CS 204: Multipath TCP

Jiasi Chen
Lectures: MWF 12:10-1pm in WCH 139

http://www.cs.ucr.edu/~jiasi/teaching/cs204_spring16/
Goals

• Use the available network paths at least as well as regular TCP, but without starving TCP.
• Usable as regular TCP for existing applications.
• Enabling MPTCP must not prevent connectivity on a path where regular TCP works.
Network Stack

Source: http://queue.acm.org/detail.cfm?id=2591369
Network Address Translators

Source: https://en.wikibooks.org/wiki/Communication_Networks/NAT_and_PAT_Protocols
Connection Setup

• Use MP-CAPABLE flag to indicate sender has MPTCP capability

• **Problem:** Middleboxes remove TCP options
  • Option removed on msg 1?
  • Option removed on msg 2?
Connection Setup

- Use MP-CAPABLE flag to indicate sender has MPTCP capability

- **Problem:** Middleboxes remove TCP options

  - Option removed on msg 1? → **fall back to TCP**

  - Option removed on msg 2? → host A and host B’s views are inconsistent → **add another MPT-CAPABLE to msg 3 if MP-CAPABLE recv’d in msg 2**
Adding New Flows: Naïve solution

- Host A has addresses A1 and A2
- Assume Host B knows these addresses and starts sending data to both

- **Problem**: Middleboxes will not allow data to be sent without SYN → need 3-way handshake for new subflows
Adding New Flows: Identification

• TCP flows traditionally identified by <source IP, source port, dest IP, dest port>

• Problem: when adding new subflow to existing connection, don’t know the source IP
Adding New Flows: Identification

- TCP flows traditionally identified by <source IP, source port, dest IP, dest port>

- Problem: when adding new subflow to existing connection, don’t know the source IP → add a token to identify the connection
  - token = hash(key)
Adding New Flows: Authentication

• **Problem**: attacker could use the same token

→ **authentication using HMAC**
Hash-based Message Authentication Code (HMAC)

Adding New Flows: Authentication

• **Problem:** attacker could use the same token → authentication using HMAC
  • HMAC = f(key, rand)
  • Attacker gets one change to guess the HMAC, otherwise rand changes
Adding New Flows: Addresses

- Implicit

- Explicit
  - **Problem:** second subflow can’t reach client because of NAT
  - Server sends ADD_ADDR option
Sequence Numbers

• Naïve: Use one sequence of numbers, send subset those numbers on each subflow

• Problem: middleboxes re-initialize sequence numbers
• Problem: middleboxes don’t like gaps in sequence numbers

→ use flow-level sequence numbers along with per-subflow sequence numbers
Sequence Numbers: ACKs

- Flow-level sequence numbers needed
- Are flow-level ACKs needed? Can we infer them from subflow ACKs?
- Example: receive buffer size 2

(a) Drops due to incorrect inference

(b) Stalls due to incorrect inference

Source: [3]
Sequence Numbers: Mapping

• Mapping from subflow sequence number to data sequence number

• Naïve: On each packet, record absolute value of data sequence number

• TCP segmentation offload (TSO)
  • Divide large segments into smaller chunks
  • Performed by NICs to save CPU

• Problem: TSO copies same data sequence number onto multiple packets

→ record exact mapping between subflow and data sequence numbers
Sequence Numbers: Encoding

- **Option 1**: Encode in data payload
- **Problem**: Data ACKs can get stuck from flow control

Encode data sequence numbers and ACKs in TCP options
Flow Control

- Naïve: Use one receive window for each flow
  → one receive window for each subflow

- Problem: Subflow failure can lead to deadlock
  1. Application waiting for subflow 1’s data
  2. Subflow 1 fails, doesn’t send data
  3. No space left in subflow 2’s rwnd to transmit new data

→ One receive window for the overall flow
Retransmissions

• What if data on a subflow times out?
  • Can resend on a different subflow

• Still need to retransmit on the original subflow
  • No holes in subflow sequence numbers for middlebox compatibility
  • Wastes bandwidth

• Protocol not defined by RFC
  • Aggressive: Re-transmit every packet not received on a different subflow
  • Conservative: Re-transmit after fixed number of retries on the original subflow
Congestion Control

• Naïve: use TCP congestion control separately on each path
• Problem: Not TCP-friendly

For example:
2 clients
Client A has 2 MPTCP subflows
Client B is regular TCP

Client A will receive 2/3 of capacity

Source: [2]
Congestion Control

• **Solution**: Congestion control coupled across subflows
  • Many algorithms developed
Scheduling

• When there is space in both congestion windows, which subflow to transmit on?
  • Round-robin
  • Lowest-RTT first

• ACK-clocked
  • Round-robin: if cwnd has space, send even if out of RR order?
  • Lowest-RTT first: if cwnd has space, send on higher-RTT subflow?
Practical Example

Source: [2]
Who Uses MPTCP?

- **iOS 7 for Siri**
  - Primary TCP connection over WiFi
  - Backup TCP connection over cellular data

- **Use cases**
  - Smartphones with 4G and WiFi for connectivity
  - Data center servers with multiple high-speed links for load balancing

- **Linux kernel available**
Paper Discussion

• How computationally expensive is it?

• Is TCP-friendliness too restrictive?
Sources

2. TCP Extensions for Multipath Operation with Multiple Addresses, RFC 2684.