STMS: Improving MPTCP Throughput Under Heterogenous Networks

Hang Shi[1], Yong Cui[1], Xin Wang[2], **Yuming Hu[1]**, Minglong Dai[1], Fanzhao Wang[3], Kai Zheng[3]

[1] Tsinghua University, [2]Stony Brook University, [3]Huawei Technologies

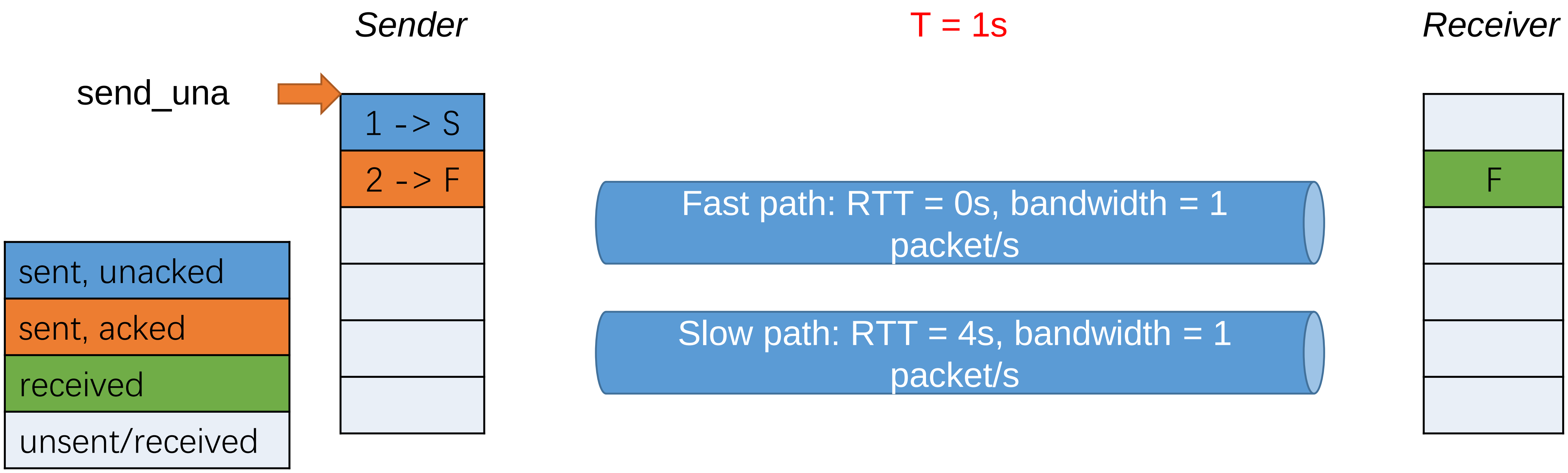
Background

- Mutipath 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[1]

[1]Mobicom 16, Understand Multipath performance on Mobile devices

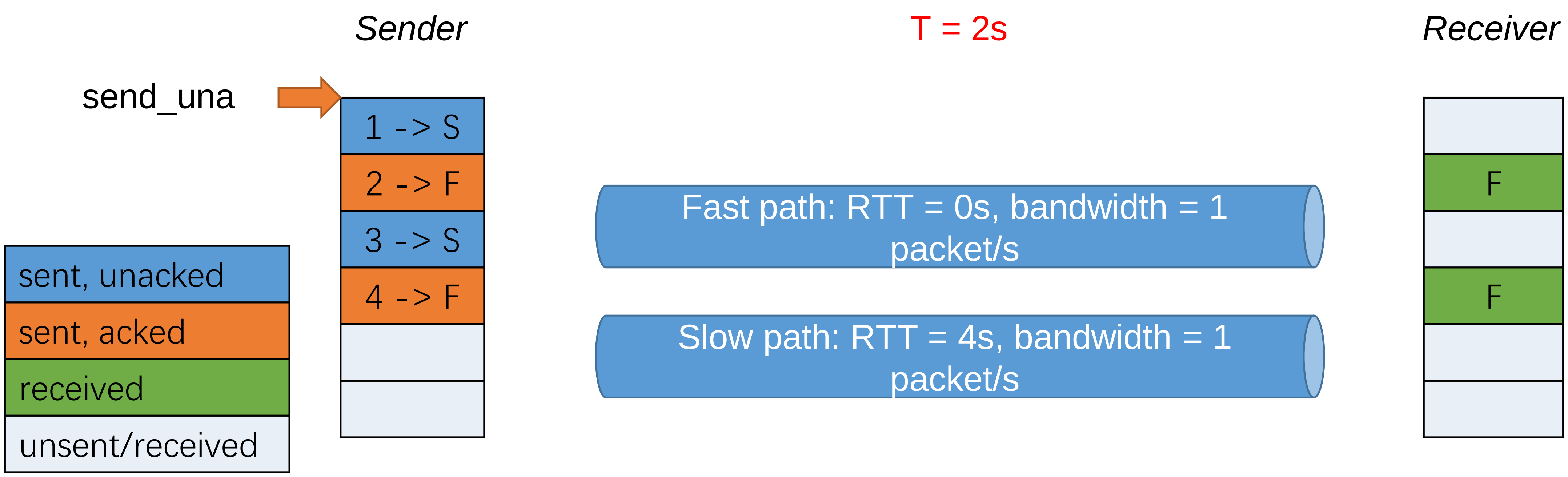
Big host buffer requirement

- Default scheduler: send packets through fastest available path



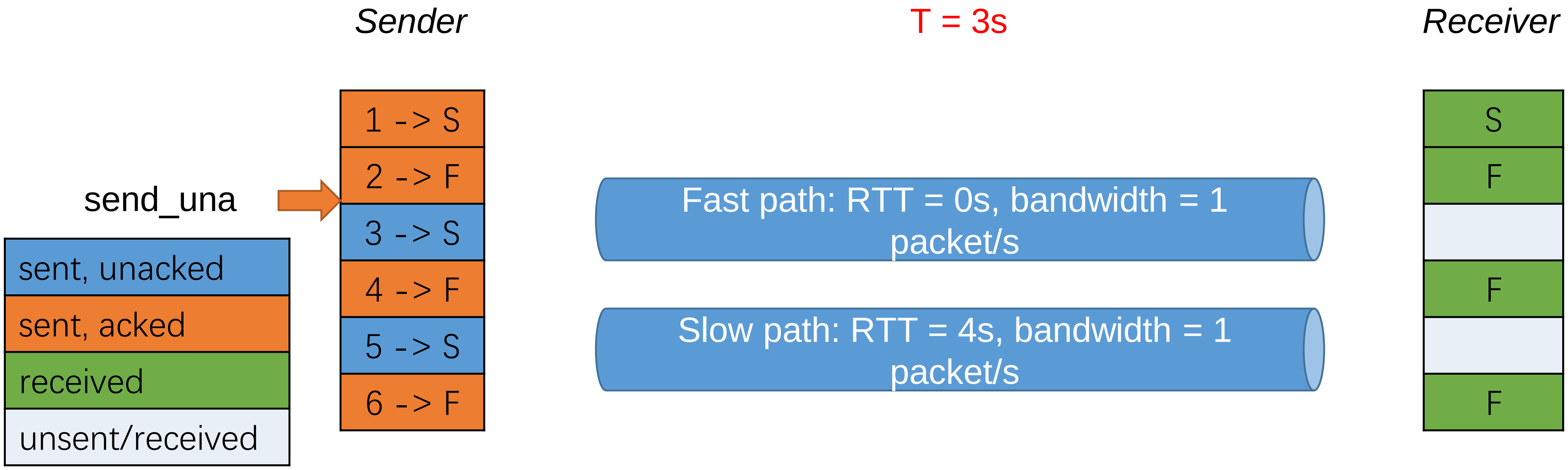
Big host buffer requirement

- Default scheduler: send packets through fastest available path



Big host buffer requirement

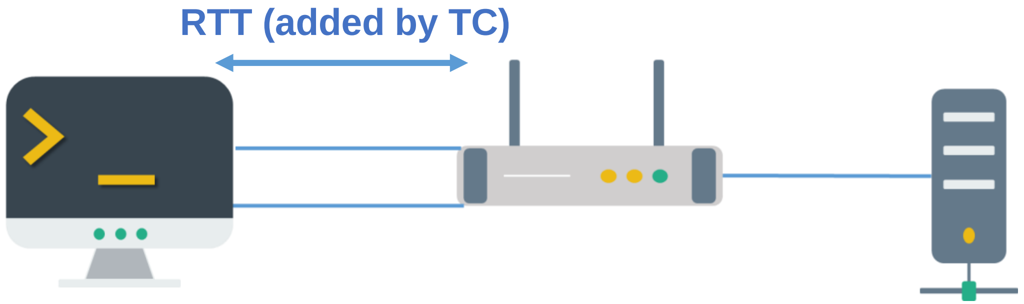
- Default scheduler: send packets through fastest available path
- Packet sent from slow path arrive late. Can not submit to application. Need more buffer.



Host buffer is not the only bottleneck

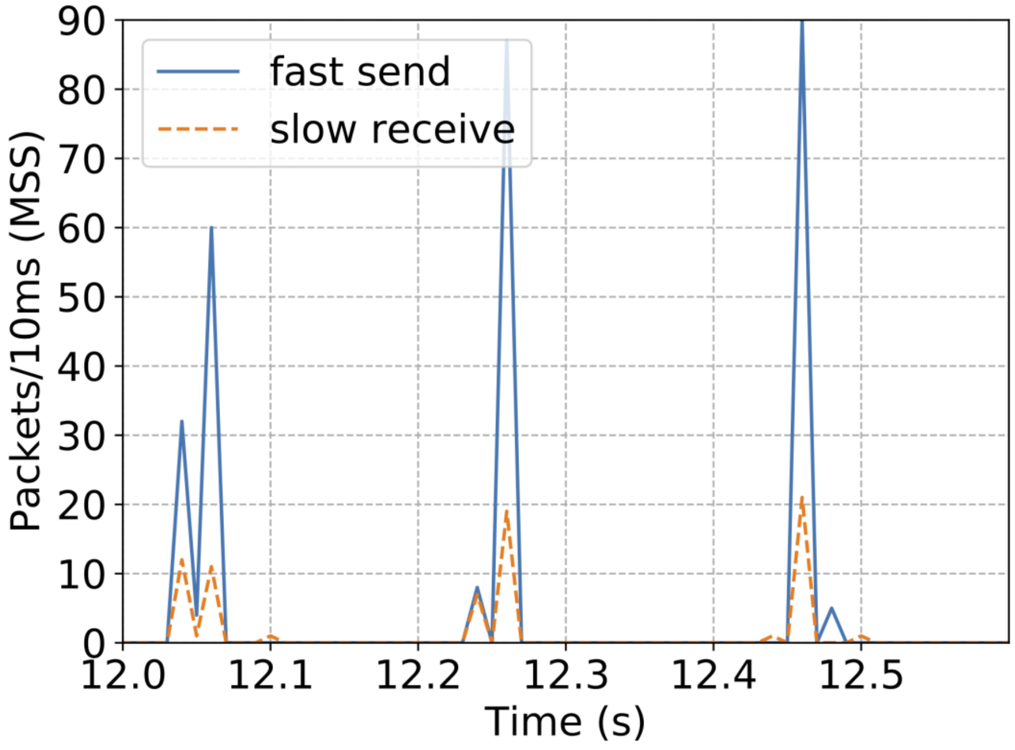
- TC running in OpenWrt router to regulate bandwidth and RTT
- iPerf to measure the throughput (send packets continuously)
- Bandwidth = 30Mbps, loss rate = 0.01%
- Host buffer big enough(6M)

RTT	20ms vs 200ms	20ms vs 20ms
Aggregated Throughput (Mbps)	33.1	56.5
Fast path Throughput (Mbps)	12.1	28.3
Fast path Loss rate (%)	0.05	0.01

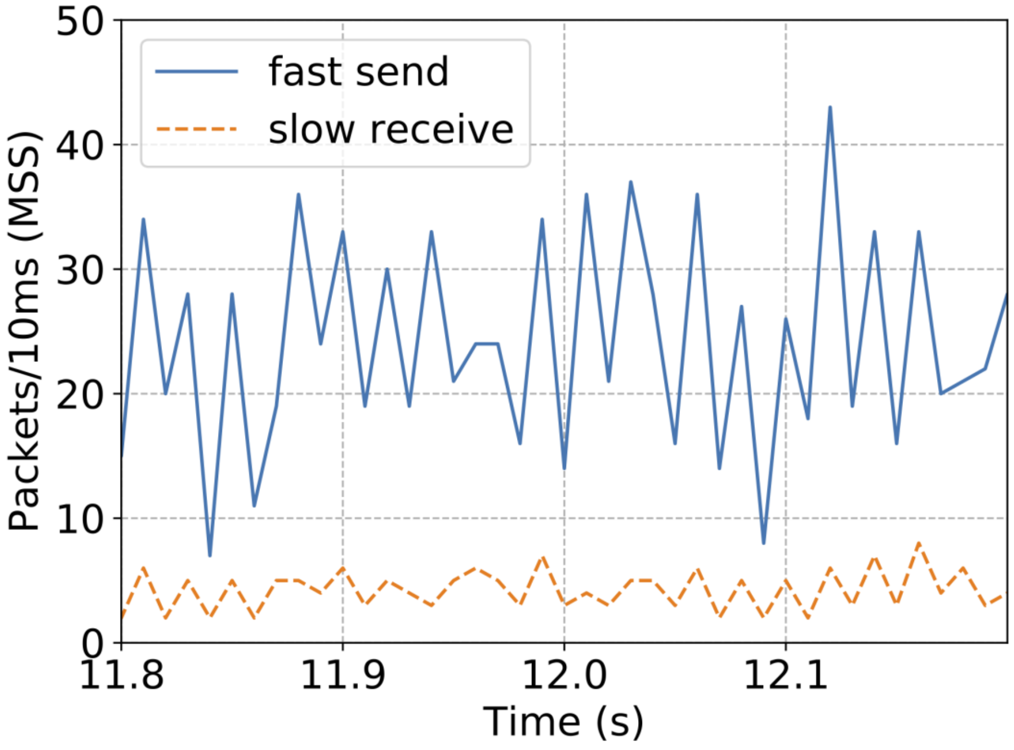


Burst sending pattern

- Fast path sends packets in burst.



20ms vs 200ms

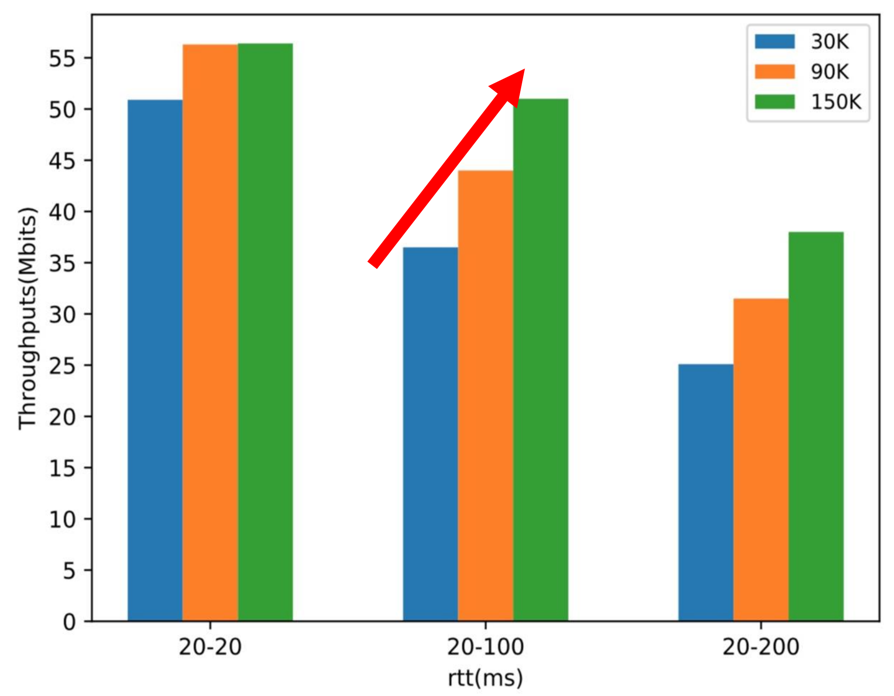


20ms vs 20ms

Big in-network buffer requirement

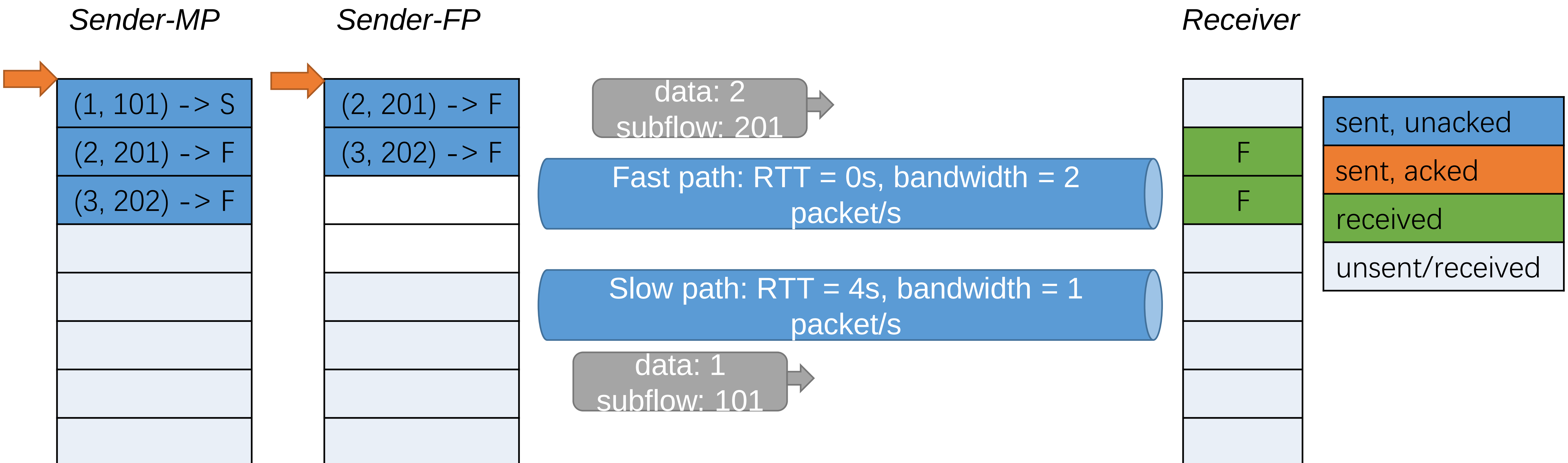
- Bigger in-network buffer is needed to tolerant the burst.
- When in-network buffer is limited, MPTCP can not compete against single path TCP. (More packet loss)

In-network buffer/K	MPTCP Fast path /Mbps	MPTCP overall TP /Mbps	SPTCP fast/Mbps	Utilization of fast path
30	12.1	31	28.4	42.61%
60	22	36	28.4	77.46%
90	24.9	40.2	28.4	87.68%
150	28.3	46.3	28.4	99.65%



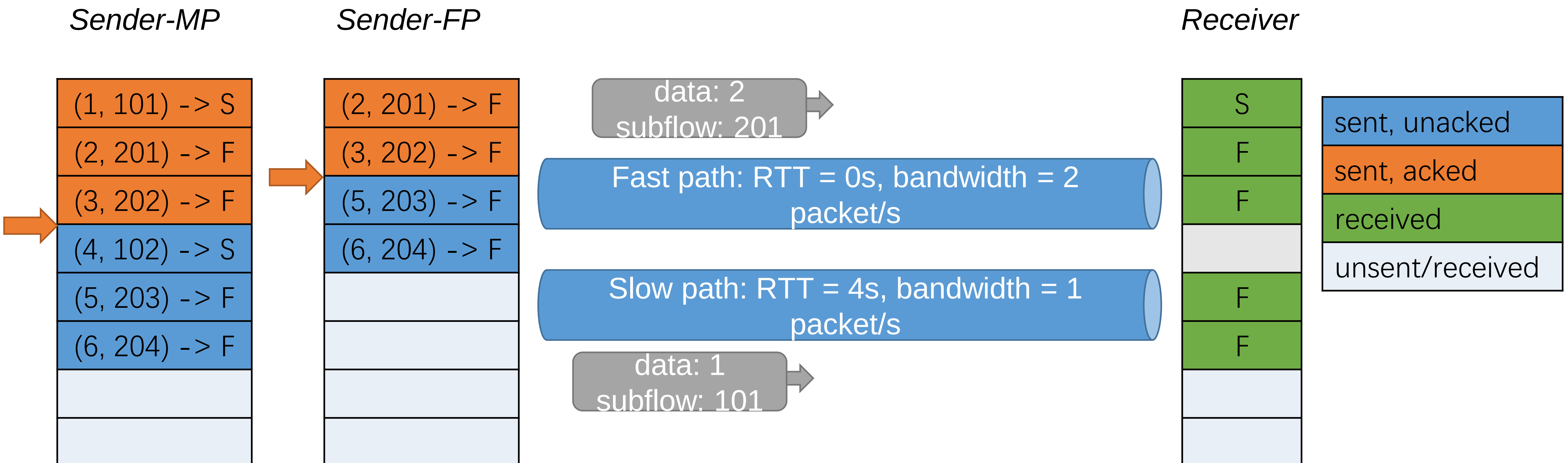
MPTCP 2 level sequence number

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



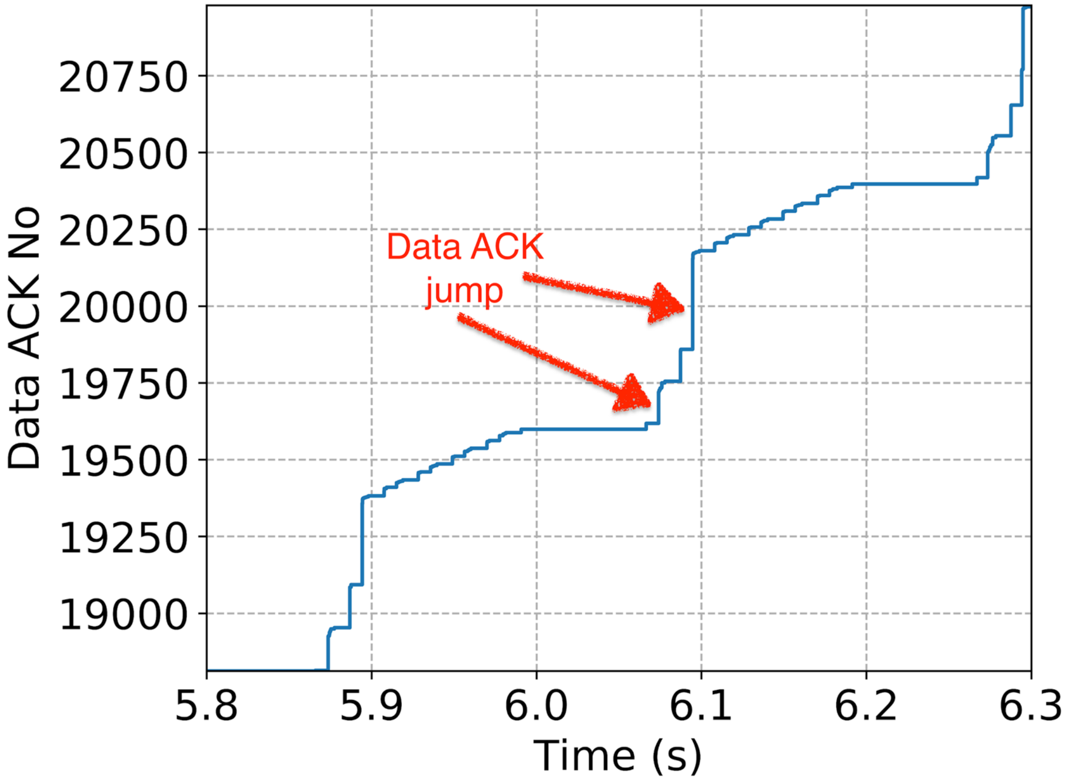
Burst sending of fast path

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

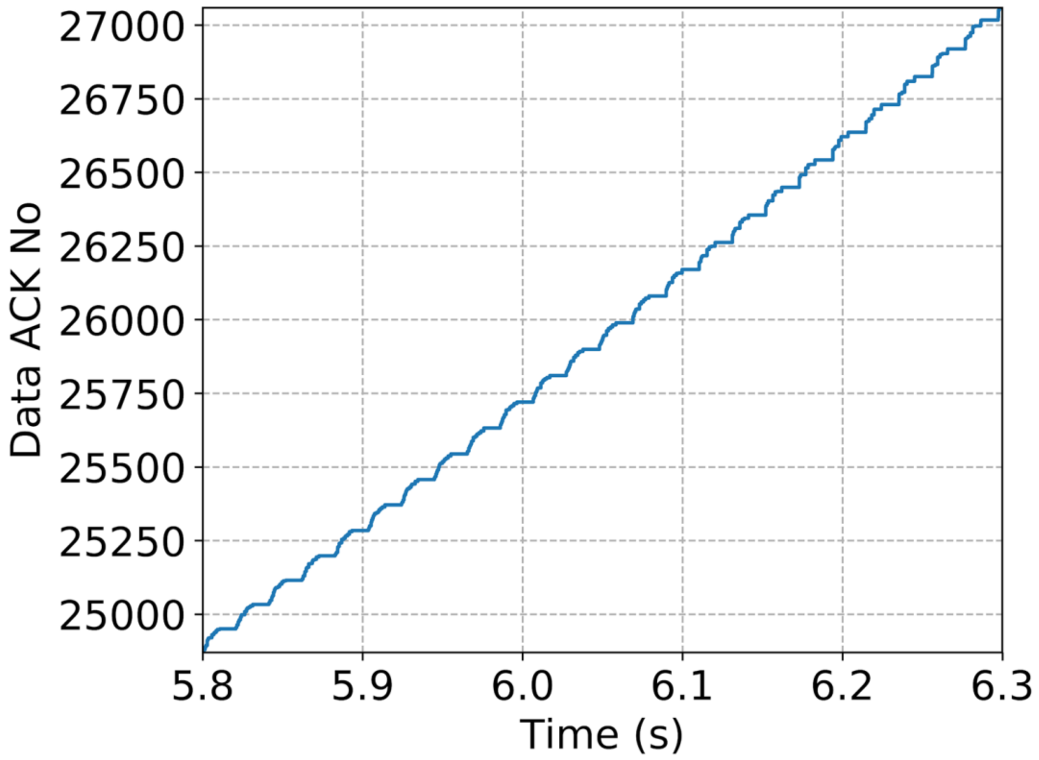


MPTCP-level window sliding

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



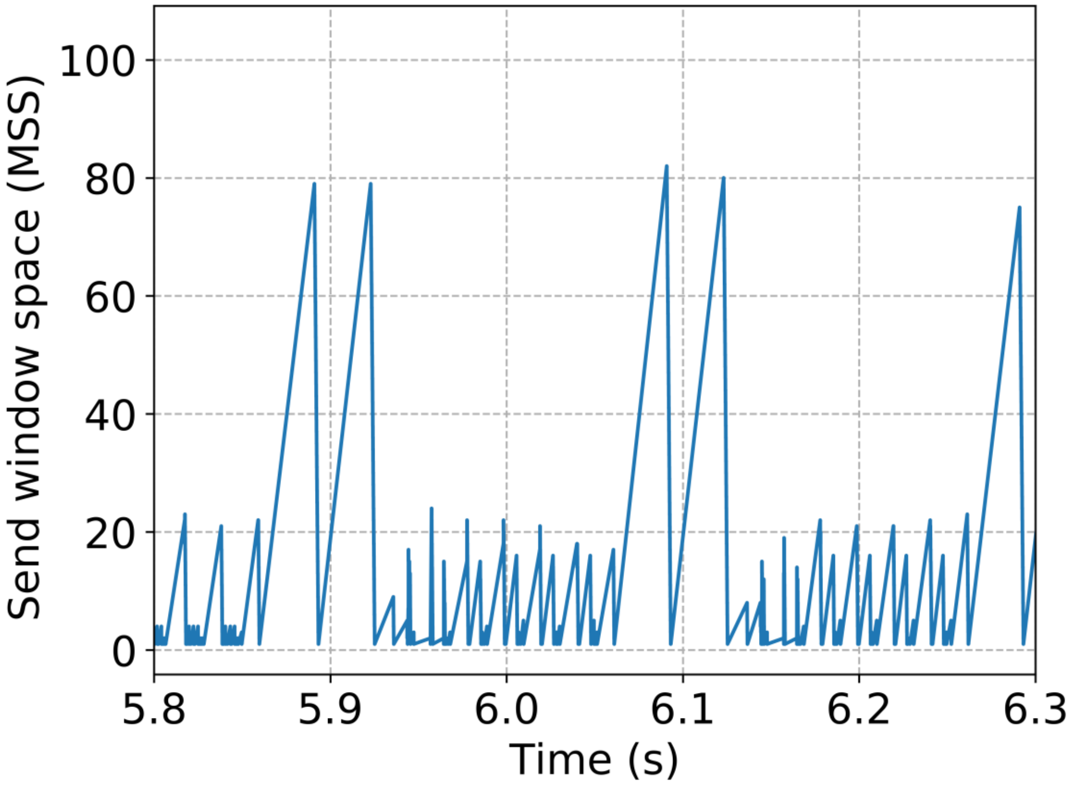
20ms vs 200ms



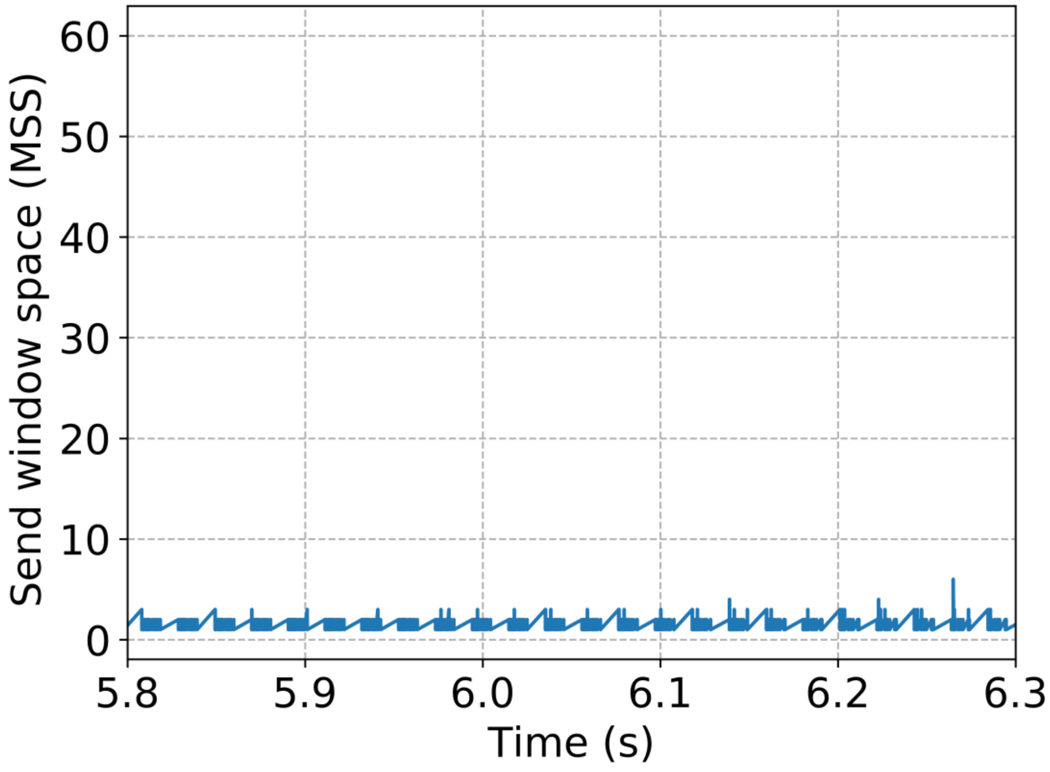
20ms vs 20ms

CWND free space of fast path

- Break the ACK clocking of single TCP.



20ms vs 200ms



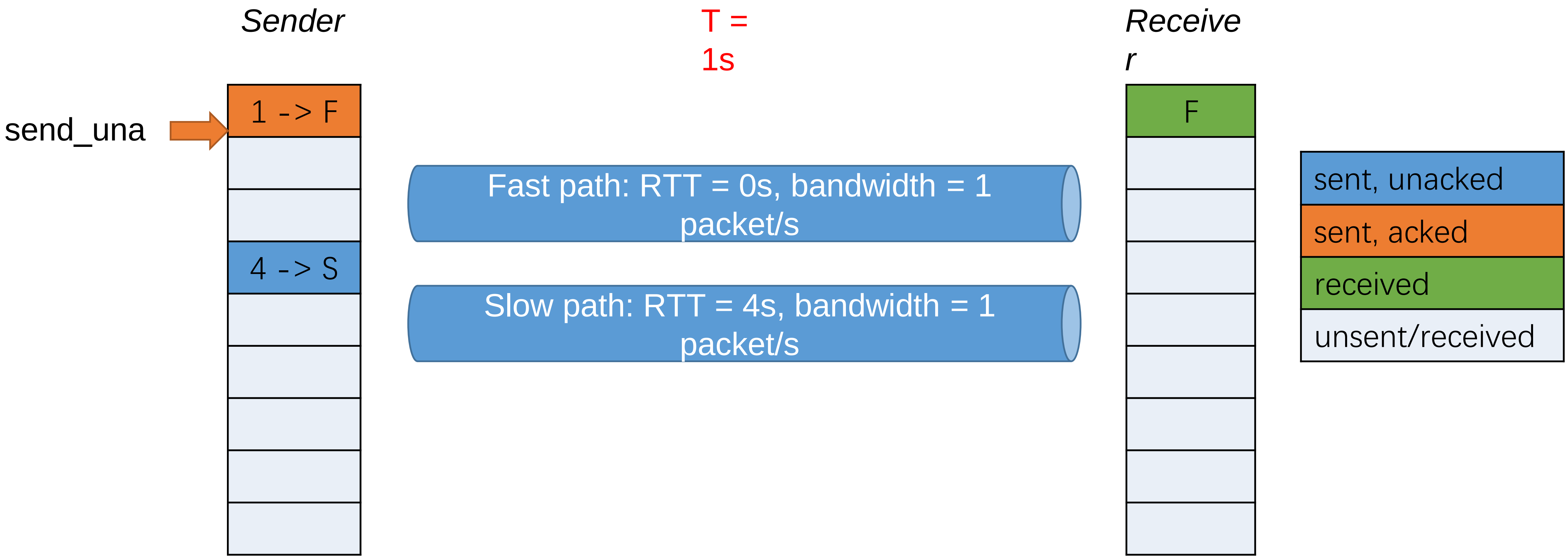
20ms vs 20ms

Solution space

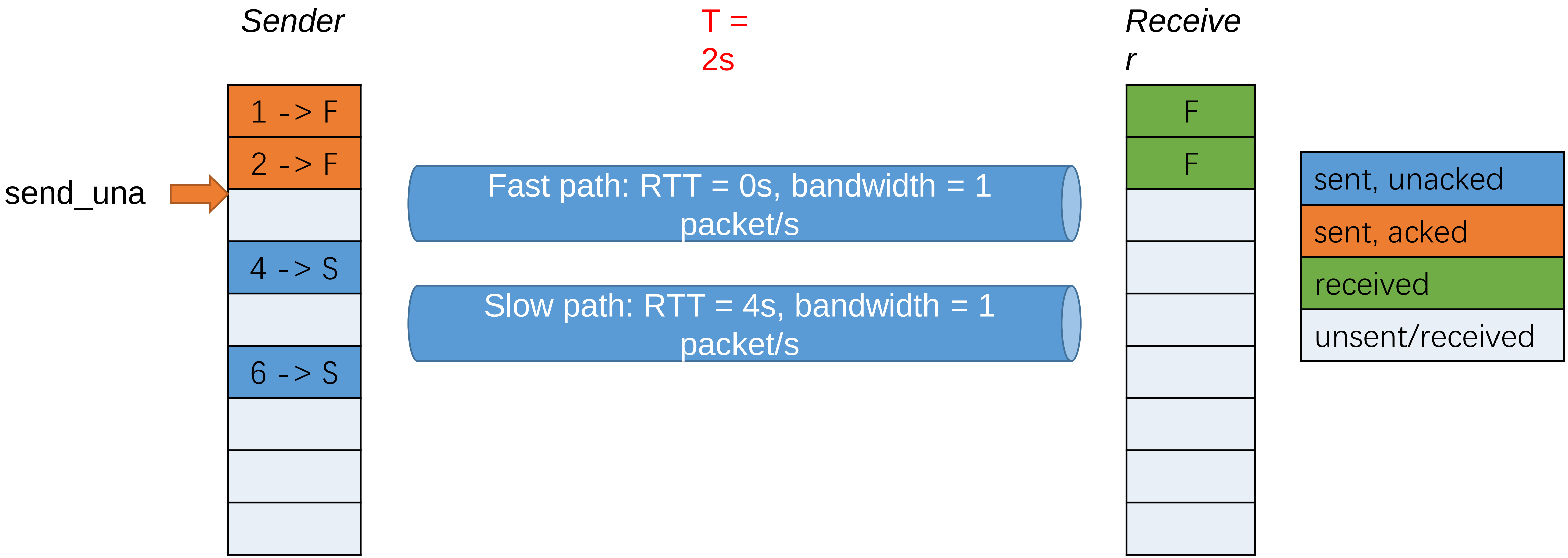
- Retransmission and penalization[1] can alleviate host buffer problem. Can not solve in-network buffer problem
- Pacing can solve in-network buffer problem.
  - TC pacing, need set the pacing rate manually
  - BBR congestion control, not fair with single path TCP
- Our solution: Dynamically out-of-order 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

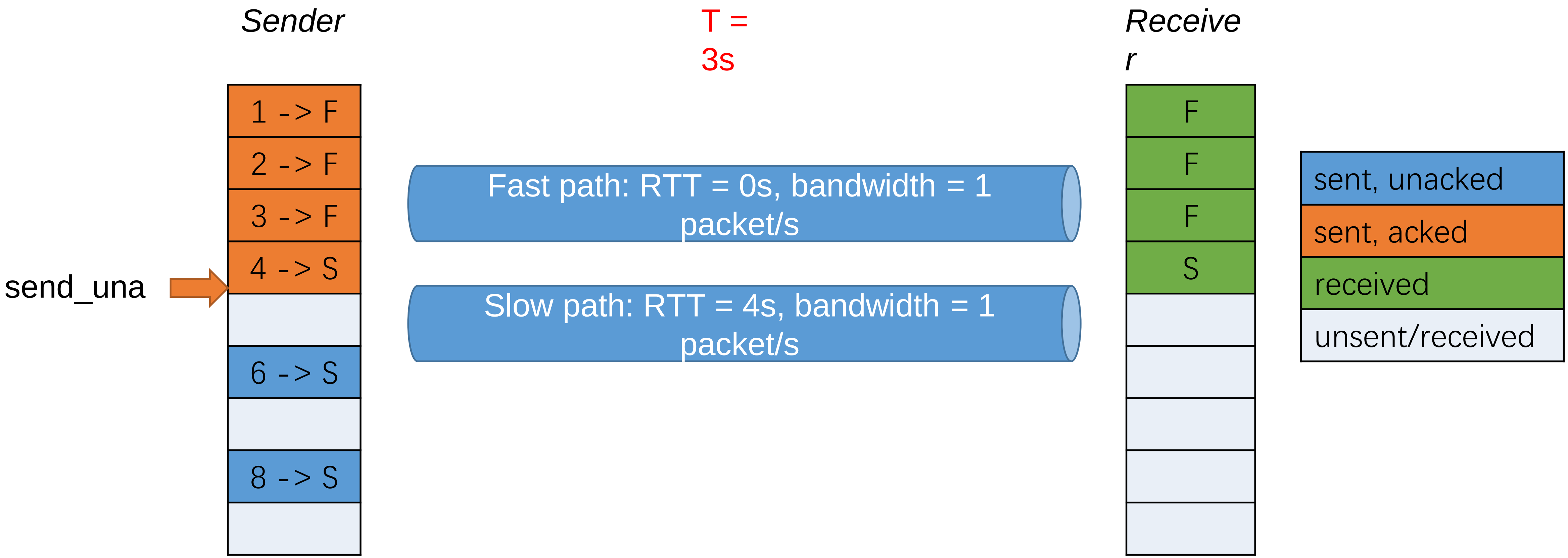
Out-of-order sending



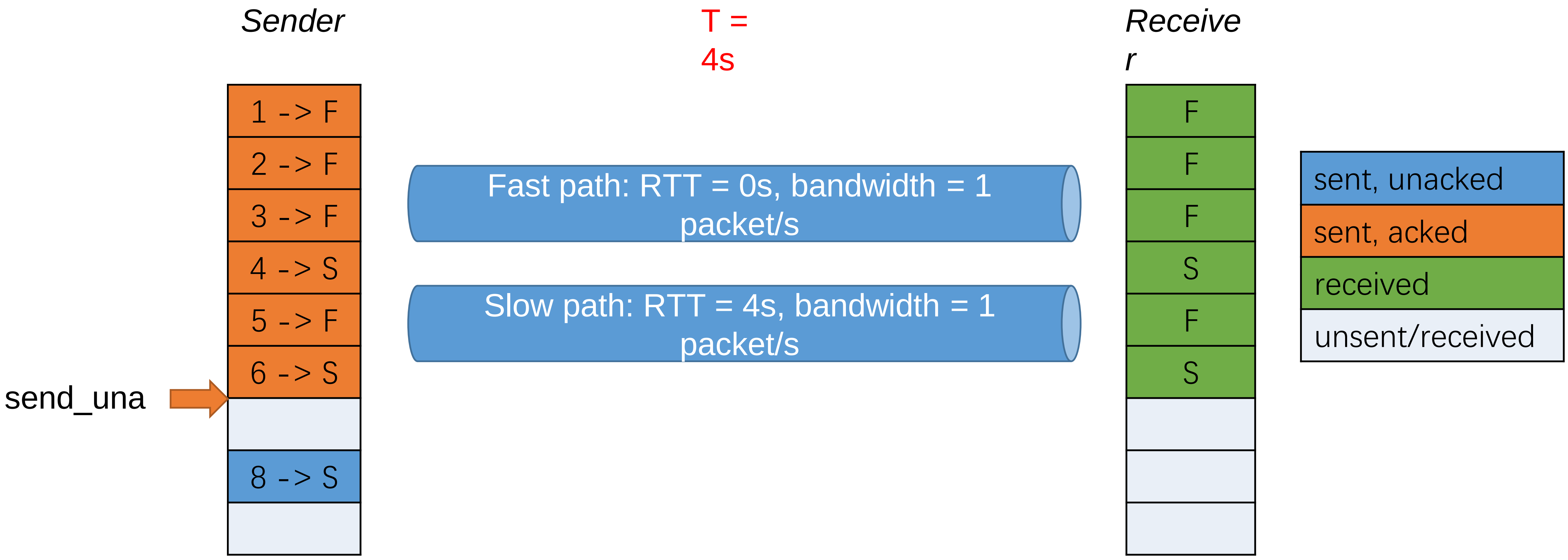
Out-of-order sending



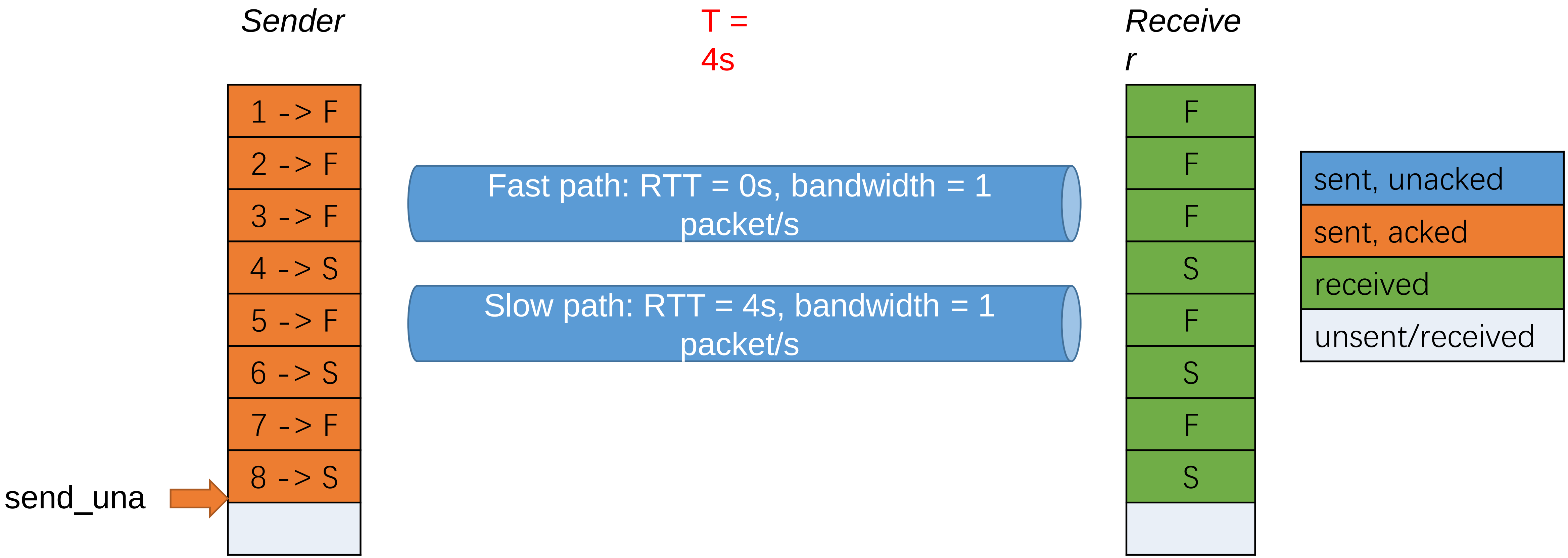
Out-of-order sending



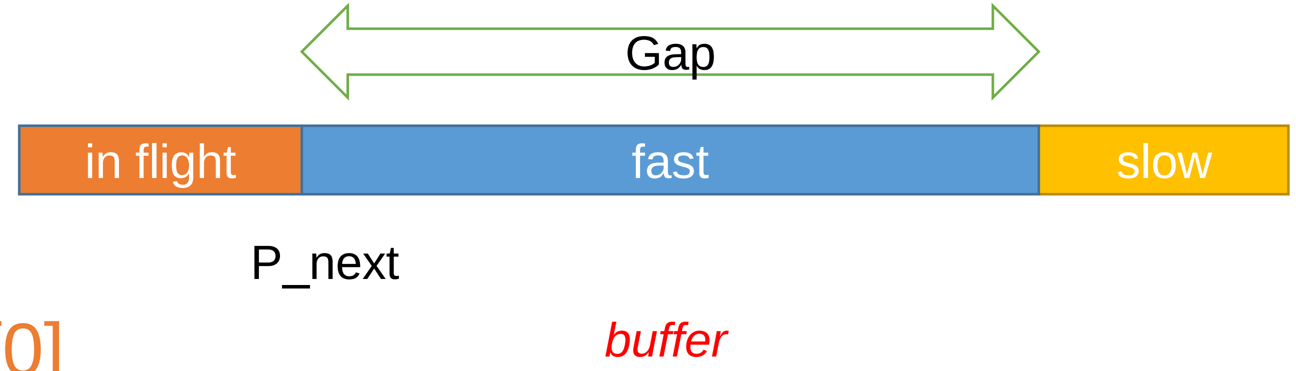
Out-of-order sending



Out-of-order sending



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 -> in-order arrival
- $\frac{Gap}{Bandwidth(fast)} + Delay(fast) = Delay(slow)$

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.

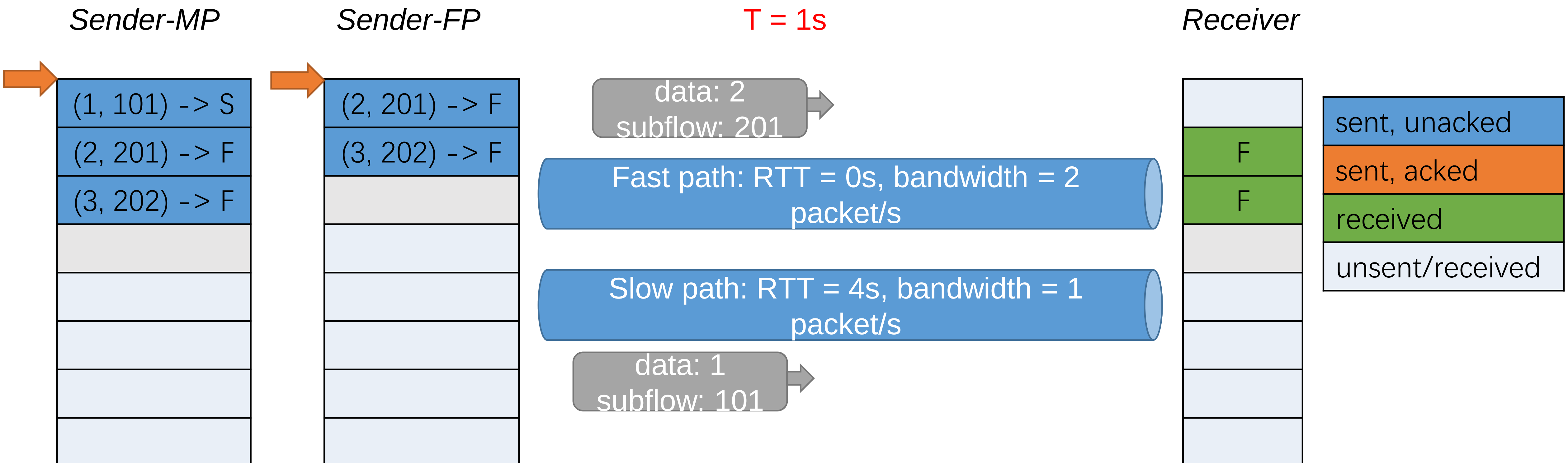
How to get GAP value

- Naive way: Calculate from path condition measurement.
  - $Gap = Bandwidth(fast) * (Delay(slow) - Delay(fast))$
  - 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

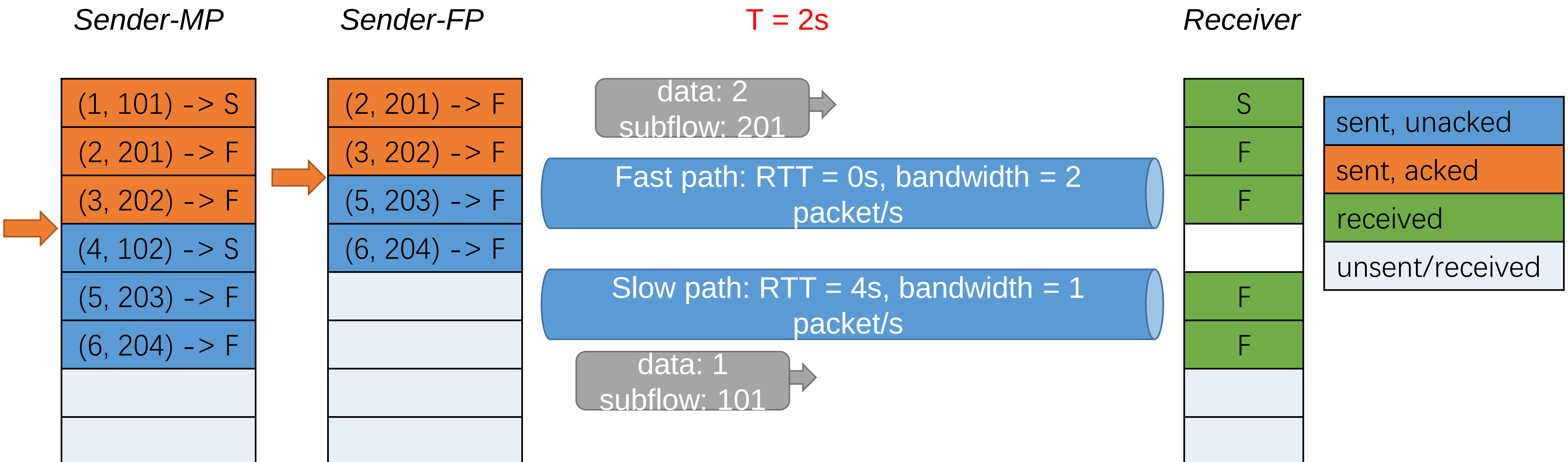
Key insight

- 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

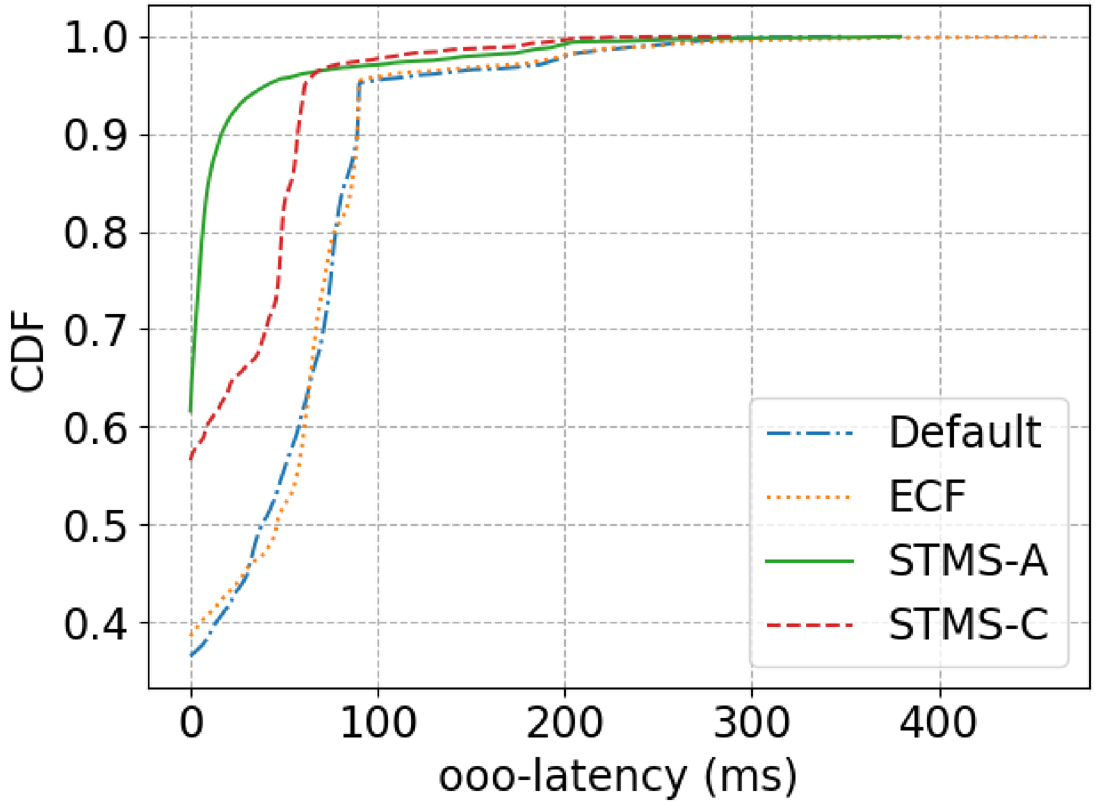
- Burst MP-level ACK(data ACK)
  - Packet[send_una] sent from slow path, Gap += delta[data_ack] - 2
  - Packet[send_una] sent from fast path, Gap -= delta[data_ack] – 2
- Limit the frequency of adjustment to avoid repeated adjustment.
- EWMA of delta over adjustment interval.

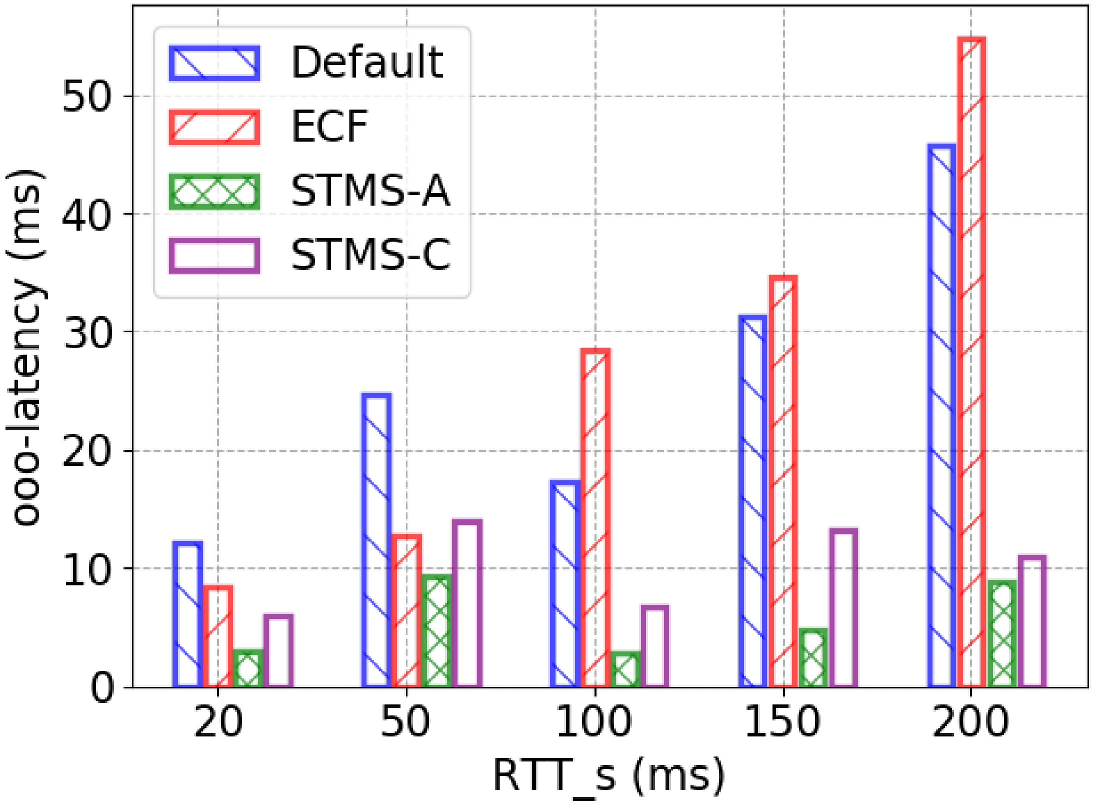
Implementation and Evaluation

- 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

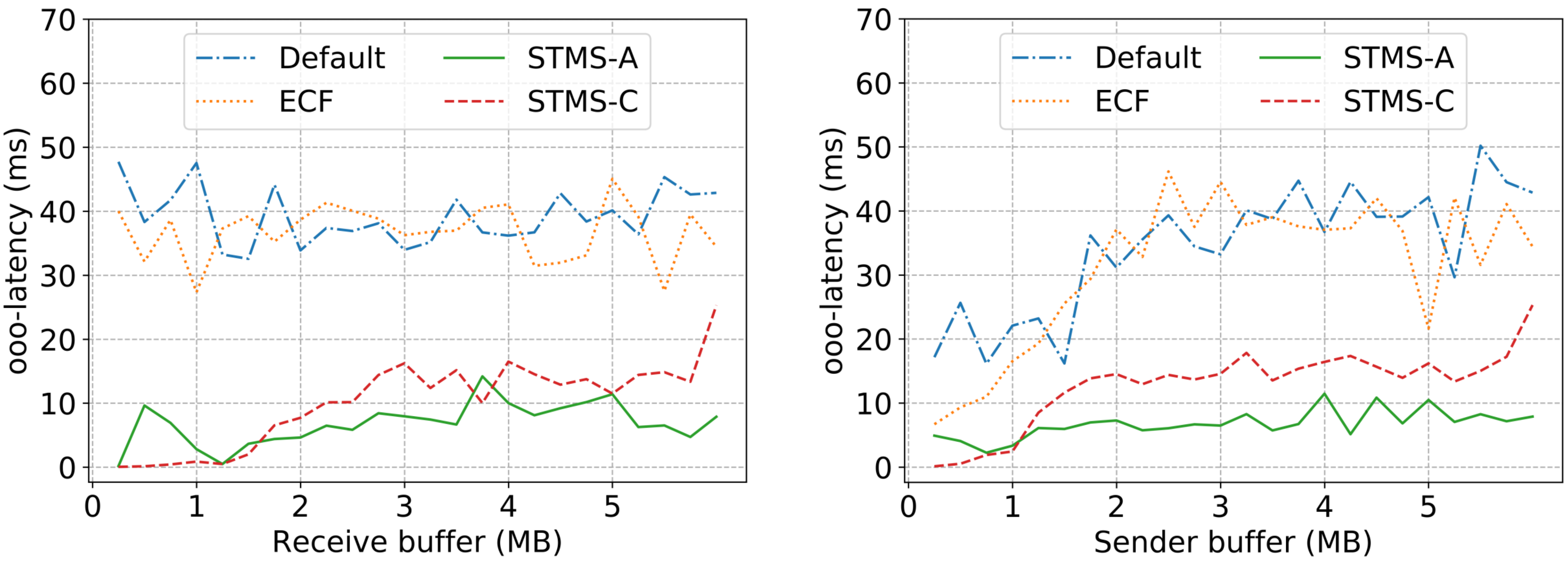
- Reduce out-of-order latency: *t(submitted) – t(arrival)*





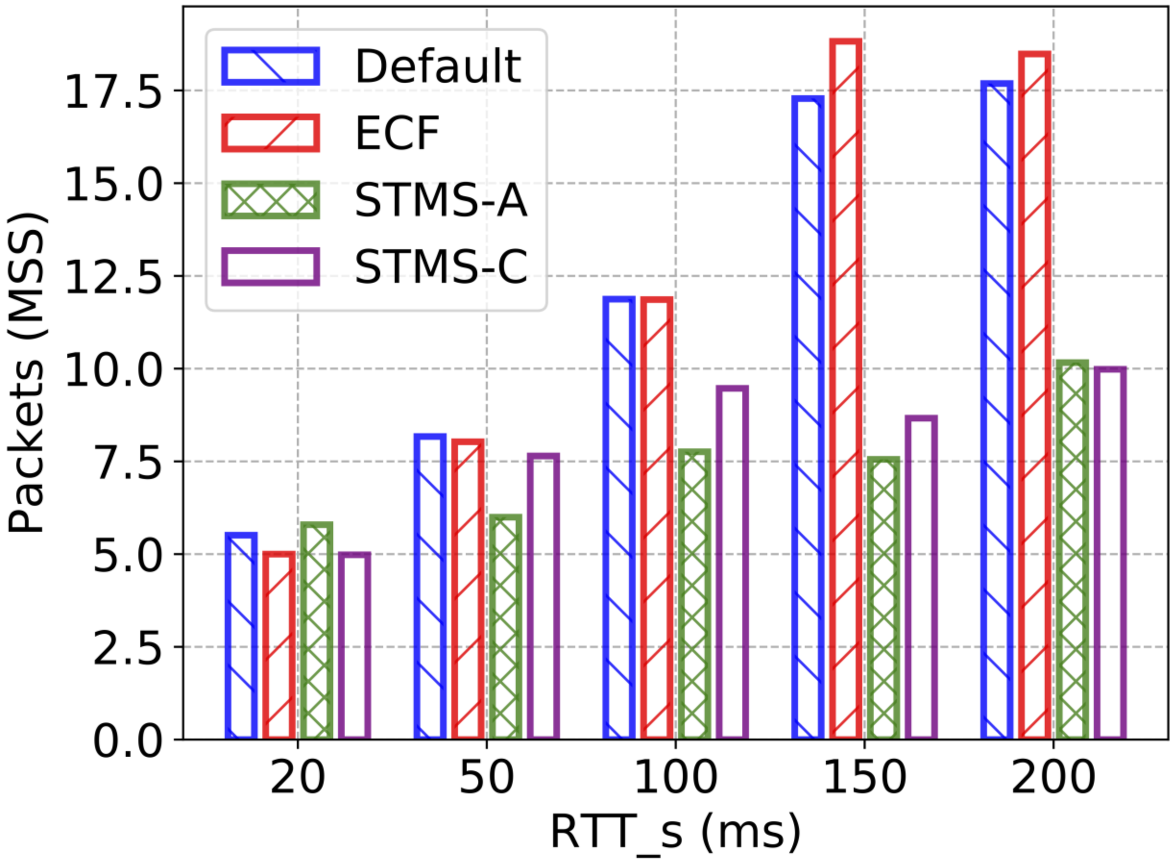
Microbenchmarks

- Varying receive buffer and send buffer size.



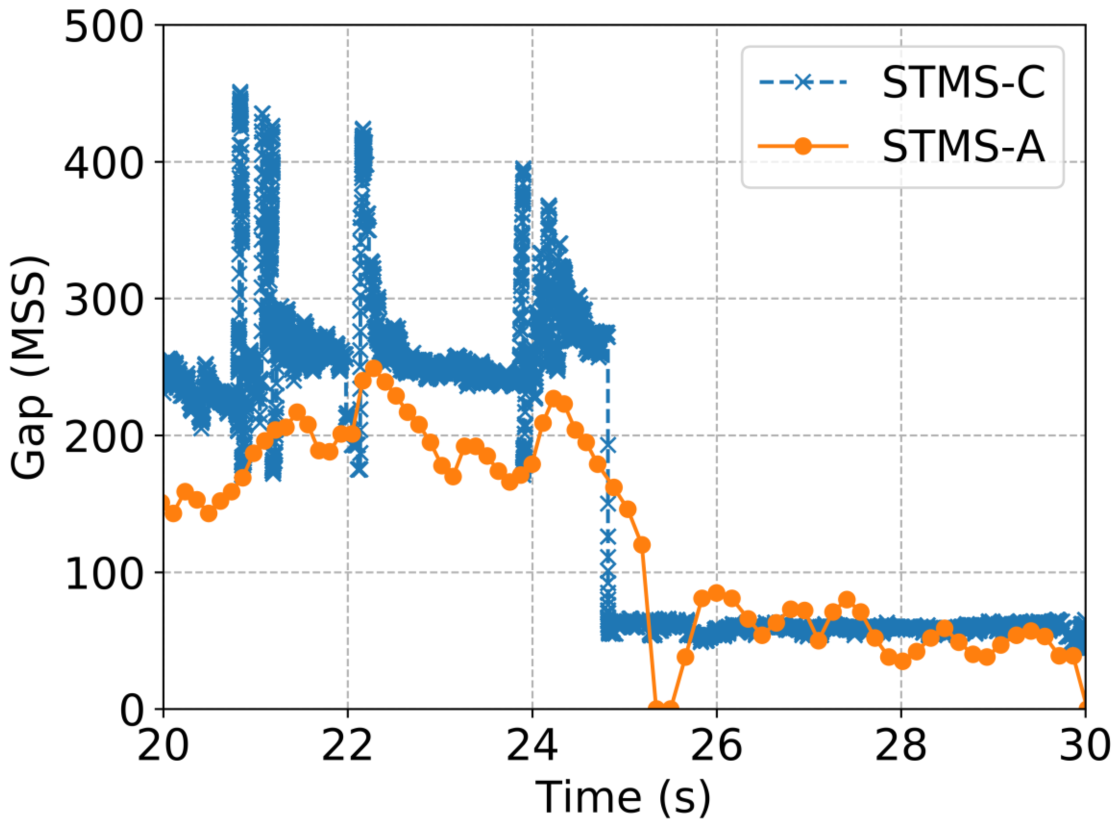
Reduce burst on the fast path

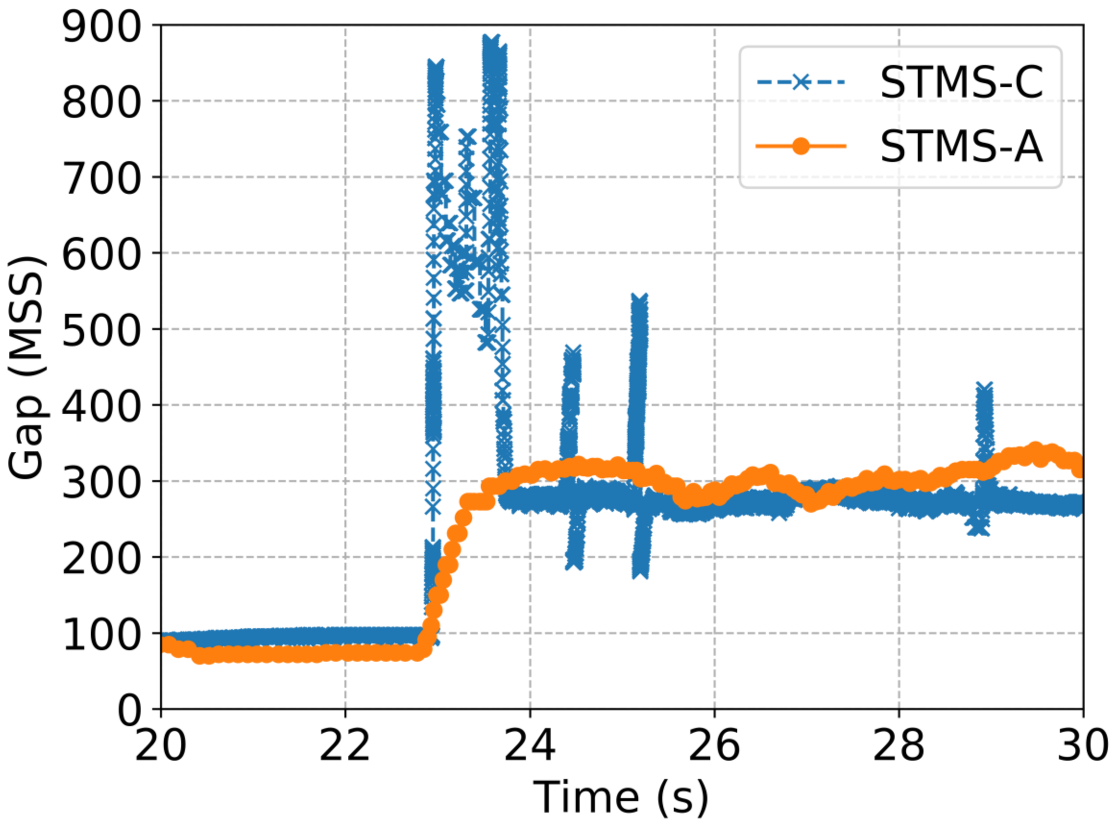
- CWND freespace when receiving ACK.
- iPerf will fill the freespace.
- Big freespace -> burst sending -> big in-network buffer requirement.



Gap adjustment is dynamic

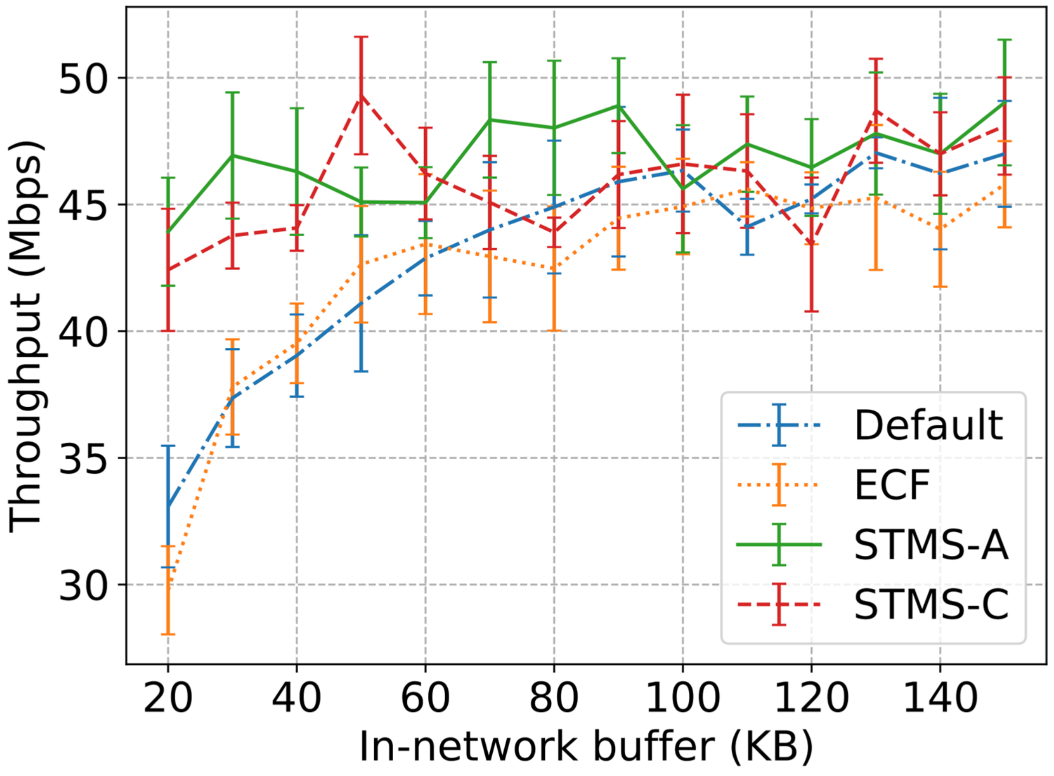
- Change the network condition suddenly.

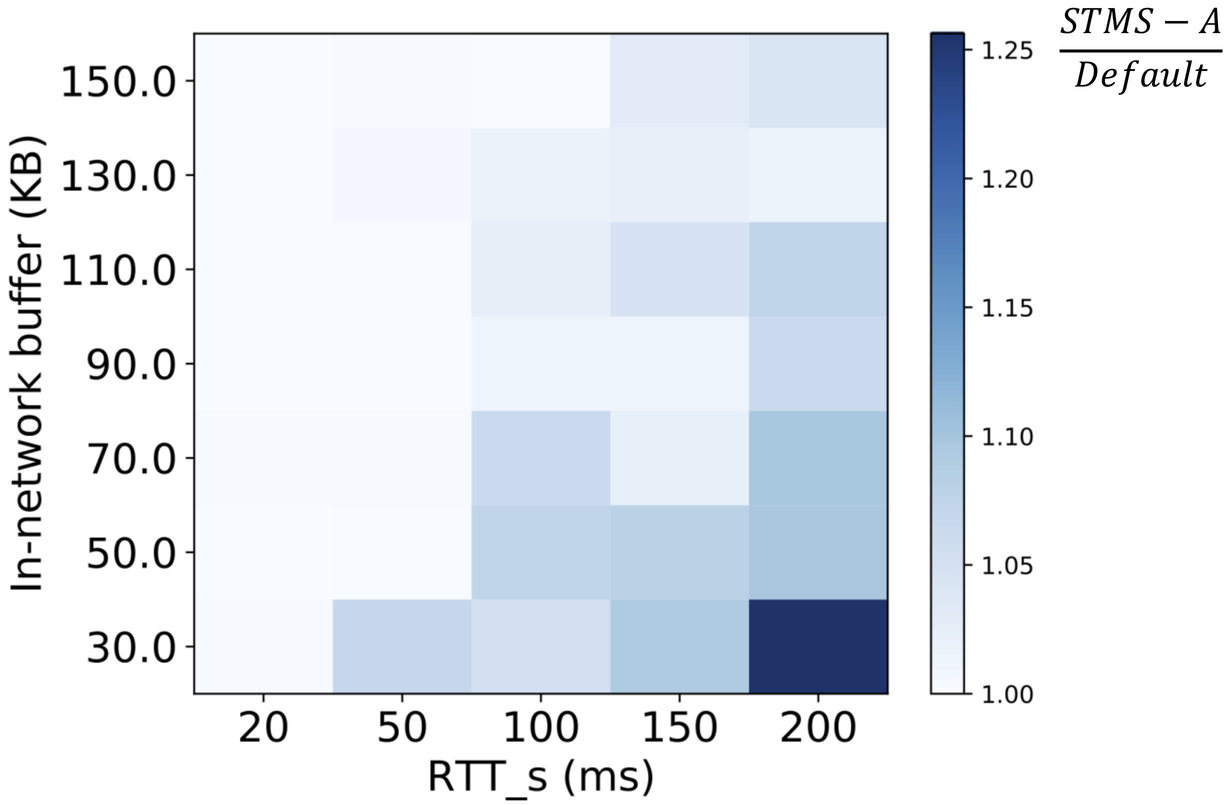




Macrobenchmarks

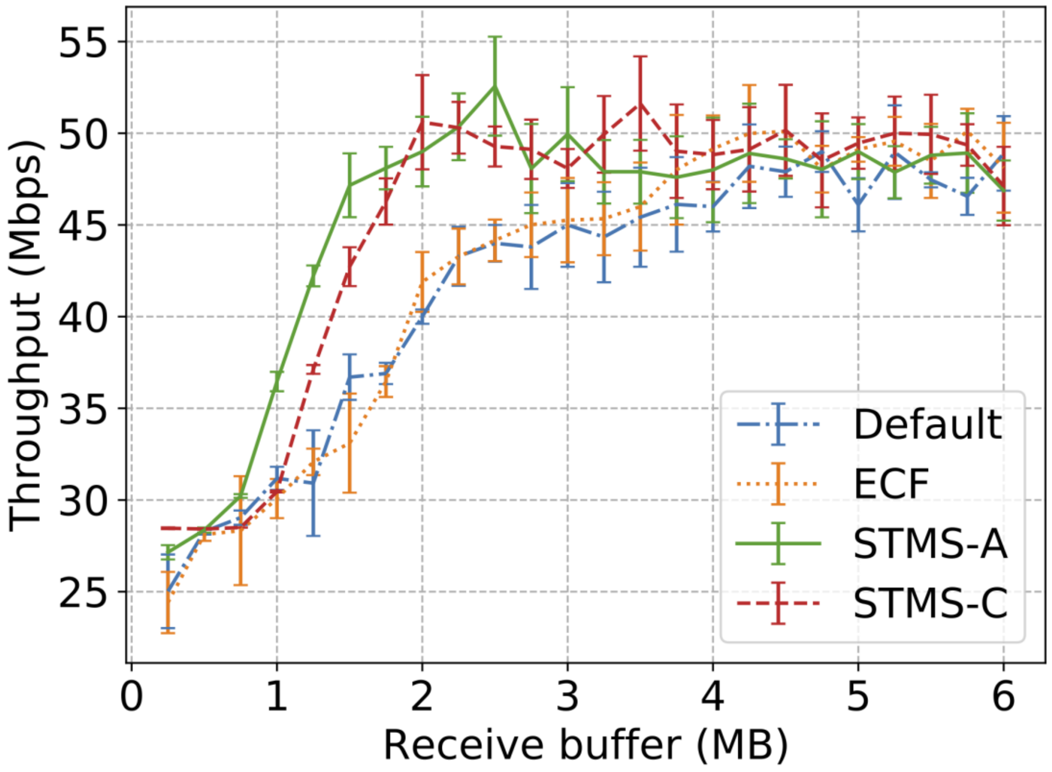
- 25% improvement when in-network buffer is limited.

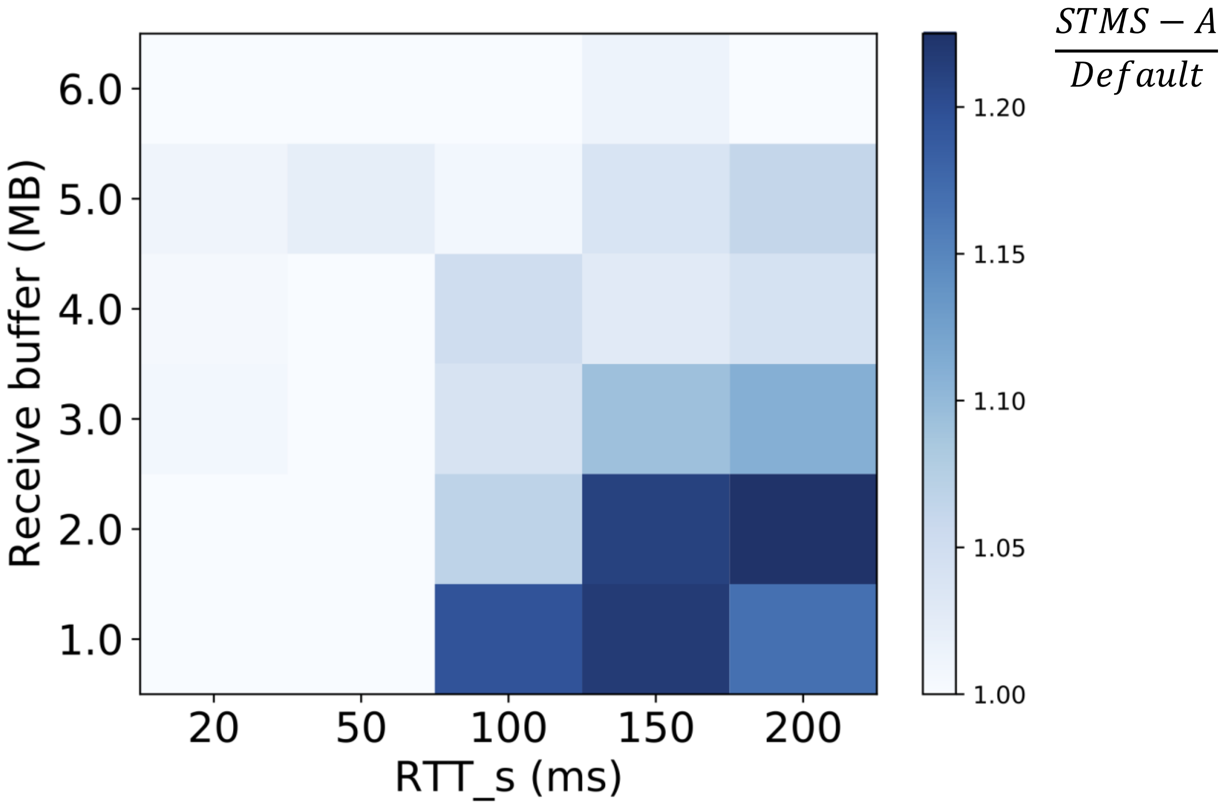




Host buffer

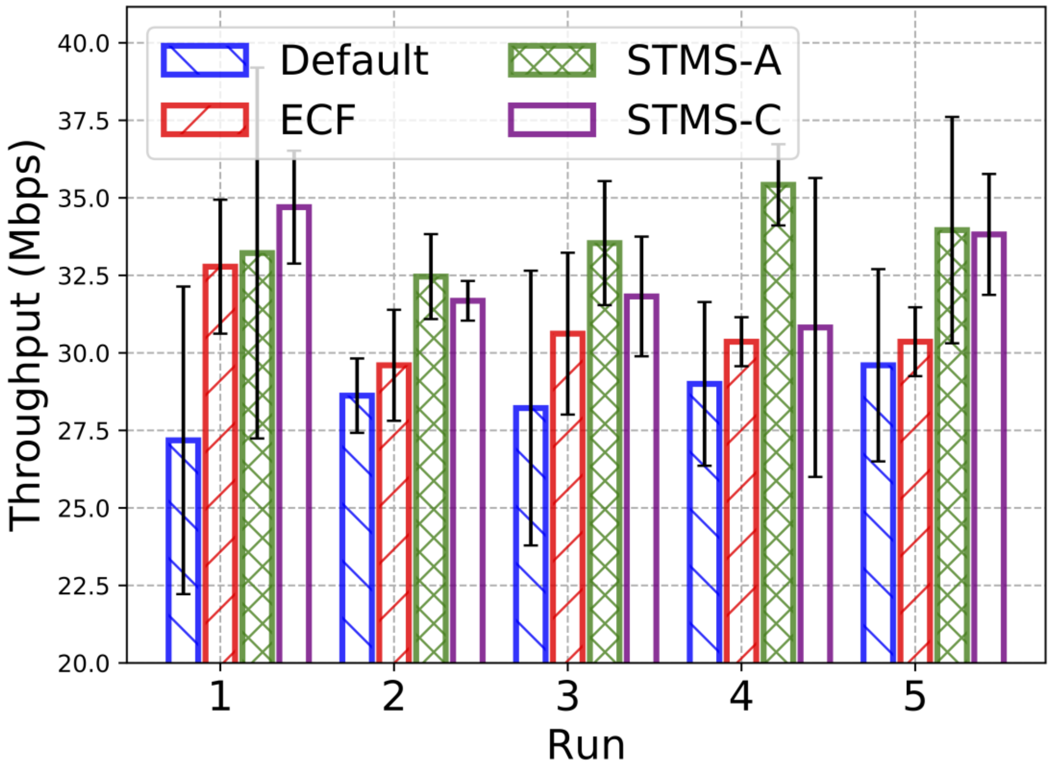
- 20% improvement when receive/send buffer is limited.

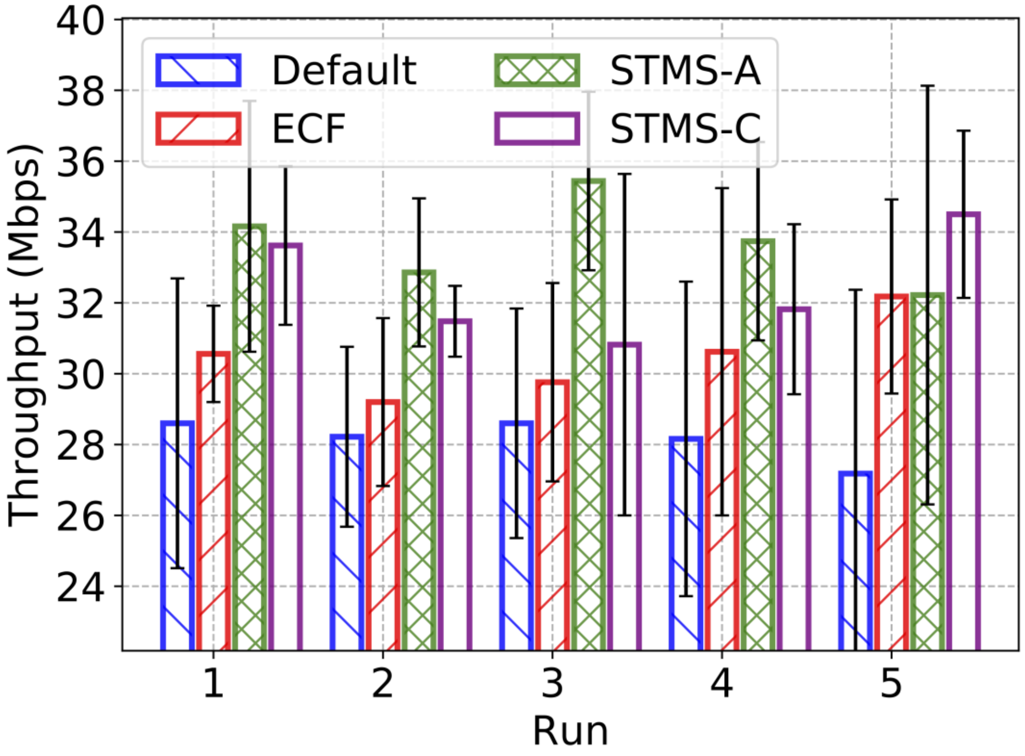




Dynamic network condition

- Change bandwidth(left) and latency(right) randomly

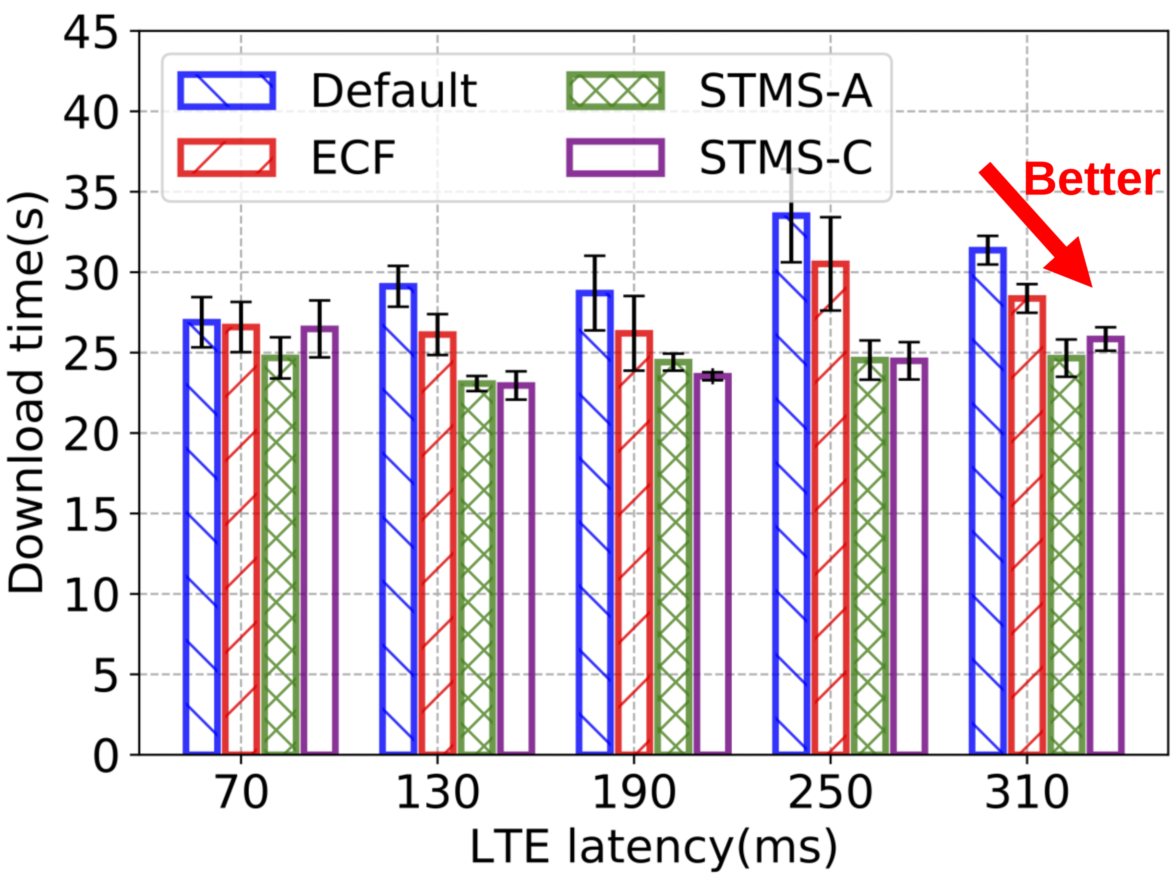




Real-world evaluation

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

	BD(Mbps)	Latency(ms)
WiFi	40	50
LTE	30	70



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