TCP Revisited

• TCP 3-way handshake
TCP Revisited

- TCP 3-way handshake

Hello: I Use MPTCP
TCP Revisited

• TCP 3-way handshake
TCP Revisited

• TCP 3-way handshake

Let’s talk

Hello:

SYN Flag
SYN/ACK Flag
ACK Flag

I’m expecting MPTCP… What Happened???
Negotiating MPTCP

• Ideal: Use TCP Options

• Problem: Network Middleboxes
  – MB: strip all TCP options from 10% of paths
  – MB: strip TCP options on SYN/ACK for 5% of paths

• Solution:
  – Fall back to normal TCP if options are stripped
  – If options are only stripped on SYN/ACK the problematic:
    • Receiver thinks MPTCP
    • Sender think fallback to TCP
    • Receiver should keep sending MPTCP options on subsequent ACKs
Implementing MPTCP

• How do sender/receiver agree to use MPTCP?

• How do you start new subflows?

• How do you ensure flow-control and reliable delivery?
  – Sequence numbers and ACKs

• How do you end the connection?
Starting New Subflows

- **Ideal:**
  - Open new connection and send packets
  - All you need is a different IP address
  - Use same sequence numbers for all connections → allow MPTCP to reuse TCP’s congestion control
How do you Start New Subflows?

• Ideal:
  – Open new connection and send packets
  – All you need is a different IP address
  – Use same sequence numbers for all connections → allow MPTCP to reuse TCP’s congestion control

• Implication:
  – New connection has no TCP-Handshake
  – Each connection will have gap in sequence numbers
Middleboxes Attack Again!

- Problem: Middleboxes
  - NAT/FW: If no TCP-Handshake, connection is dropped

Subflow 1

Subflow 2

If no handshake. Then drop
Middleboxes Attack Again!

- Problem: Middleboxes
  - NAT/FW: If no TCP-Handshake, connection is dropped

Each subflow starts with TCP-Handshake

If no handshake. Then drop

Change IP address
Middleboxes Attack Again!

- Problem: Middleboxes
  - NAT/FW: If no TCP-Handshake, connection is dropped
  - Other MB: if gap in sequence number connection is dropped
  - Other MB: rewrite beginning seq # to add more randomness

Each subflow starts with TCP-Handshake

If no handshake. Then drop

Change IP address
Middleboxes Attack Again!

- Problem: Middleboxes
  - NAT/FW: If no TCP-Handshake, connection is dropped
  - Other MB: if gap in sequence number, connection is dropped
  - Other MB: rewrite beginning seq # to add more randomness

Each subflow starts with TCP-Handshake
Handshake carries two keys:
Key-1: allows the receiver to map sub-flow to original connections

If no handshake. Then drop

Subflow 1
NAT
Subflow 2
Change IP address
Middleboxes Attack Again!

• Problem: Middleboxes
  – NAT/FW: If no TCP-Handshake, connection is dropped
  – Other MB: if gap in sequence number connection is dropped
  – Other MB: rewrite beginning seq # to add more randomness

I’m just evil!
A Regular TCP Stream

The same stream split across subFlows

I’m just evil!
Middleboxes Attack Again!

- Problem: Middleboxes
  - NAT/FW: If no TCP-Handshake, connection is dropped
  - Other MB: If gap in sequence number, connection is dropped
  - Other MB: Rewrite beginning seq # to add more randomness

Each subflow has unique seq #

Subflow 1
Subflow 2

I’m just evil!

Change IP address
<table>
<thead>
<tr>
<th>Bit 0</th>
<th>Bit 15</th>
<th>Bit 16</th>
<th>Bit 31</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subflow</strong> Source Port</td>
<td><strong>Subflow</strong> Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Subflow</strong> Sequence Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Subflow</strong> Acknowledgment Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Header Length</strong></td>
<td>Reserved</td>
<td>Code bits</td>
<td>Receive Window</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>
The same stream split across subFlows

I’m just evil!
But We still Want to use TCP’s Connection Control

• At the receiver need a way to convert from
  – From: Subflow sequence #
  – Into: flow sequence #

A Regular TCP Stream

I’m just evil!
But We still Want to use TCP’s Connection Control

• At the receiver need a way to convert from
  – **From**: Subflow sequence #
    - Each subflow has unique seq #
    - But also carries, as TCP Options, offset to original seq #
    - So Seq # can be contiguous

A Regular TCP Stream

4 3 2 1
Actual Attacks Can Use the Random Sequence

• There are Real bad people in the network!!
  – Start new subflows with random sequence #

• Need a mechanism to validate new subflows
  – Add a random key during start-up
  – Each subflow uses this a token created form this key
Middleboxes Attack Again!

• There are Real bad people in the network!!
  – Start new subflows with random sequence #
  – Need a mechanism to validate new subflows
    – Add a random key during start-up
    – Each subflow uses this key

Each subflow starts with TCP-Handshake

  Handshake carries two keys:
  Key-1: allows the receiver to map sub-flow to original connections
  Key-2: prevents hackers from hijacking the connection
Problems with Starting New Subflows?

• **Ideal:**
  – Open new connection and send packets
  – All you need is a different IP address
  – Use same sequence numbers for all connections \(\rightarrow\) **allow MPTCP to reuse TCP’s congestion control**

• **Implication:**
  – New connection has no TCP-Handshake
  – Each connection will have gap in sequence numbers

• **Problem: Middleboxes**
  – **NAT/FW:** If no TCP-Handshake, connection is dropped
  – **Other MB:** if gap in sequence number connection is dropped
  – **Other MB:** rewrite beginning seq # to add more random-ness
  – **Attackers:** can new malicious new subflows
MPTCP’s Solutions

• Each subflow starts with TCP-Handshake
  – Handshake carries two keys:
    • Key-1: allows the receiver to map sub-flow to original connections
    • Key-2: prevents hackers from hijacking the connection

• Each subflow has unique seq #
  – But also carries, as TCP Options, offset to original seq #
  – So Seq # can be contiguous

• If MB rewrites seq #
  – The offset helps to map back to original TCP
Implementing MPTCP

• How do sender/receiver agree to use MPTCP?

• How do you start new subflows?

• How do you ensure flow-control and reliable delivery?
  – Sequence numbers and ACKs

• How do you end the connection?
Ending a Flow

- Traditionally Two ways to end a connection:
  - RST Flag: End connection because of error
  - FIN Flag: End connection → all packets send
Ending a Flow

• Traditionally Two ways to end a connection:
  – RST Flag: End connection because of error
  – FIN Flag: End connection → all packets send

• RST on a subflow → subflow error
  – Other subflows continue

• FIN on subflow → confusing!!!
  – Is flow finished? Or just subflow?
  – Need to be careful with FIN, if FIN is sent but subflow packets are lost and need to be resent then the connection is in trouble
Ending a Flow

• Each subflow
  – Sends FIN as a TCP option

• After all subflow send FIN
  – Then send FIN as TCP Flag
Outline

• TCP Background

• MPTCP Design

• MPTCP Deployment issues
Keeping the Same Socket API

• Backwards compatibility with existing apps
  – Present the same socket API and expectations

• Establish the TCP connection in the same way
  – Create a socket to a single remote IP address/port
  – … and then add more subflows to the connection

• Work in all scenarios where regular TCP works
  – If a subflow fails, the connection should continue
  – … as long as some other subflow has connectivity
MPTCP in the Network Stack

From http://queue.acm.org/detail.cfm?id=2591369
Receive Buffer Space

• Each TCP connection has a receive buffer
  – Buffer space to store incoming data
  – … until it is read by the application

• TCP flow control
  – Receiver advertises the available buffer space
  – … using the “receive window”

• Should each subflow have its own receive window?
  – Starvation of some subflows in a connection?
  – Fairness relative to other TCP connections?
  – Fragmentation of the available buffer space?

• Instead, use a common receive window
Insights

• Drawbacks of Normal TCP (TCP Reno)
  – Slow to determine N/W capacity
  – Unfair on the level of RTT
    • Ties congestion control to loss rate rather than RTT
  – Doesn’t take advantage of multiple paths
    • Create Subflow for multiple paths

• Middleboxes make it hard to modify TCP
  – Lots of built in assumptions about TCP