STMS: Improving MPTCP Throughput Under Heterogenous Networks

Hang Shi\textsuperscript{1}, Yong Cui\textsuperscript{1}, Xin Wang\textsuperscript{2}, Yuming Hu\textsuperscript{1}, Minglong Dai\textsuperscript{1}, Fanzhao Wang\textsuperscript{3}, Kai Zheng\textsuperscript{3}

\textsuperscript{1} Tsinghua University, \textsuperscript{2}Stony Brook University, \textsuperscript{3}Huawei Technologies
Background

• Multipath TCP is widely adopted to aggregate bandwidth of multiple interfaces of mobile devices
• Transparent to both application and middlebox
• However, mobile WiFi and LTE are heterogeneous:
  • 20% of top 500 sites has RTT difference > 45ms, as high as 134ms

1Mobicom 16, Understand Multipath performance on Mobile devices
Big host buffer requirement

- Default scheduler: send packets through fastest available path

T = 1s

- Fast path: RTT = 0s, bandwidth = 1 packet/s
- Slow path: RTT = 4s, bandwidth = 1 packet/s
Big host buffer requirement

- Default scheduler: send packets through fastest available path

```
1 -> S
2 -> F
3 -> S
4 -> F
```

- Sender
- Fast path: RTT = 0s, bandwidth = 1 packet/s
- Slow path: RTT = 4s, bandwidth = 1 packet/s
- Receiver
- $T = 2s$

send_una

sent, unacked

sent, acked

received

unsent/received
Big host buffer requirement

- Default scheduler: send packets through fastest available path
- Packet sent from slow path arrive late. Can not submit to application. Need more buffer.

**Sender**

1 -> S
2 -> F
3 -> S
4 -> F
5 -> S
6 -> F

**Receiver**

\( S \)
\( F \)
\( F \)
\( F \)

Fast path: RTT = 0s, bandwidth = 1 packet/s

Slow path: RTT = 4s, bandwidth = 1 packet/s

**Diagram:**

- Send unacked
- Sent, unacked
- Sent, acked
- Received
- Unsent/received

**Time:** \( T = 3s \)
Host buffer is not the only bottleneck

- TC running in OpenWrt router to regulate bandwidth and RTT
- iPerf to measure the throughput (send packets continuously)
- Bandwidth = 30Mbps, loss rate = 0.01%
- Host buffer big enough (6M)

<table>
<thead>
<tr>
<th>RTT</th>
<th>20ms vs 200ms</th>
<th>20ms vs 20ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregated Throughput (Mbps)</td>
<td>33.1</td>
<td>56.5</td>
</tr>
<tr>
<td>Fast path Throughput (Mbps)</td>
<td>12.1</td>
<td>28.3</td>
</tr>
<tr>
<td>Fast path Loss rate (%)</td>
<td>0.05</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Burst sending pattern

- Fast path sends packets in burst.

![Graph showing burst sending pattern](image)

20ms vs 200ms
Big in-network buffer requirement

- Bigger in-network buffer is needed to tolerant the burst.
- When in-network buffer is limited, MPTCP can not compete against single path TCP. (More packet loss)

<table>
<thead>
<tr>
<th>In-network buffer/K</th>
<th>MPTCP Fast path /Mbps</th>
<th>MPTCP overall TP /Mbps</th>
<th>SPTCP fast/Mbps</th>
<th>Utilization of fast path</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>12.1</td>
<td>31</td>
<td>28.4</td>
<td>42.61%</td>
</tr>
<tr>
<td>60</td>
<td>22</td>
<td>36</td>
<td>28.4</td>
<td>77.46%</td>
</tr>
<tr>
<td>90</td>
<td>24.9</td>
<td>40.2</td>
<td>28.4</td>
<td>87.68%</td>
</tr>
<tr>
<td>150</td>
<td>28.3</td>
<td>46.3</td>
<td>28.4</td>
<td>99.65%</td>
</tr>
</tbody>
</table>
MPTCP 2 level sequence number

- Separate send window of MP level and subflow level
- 2 level sequence number and cumulative ACK.

Sender-MP

- (1, 101) -> S
- (2, 201) -> F
- (3, 202) -> F

Sender-FP

- (2, 201) -> F
- (3, 202) -> F

Receiver

- Fast path: RTT = 0s, bandwidth = 2 packet/s
- Slow path: RTT = 4s, bandwidth = 1 packet/s
- data: 1 subflow: 101
- data: 2 subflow: 201

- sent, unacked
- sent, acked
- received
- unsent/received
Burst sending of fast path

- When ACK of slow path returns, MP-level send window slides, fill the CWND space of fast path.
MPTCP-level window sliding

- the left edge of MP send window almost only slides after receiving ACK from slow path.
CWND free space of fast path

- Break the ACK clocking of single TCP.
Solution space

- Retransmission and penalization\(^1\) can alleviate host buffer problem. Can not solve in-network buffer problem.
- Pacing can solve in-network buffer problem.
  - TC pacing, need set the pacing rate manually
  - BBR congestion control, not fair with single path TCP
- Our solution: Dynamically out-of-order sending for in-order arrival
  - Solve both host buffer and in-network buffer.
  - Congestion control agnostic.

\(^1\) NSDI 12: How hard can it be? designing and implementing a deployable multipath tcp
Out-of-order sending

Sender

send_una

1 -> F

T = 1s

4 -> S

Fast path: RTT = 0s, bandwidth = 1 packet/s

Slow path: RTT = 4s, bandwidth = 1 packet/s

Receiver

F

sent, unacked

sent, acked

received

unsent/received
Out-of-order sending

Sender

1 -> F
2 -> F
4 -> S
6 -> S

Fast path: RTT = 0s, bandwidth = 1 packet/s

Slow path: RTT = 4s, bandwidth = 1 packet/s

Receiver

T = 2s

sent, unacked
sent, acked
received
unsent/received
Out-of-order sending

Sender
1 -> F
2 -> F
3 -> F
4 -> S
6 -> S
8 -> S

t = 3s

Fast path: RTT = 0s, bandwidth = 1 packet/s

Slow path: RTT = 4s, bandwidth = 1 packet/s

Receiver

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>F</th>
<th>S</th>
</tr>
</thead>
</table>

sent, unacked
sent, acked
received
unsent/received
Out-of-order sending

Sender

1 -> F
2 -> F
3 -> F
4 -> S
5 -> F
6 -> S
8 -> S

T = 4s

Fast path: RTT = 0s, bandwidth = 1 packet/s

Slow path: RTT = 4s, bandwidth = 1 packet/s

Receive

F
F
F
S
F
S

sent, unacked
sent, acked
received
unsent/received
Out-of-order sending

Sender

1 -> F
2 -> F
3 -> F
4 -> S
5 -> F
6 -> S
7 -> F
8 -> S

send_una

Receiver

F
F
S
F
S
F
S

T = 4s

Fast path: RTT = 0s, bandwidth = 1 packet/s

Slow path: RTT = 4s, bandwidth = 1 packet/s

sent, unacked
sent, acked
received
unsent/received
Out-of-order sending algorithm

- For fast path, send unsent[0]
- For slow path, send unsent[Gap]
- Leave Gap packets for fast path to send
- Out-of-order sending $\rightarrow$ in-order arrival

\[
\frac{\text{Gap}}{\text{Bandwidth}(\text{fast})} + \text{Delay}(\text{fast}) = \text{Delay}(\text{slow})
\]
Need more send buffer?

- Seems like moving Gap from receiver to sender?
- However, send window can slide faster. No duplicate ACK. Each ACK can acknowledge some packets.
- Actually, out-of-order sending can always get optimal throughput across all range of host buffer sizes.
How to get GAP value

• Naive way: Calculate from path condition measurement.
  • $Gap = Bandwidth(fast) \times (Delay(slow) - Delay(fast))$
  • Hard to measure. Need symmetric forward delay.

• **Our approach: Feedback based adjustment.**
  • No more options. Compatible with existing MPTCP protocol. Get feedback from existing options.
  • Deployable. Modify sender side only
Key insight

- Out-of-order arrival generate burst MP-level ack
- Gap = Number of bursting MP-level ACKs
Key insight

- Out-of-order arrival generate burst MP-level ack
- Gap = Number of burst MP-level ACKs
Gap adjustment

- Burst MP-level ACK(data ACK)
  - Packet[send_una] sent from slow path, Gap += delta[data_ack] - 2
  - Packet[send_una] sent from fast path, Gap -= delta[data_ack] – 2

- Limit the frequency of adjustment to avoid repeated adjustment.

- EWMA of delta over adjustment interval.
Implementation and Evaluation

• Based on Linux kernel MPTCP v0.92
• 2 variants: gap-calculation and gap-adjustment.
• Compared with Default and ECF (Early completion first. Sending tail packets out-of-orderly.)
• Controlled lab and real-world.
• Varying static and dynamic network environment.
• Varying in-network buffer and host buffer.
Microbenchmarks

- Reduce out-of-order latency: \( t(\text{submitted}) - t(\text{arrival}) \)
Microbenchmarks

- Varying receive buffer and send buffer size.
Reduce burst on the fast path

- CWND freespace when receiving ACK.
- iPerf will fill the freespace.
- Big freespace -> burst sending -> big in-network buffer requirement.
Gap adjustment is dynamic

- Change the network condition suddenly.
Macrobenchmarks

- 25% improvement when in-network buffer is limited.
Host buffer

- 20% improvement when receive/send buffer is limited.
Dynamic network condition

- Change bandwidth (left) and latency (right) randomly
Real-world evaluation

- Lab to Alibaba Cloud.
- No bandwidth regulation.
- Varying latency.
- Download 200MB file.

<table>
<thead>
<tr>
<th></th>
<th>BD(Mbps)</th>
<th>Latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFi</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>LTE</td>
<td>30</td>
<td>70</td>
</tr>
</tbody>
</table>
Conclusion

• Discover the in-network buffer problem of MPTCP.

• Leverage data ACK and subflow ACK for dynamically Out-of-order sending.

• Improve the throughput of MPTCP when RTTs are asymmetric and especially when the buffer is limited.
Thanks