Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Centers

Peng Cheng, Fengyuan Ren, Ran Shu, Chuang Lin

Tsinghua University





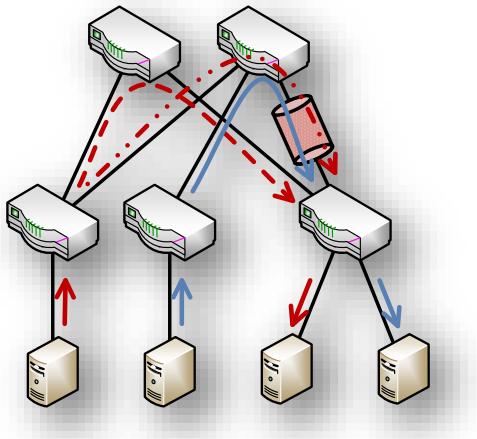




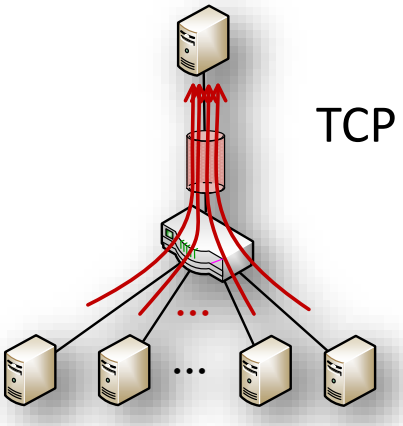




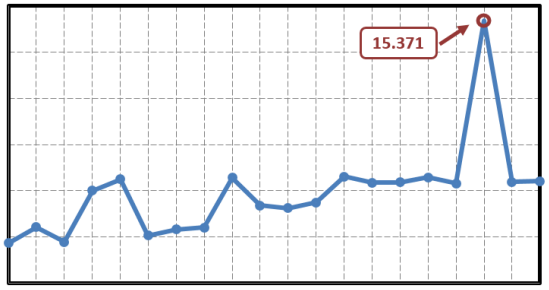




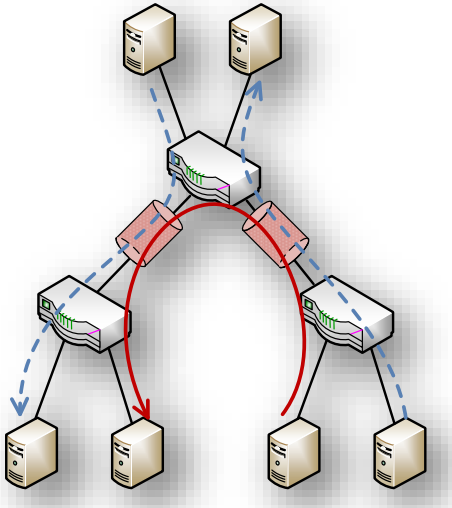
TCP Out-of-order



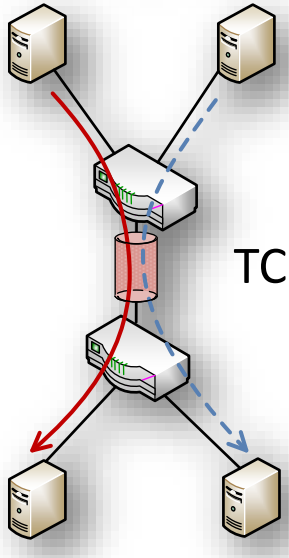
TCP Incast



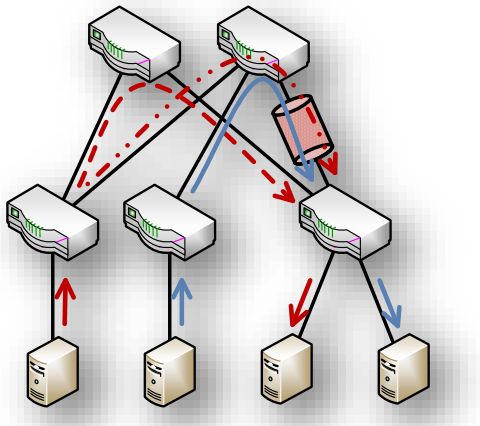
Long Query Completion Time



TCP Unfairness

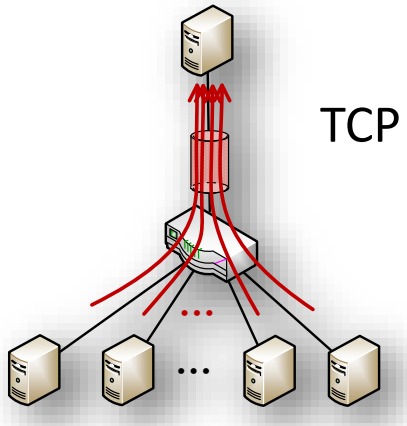


TCP Outcast

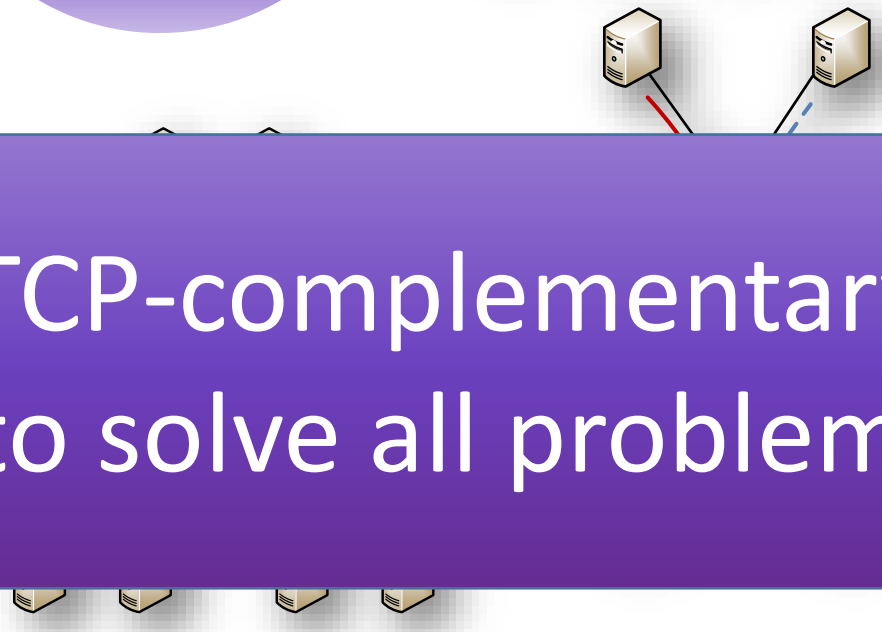


TCP Out-of-order





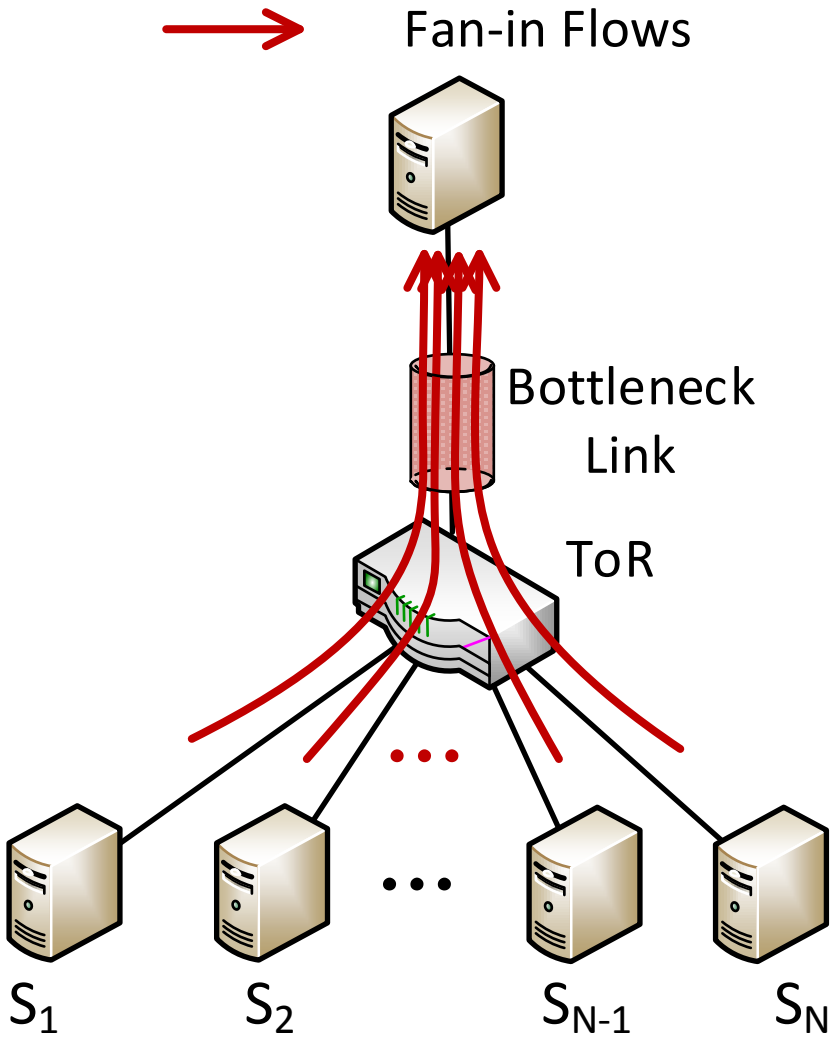
TCP Incast



TCP Unfairness

One simple TCP-complementary mechanism to solve all problem

TCP Incast



Requirement of MapReduce:

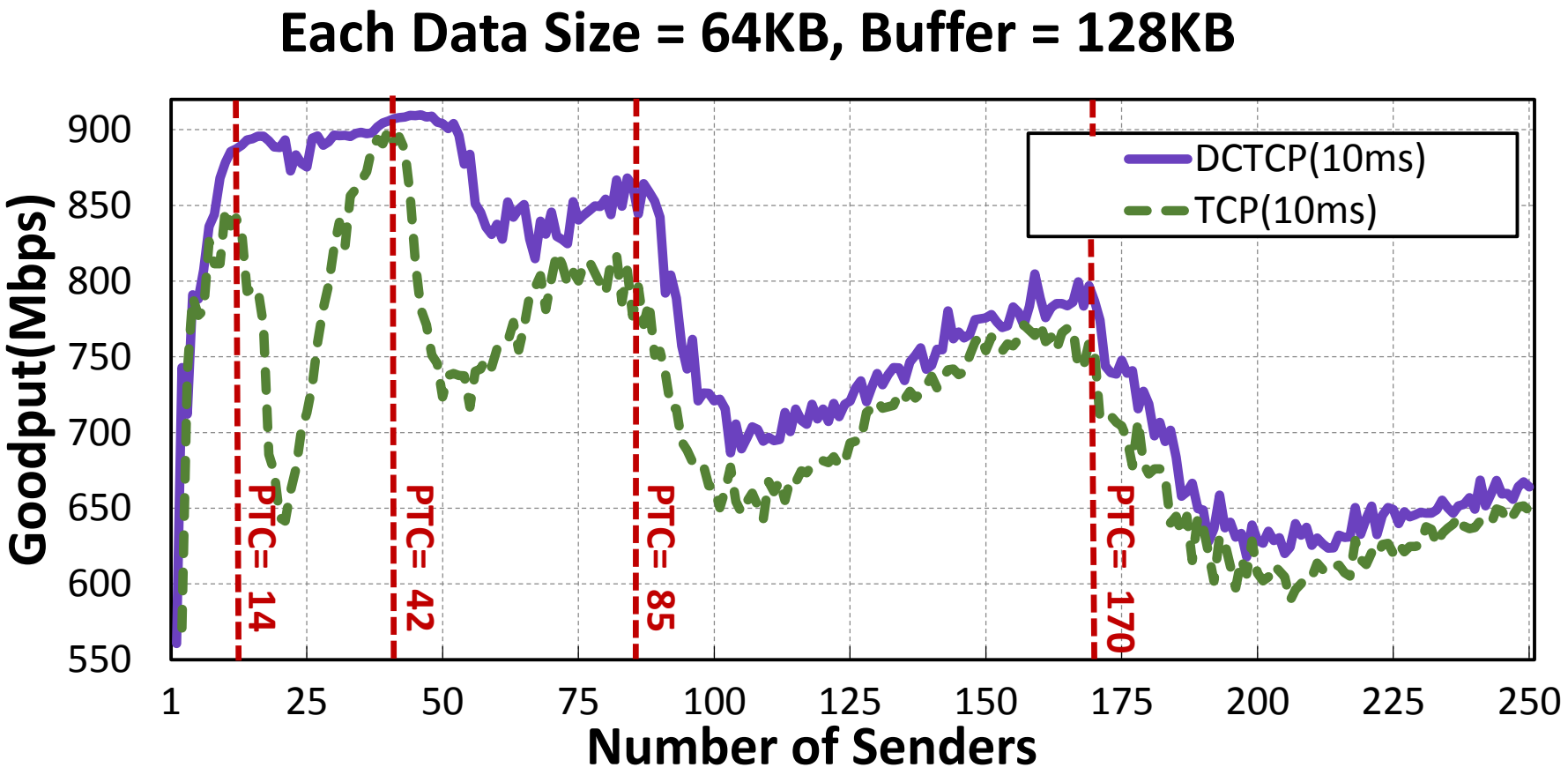
$$N = 154 \pm 558\sigma$$

[M45 Yahoo! cluster]

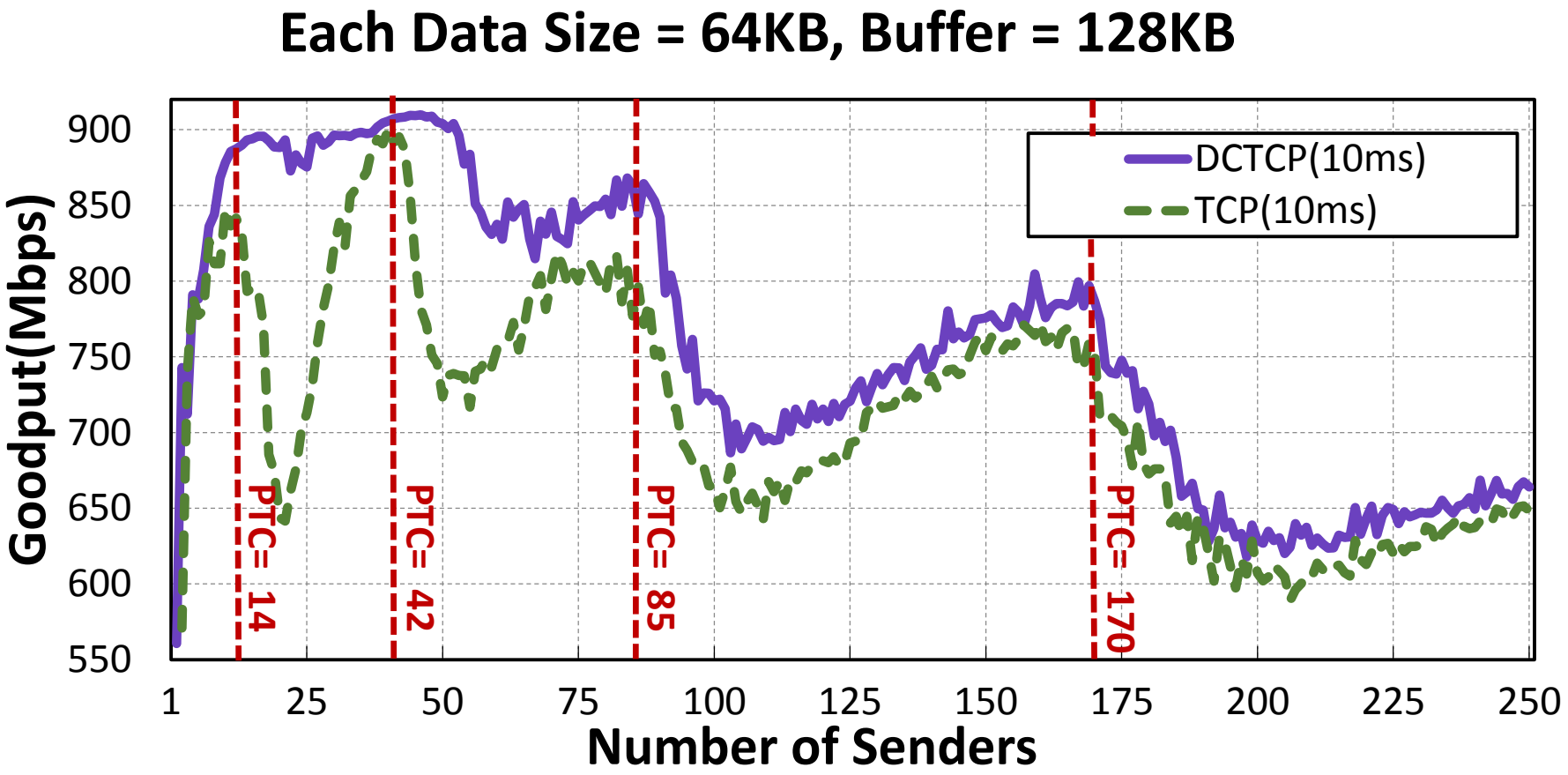
Solution:

- DCTCP X
- ICTCP X
- Reducing RTO_{min} ?

Throughput Cliff

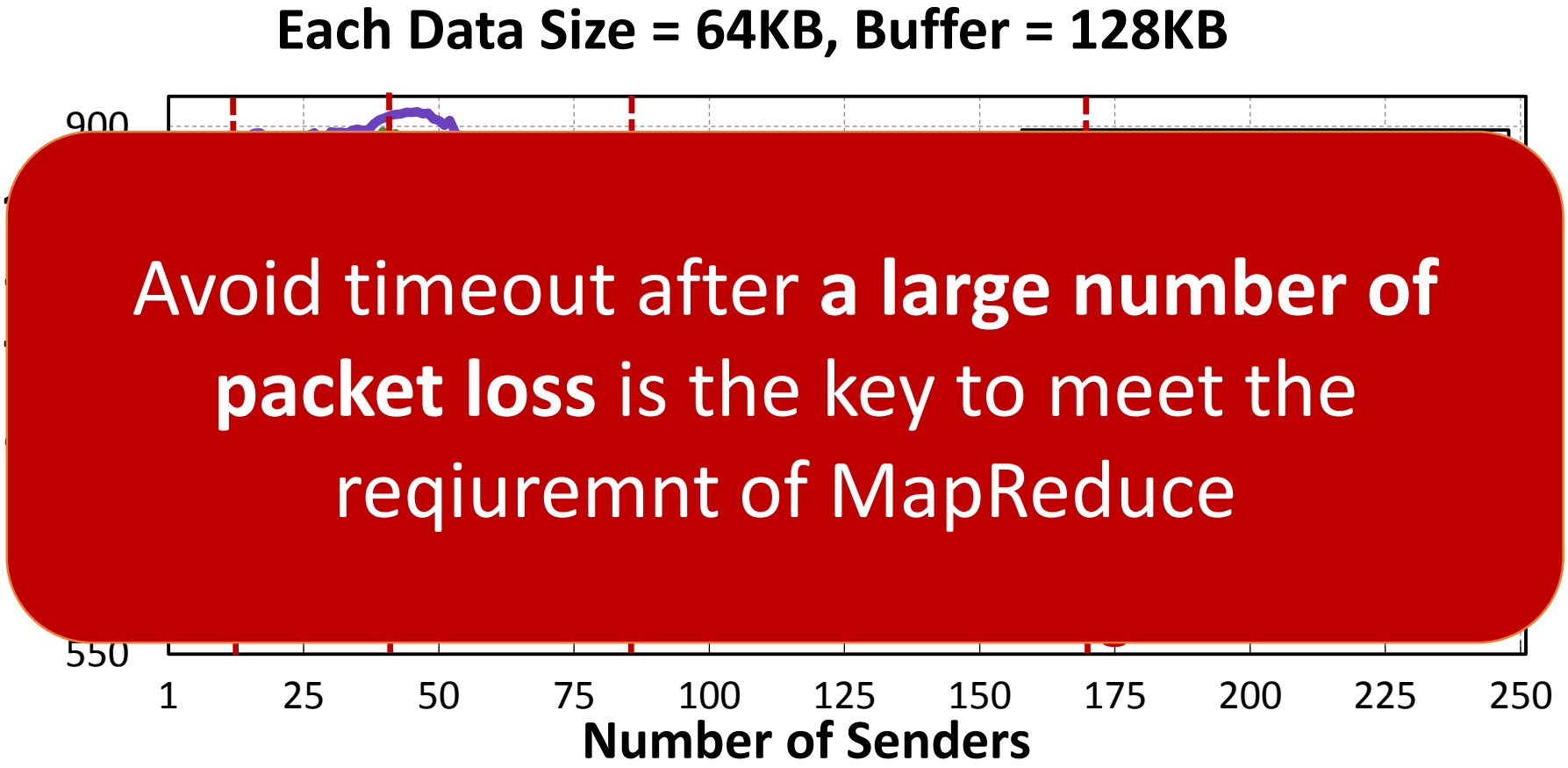


Throughput Cliff



Synchronous packet loss→ increasing Timeout

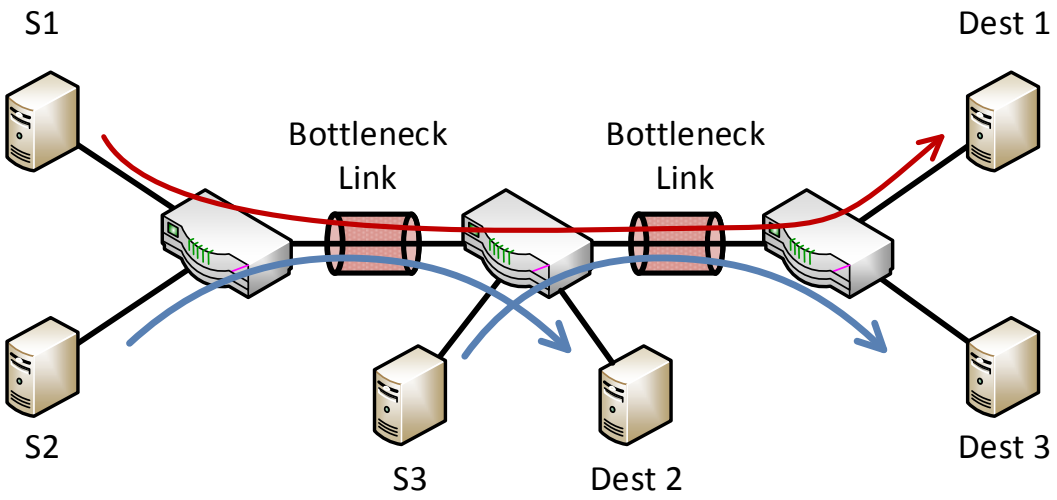
Throughput Cliff



Avoid timeout after **a large number of packet loss** is the key to meet the reqiuremnt of MapReduce

Synchronous packet loss→ increasing Timeout

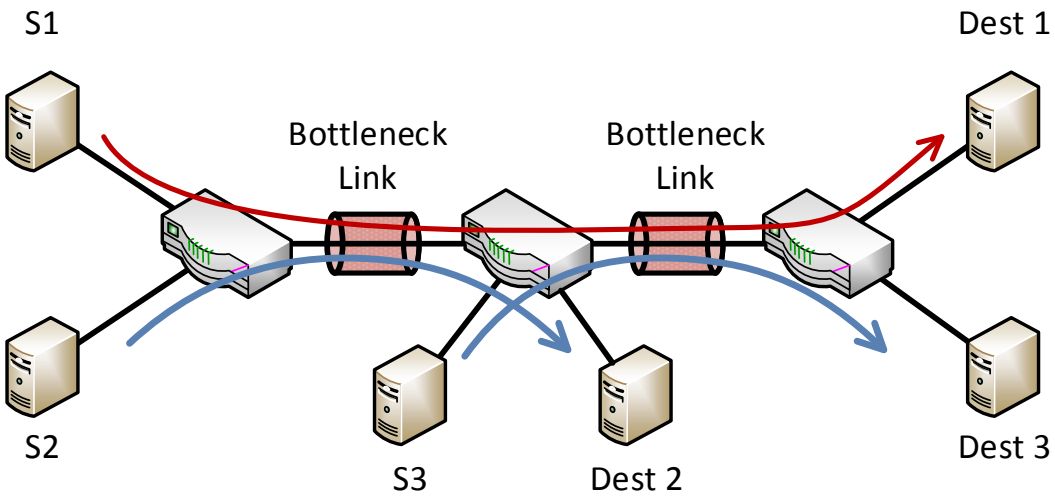
TCP Unfairness



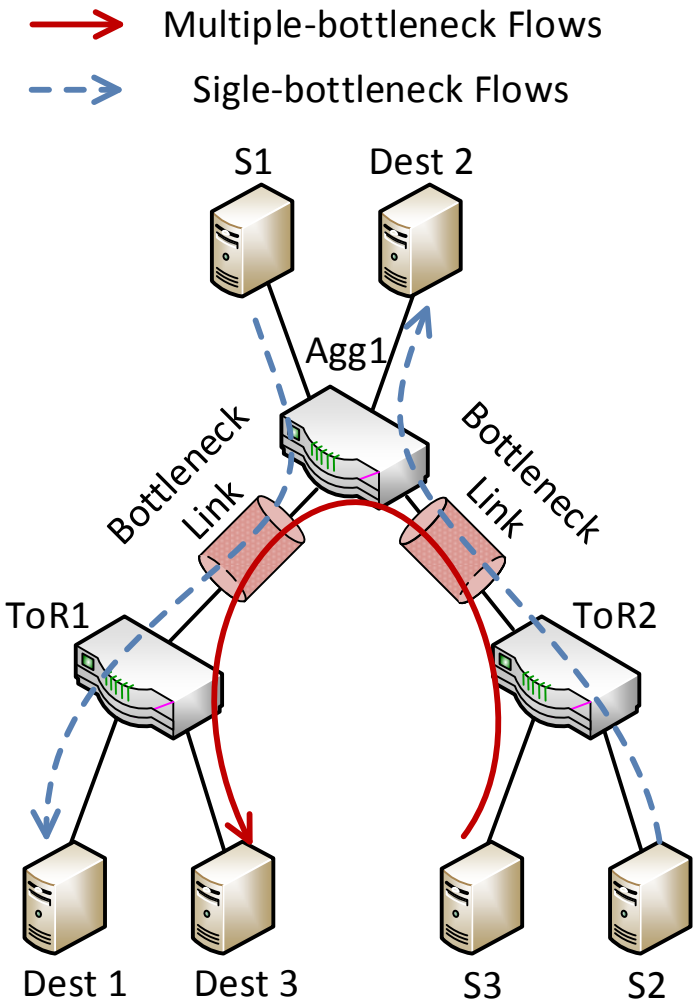
[Random Drop Congestion Control, SIGCOMM 1990]

Multiple-bottleneck Flows: Low Throughput in Internet

TCP Unfairness

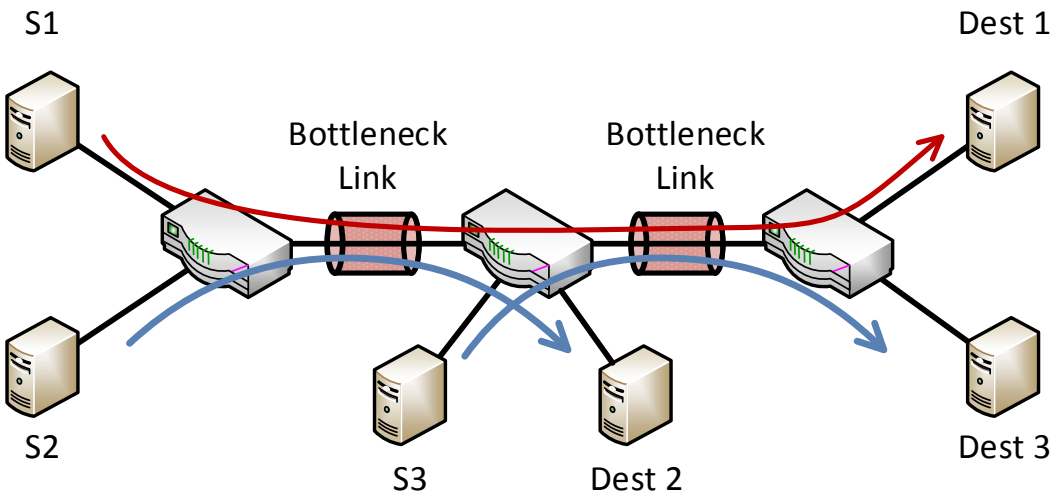


[Random Drop Congestion Control, SIGCOMM 1990]



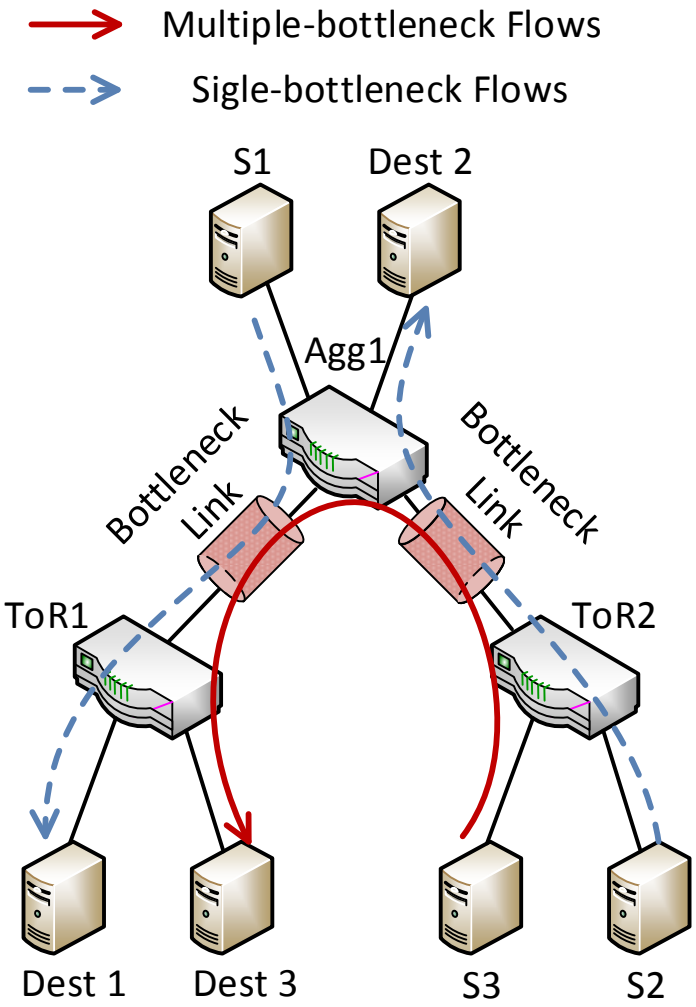
[Data Center Topology]

TCP Unfairness



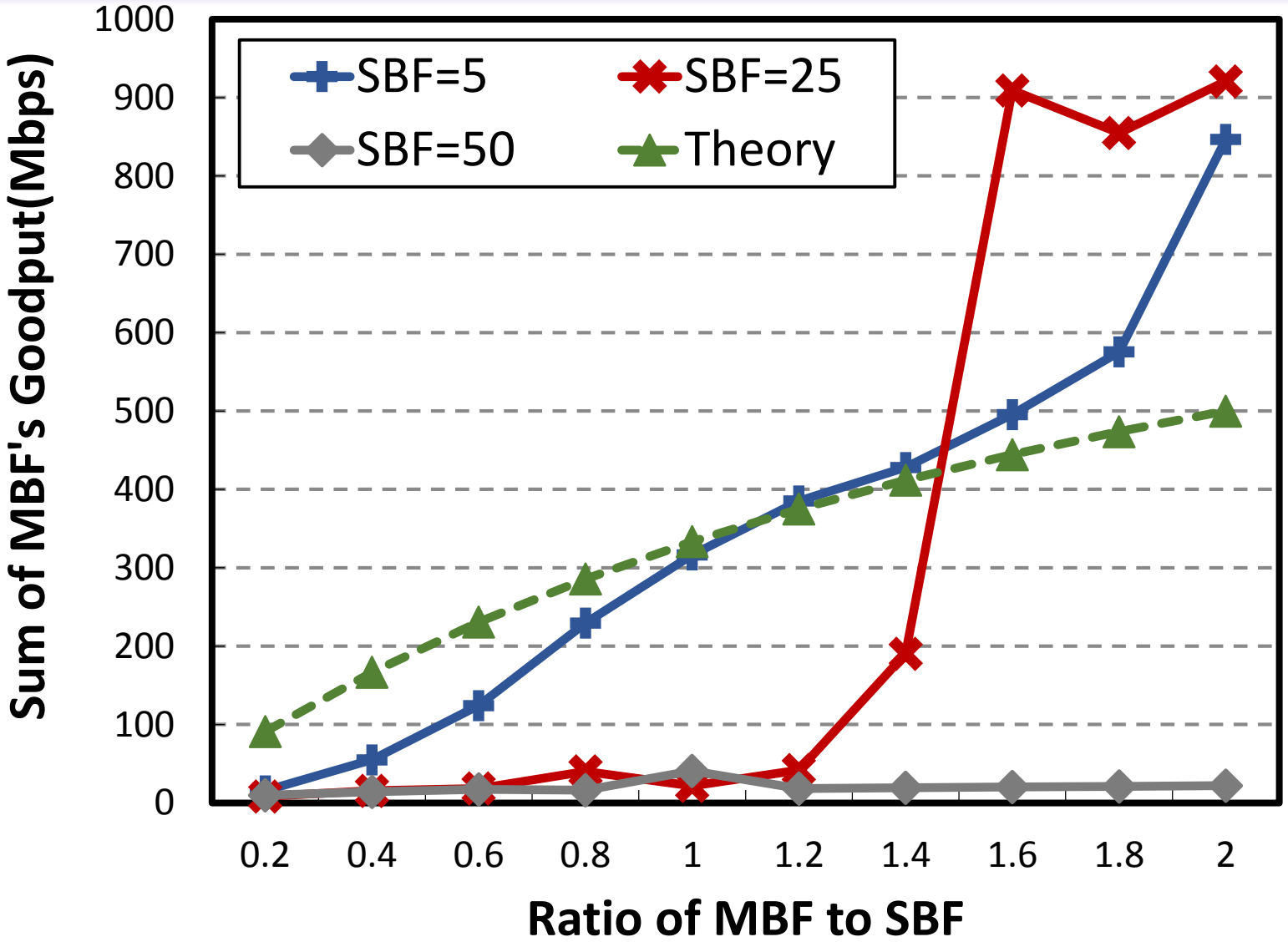
[Random Drop Congestion Control, SIGCOMM 1990]

Multiple-bottleneck Flows: ? in DCNs

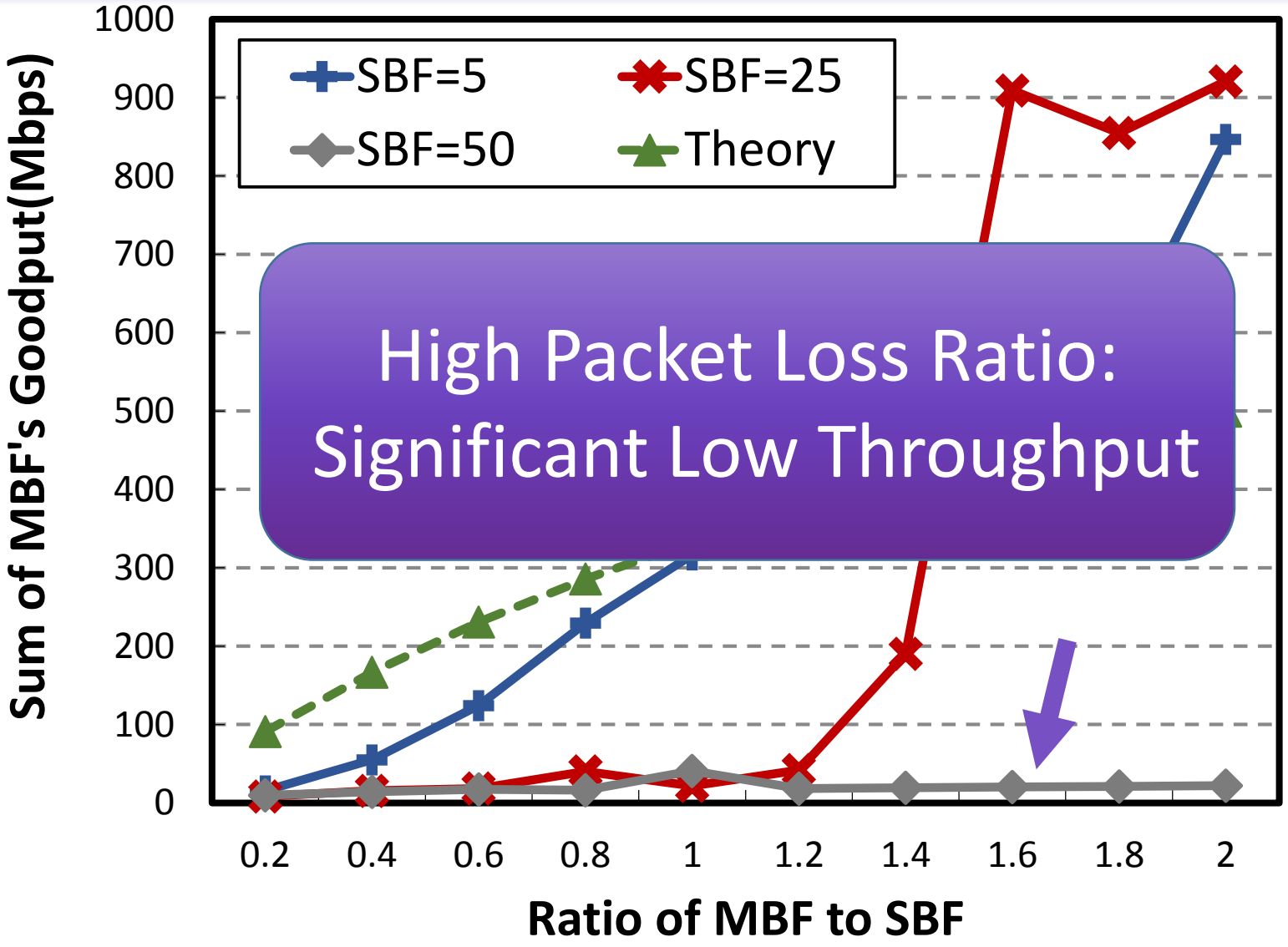


[Data Center Topology]

TCP Unfairness

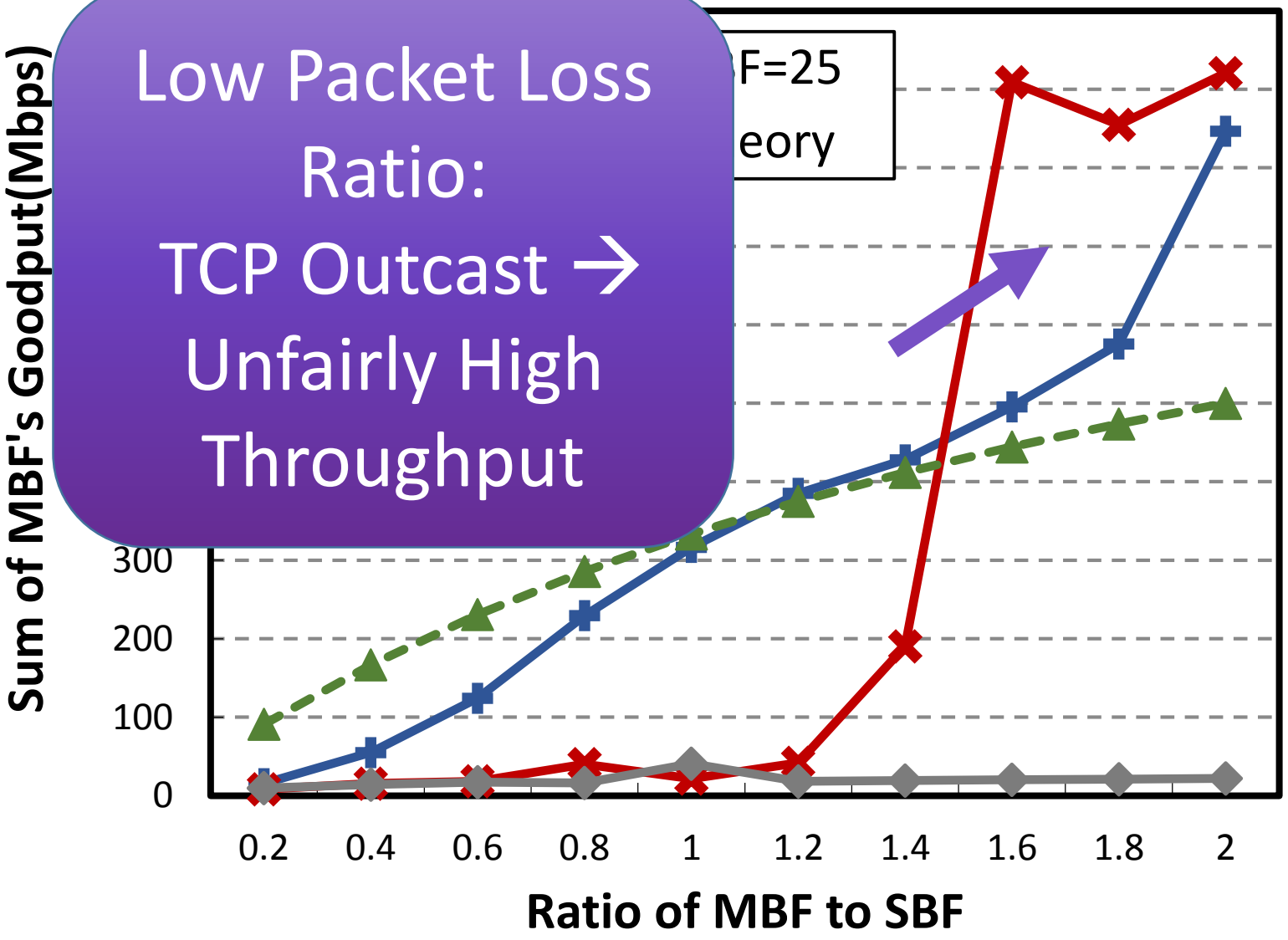


TCP Unfairness

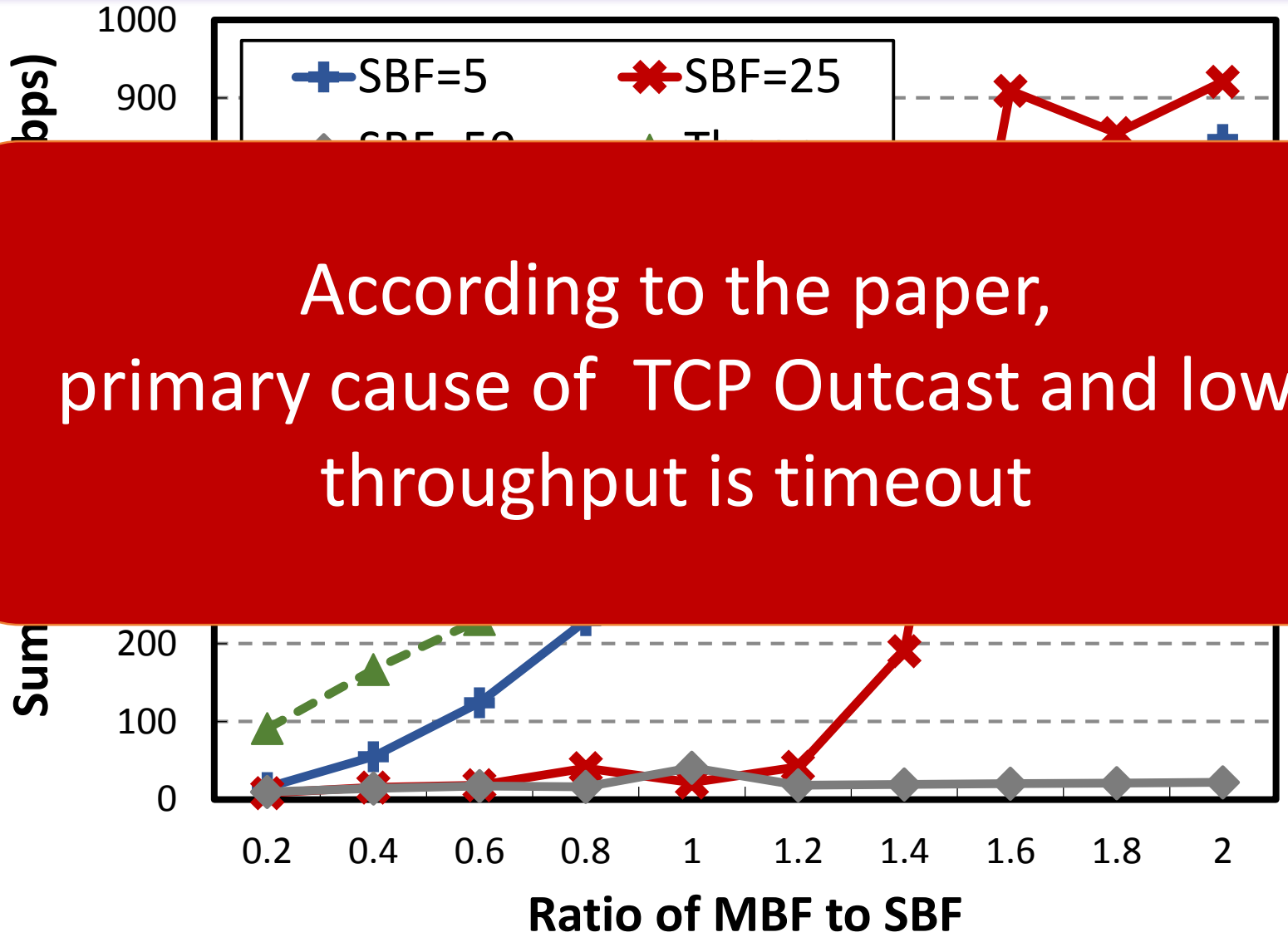


TCP Unfairness

Low Packet Loss Ratio:
TCP Outcast → Unfairly High Throughput



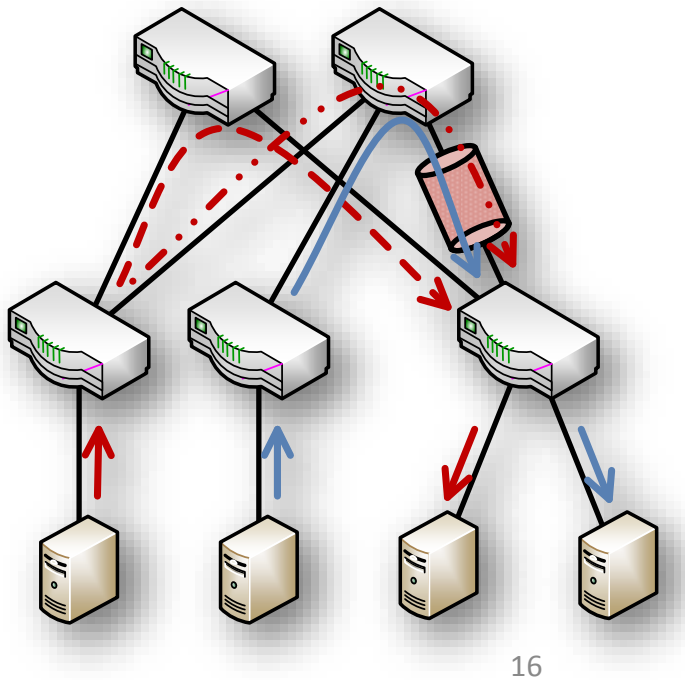
TCP Unfairness



According to the paper,
primary cause of TCP Outcast and low throughput is timeout

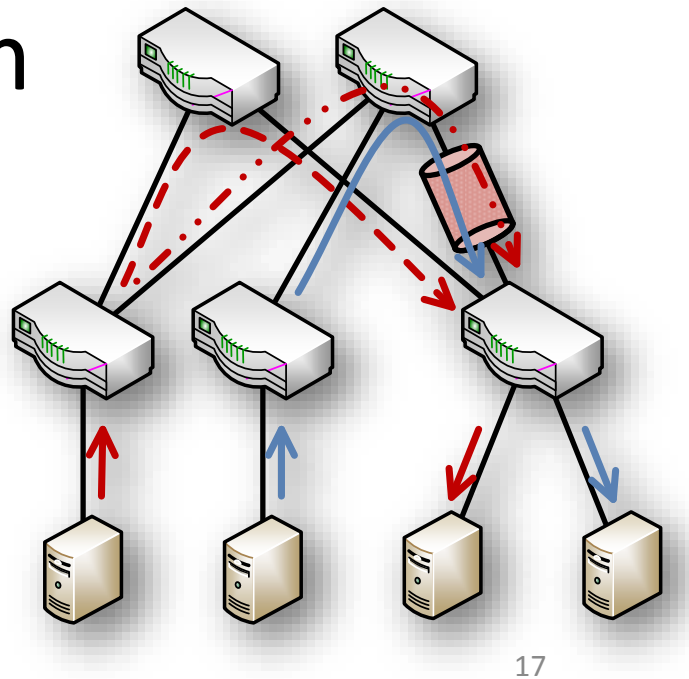
TCP Out-of-order

- Multipath routing techniques
  - Flow-based Splitting Schemes: ECMP, Hedera, VLB
  - Ideal Packet-level scheme is better



TCP Out-of-order

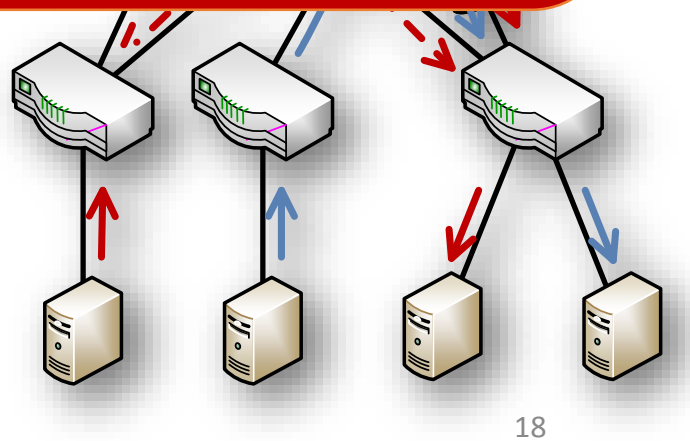
- Multipath routing techniques
  - Flow-based Splitting Schemes: ECMP, Hedera, VLB
  - Ideal Packet-level scheme is better
- TCP Out-of-order Problem
  - Using a fixed threshold
    - Spurious retransmission
    - Sluggish congestion control



TCP Out-of-order

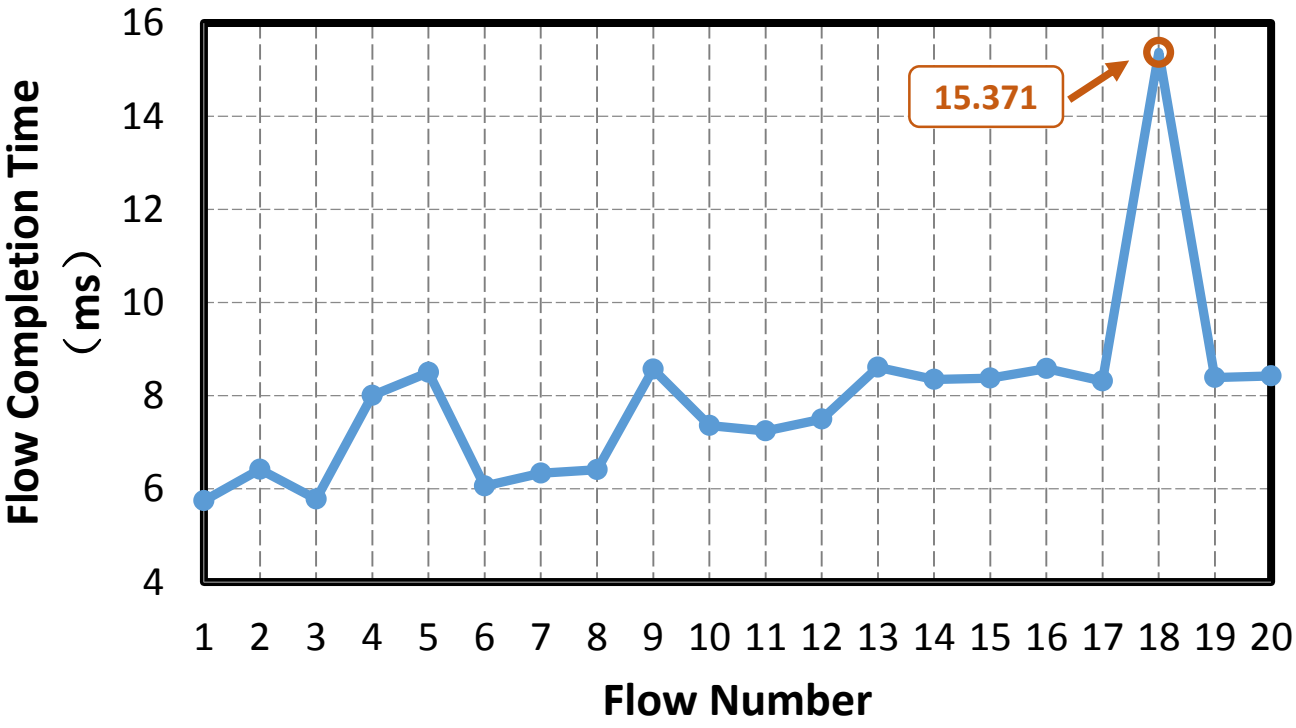
- Multipath routing techniques
  - Flow-based Splitting Schemes: ECMP, Hedera,
  - Spurious retransmission
  - Sluggish congestion control

Accurately distinguishing between lost or out-of-order packets can tackle TCP out-of-order problem



Long Query Completion Time

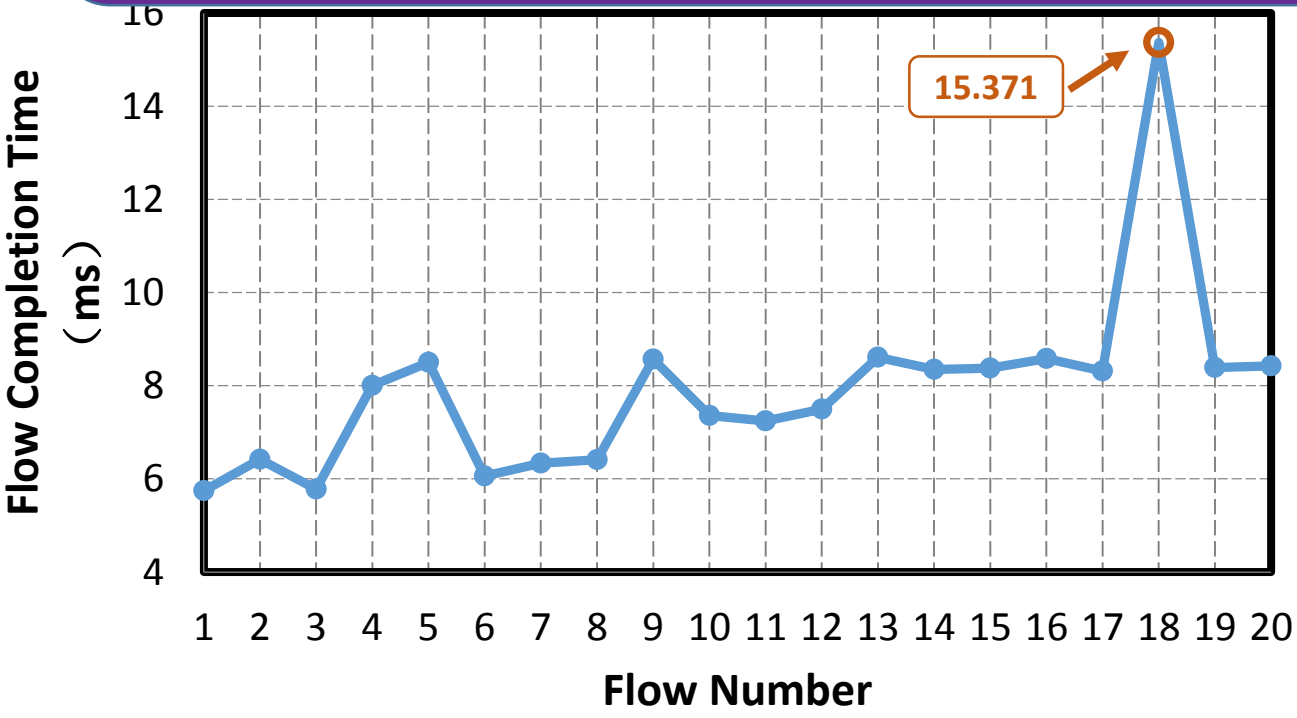
- Each of 20 DCTCP flows transfers 50KB (Avg: 8ms)
- 10,000 random instances
- Finding: 1,231 experiments > 15ms



- Suffer many retransmission and waste time retransmitting

Long Query Completion Time

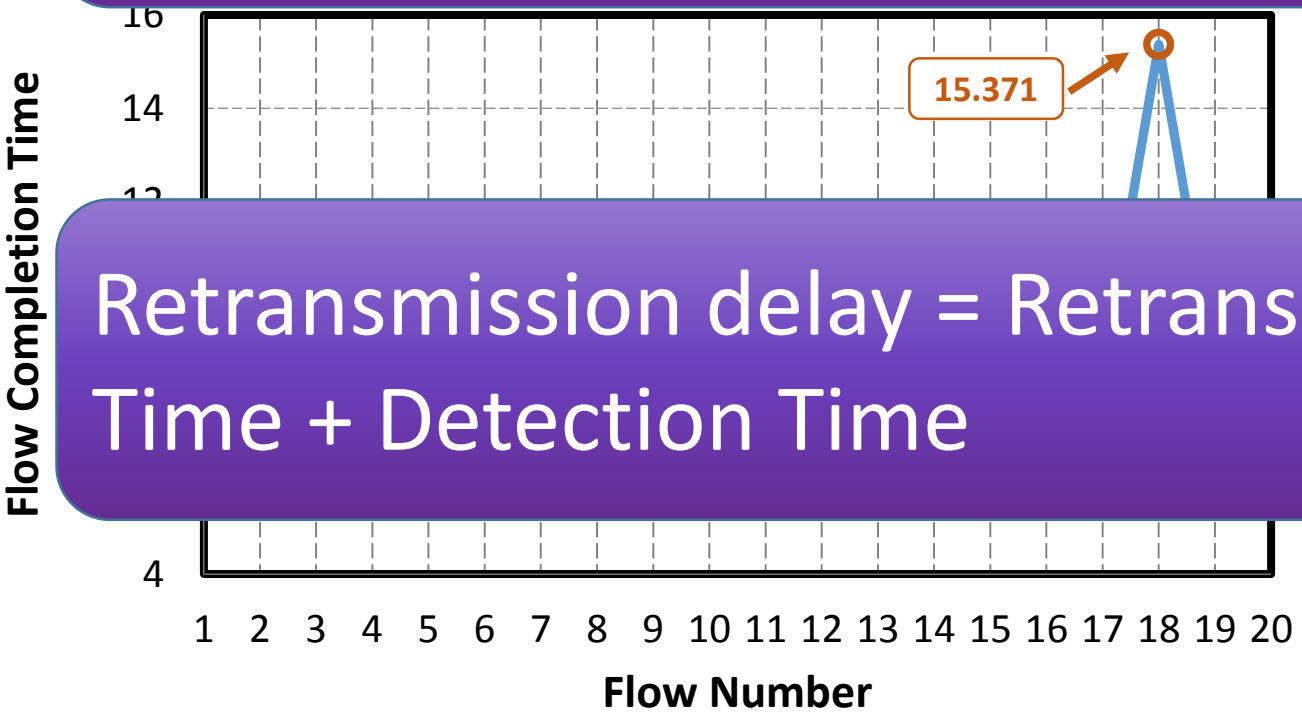
Retransmission delay imposes a significant influence on query completion time



- Suffer many retransmission and waste time retransmitting

Long Query Completion Time

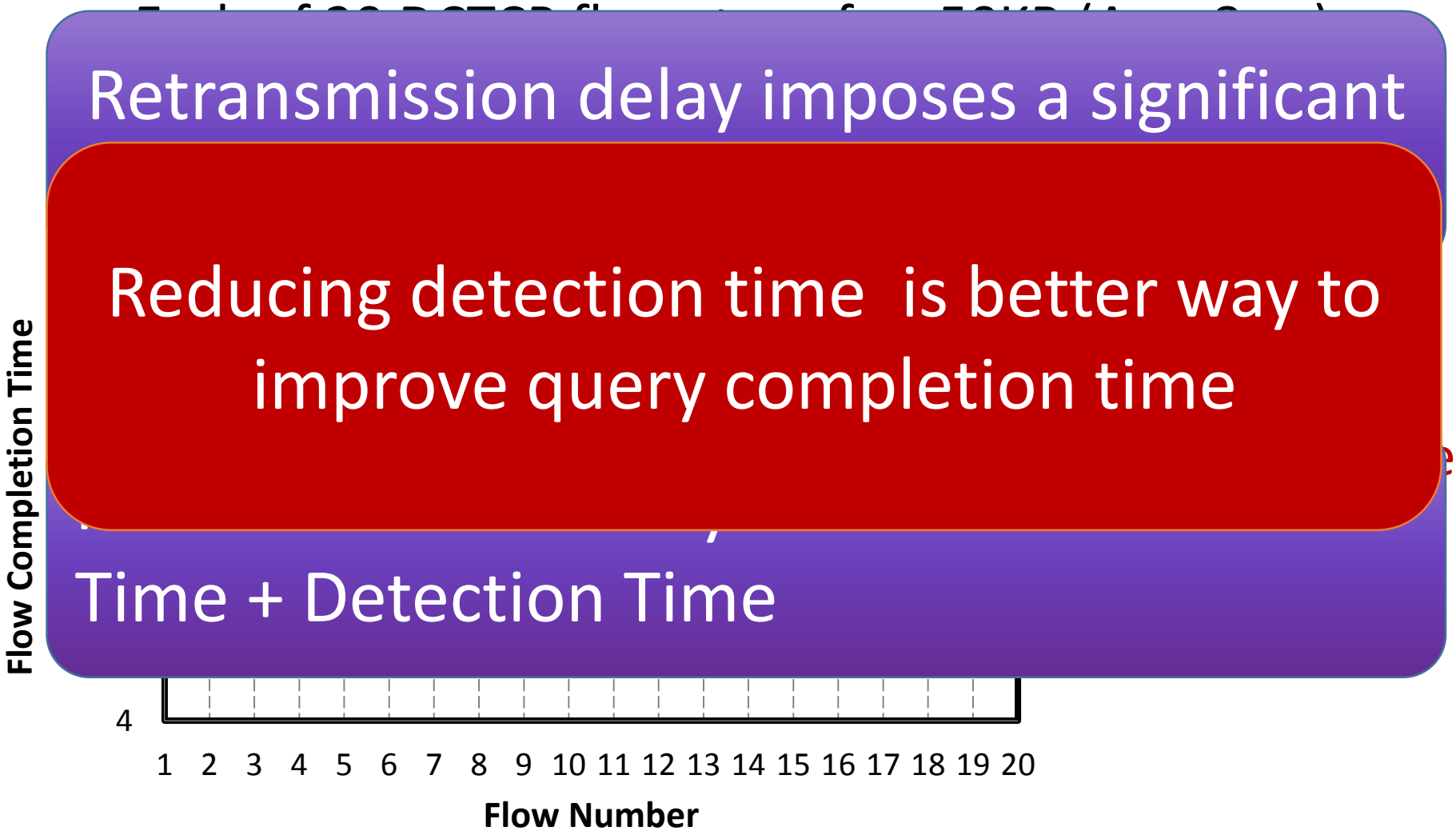
Retransmission delay imposes a significant influence on query completion time



- Suffer many retransmission

Retransmission delay = Retransmission Time + Detection Time

Long Query Completion Time



Our Goal

TCP Incast

TCP Unfairness

TCP Out-of-order

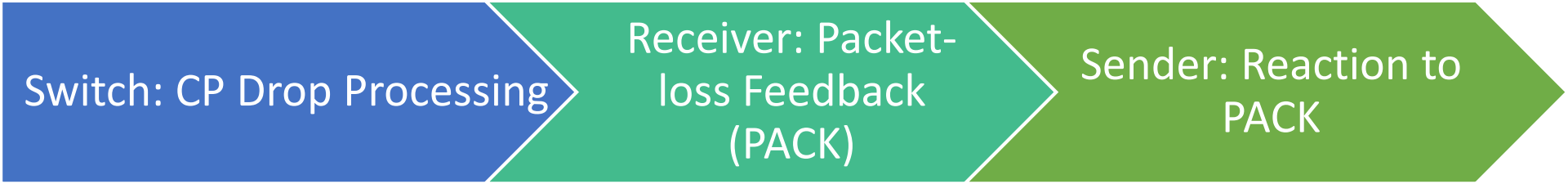
Long Completion Time

Avoid Timeout

Distinguish accurately

Reduce Packet Loss Detection Time

Our Solution: CP

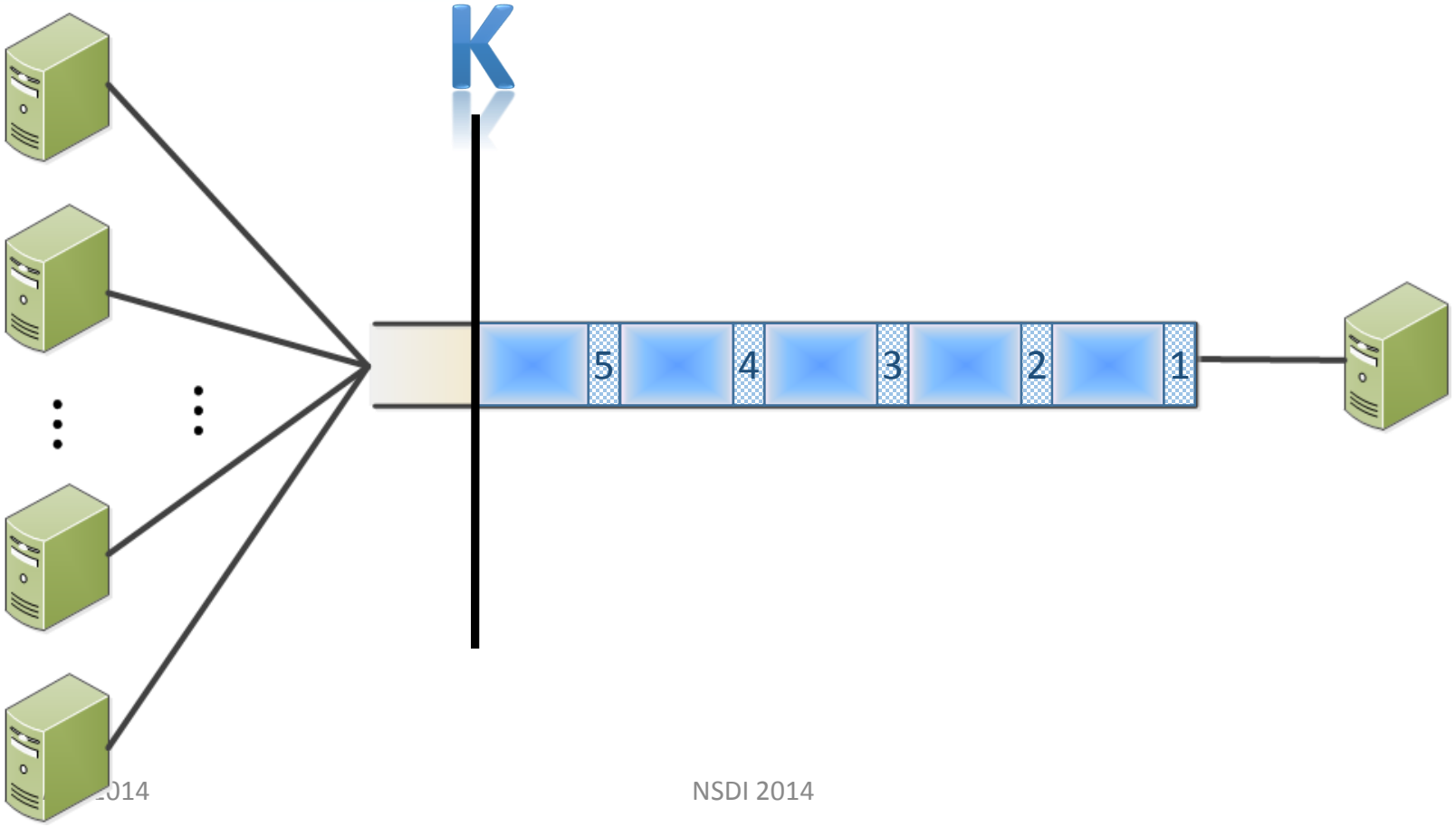


Our Solution: CP

Switch: CP Drop Processing

Receiver: Packet-loss Feedback (PACK)

Sender: Reaction to PACK

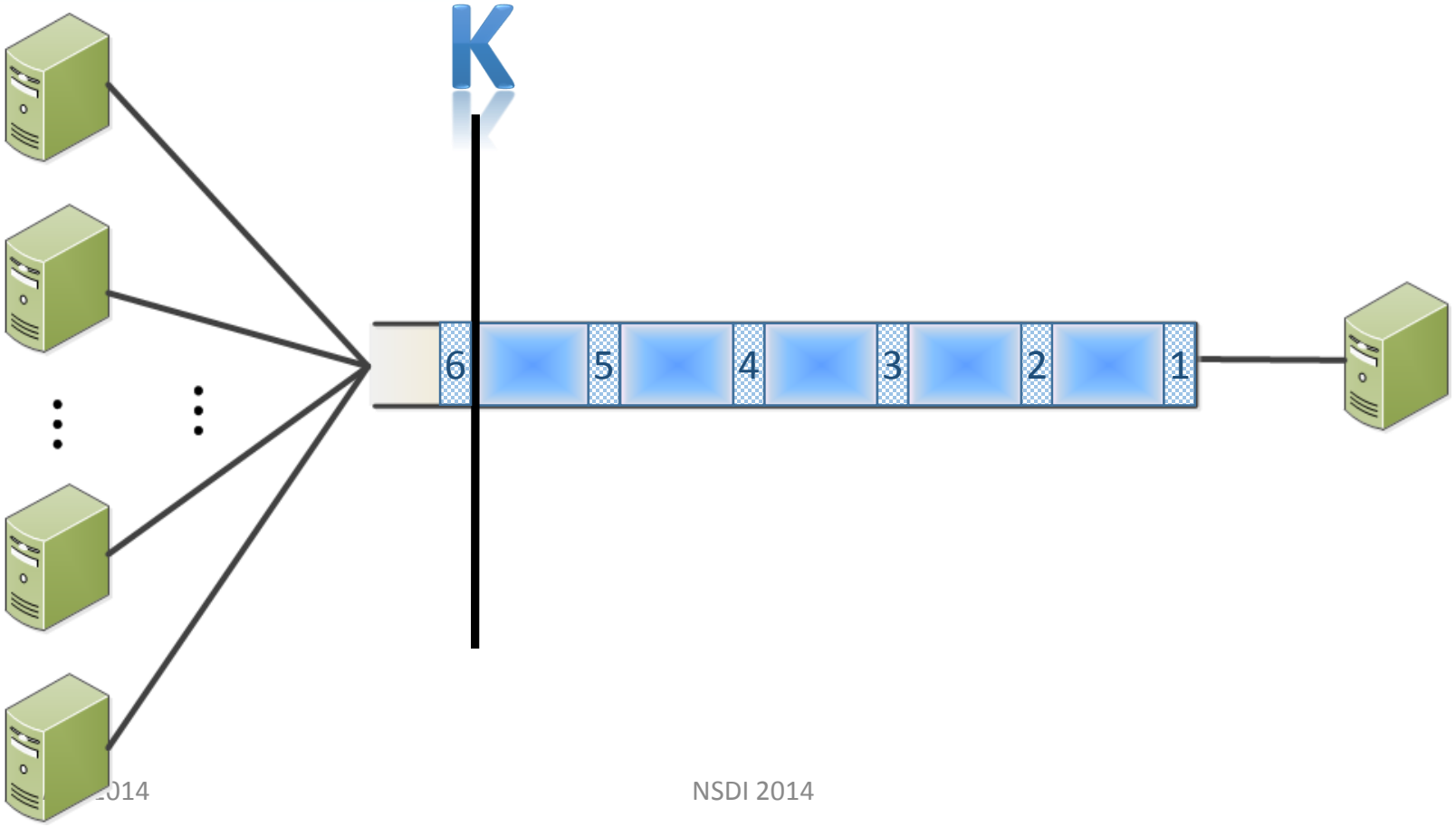


Our Solution: CP

Switch: CP Drop Processing

Receiver: Packet-loss Feedback (PACK)

Sender: Reaction to PACK

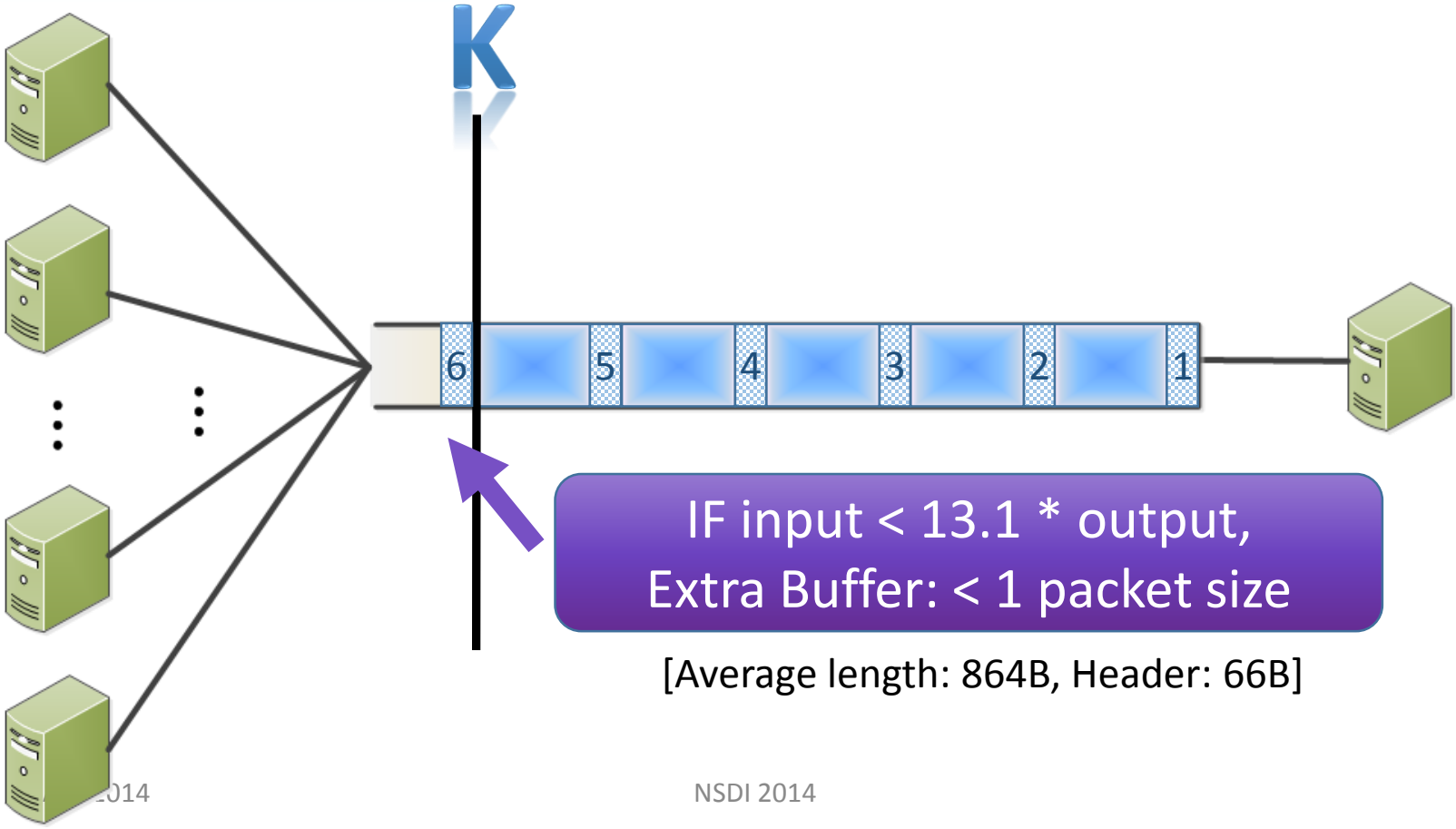


Our Solution: CP

Switch: CP Drop Processing

Receiver: Packet-loss Feedback (PACK)

Sender: Reaction to PACK



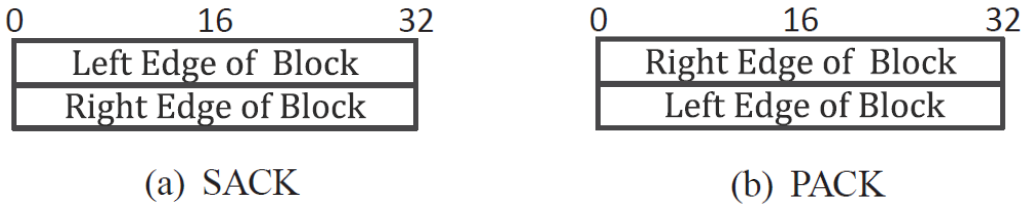
IF input < 13.1 * output,
Extra Buffer: < 1 packet size

[Average length: 864B, Header: 66B]

Our Solution: CP

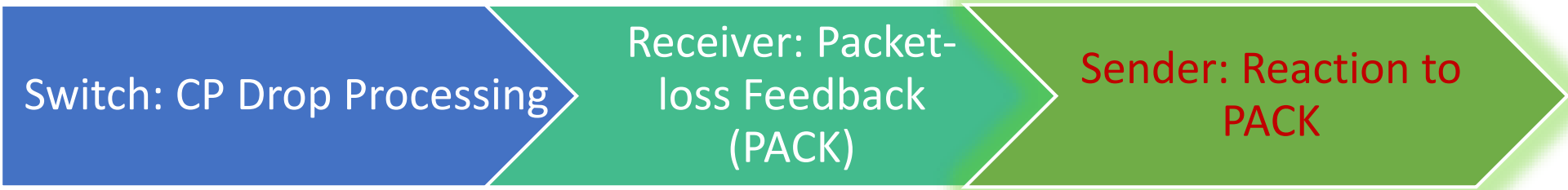
Switch: CP Drop Processing → Receiver: Packet-loss Feedback (PACK) → Sender: Reaction to PACK

- When Receiving the Payload-cut Packet:
  - **Step1:** Parse the payload-cut packet
  - **Step2:** Generate the PACK option
    - Swap the left and right edge of block
    - **Compatible with SACK**



  - **Step3:** Send the PACK immediately

Our Solution: CP



- When Receiving the PACK:
  - **Step1:** Which packet is lost
    - Parse the PACK option
  - **Step2:** Whether to be retransmitted
    - Check the packet status
  - **Step3:** Trigger FRFR and retransmit the lost packet

Benefits

Payload-cut Packet + PACK

(Maintain Self-clocking)

Avoid Timeout

PACK

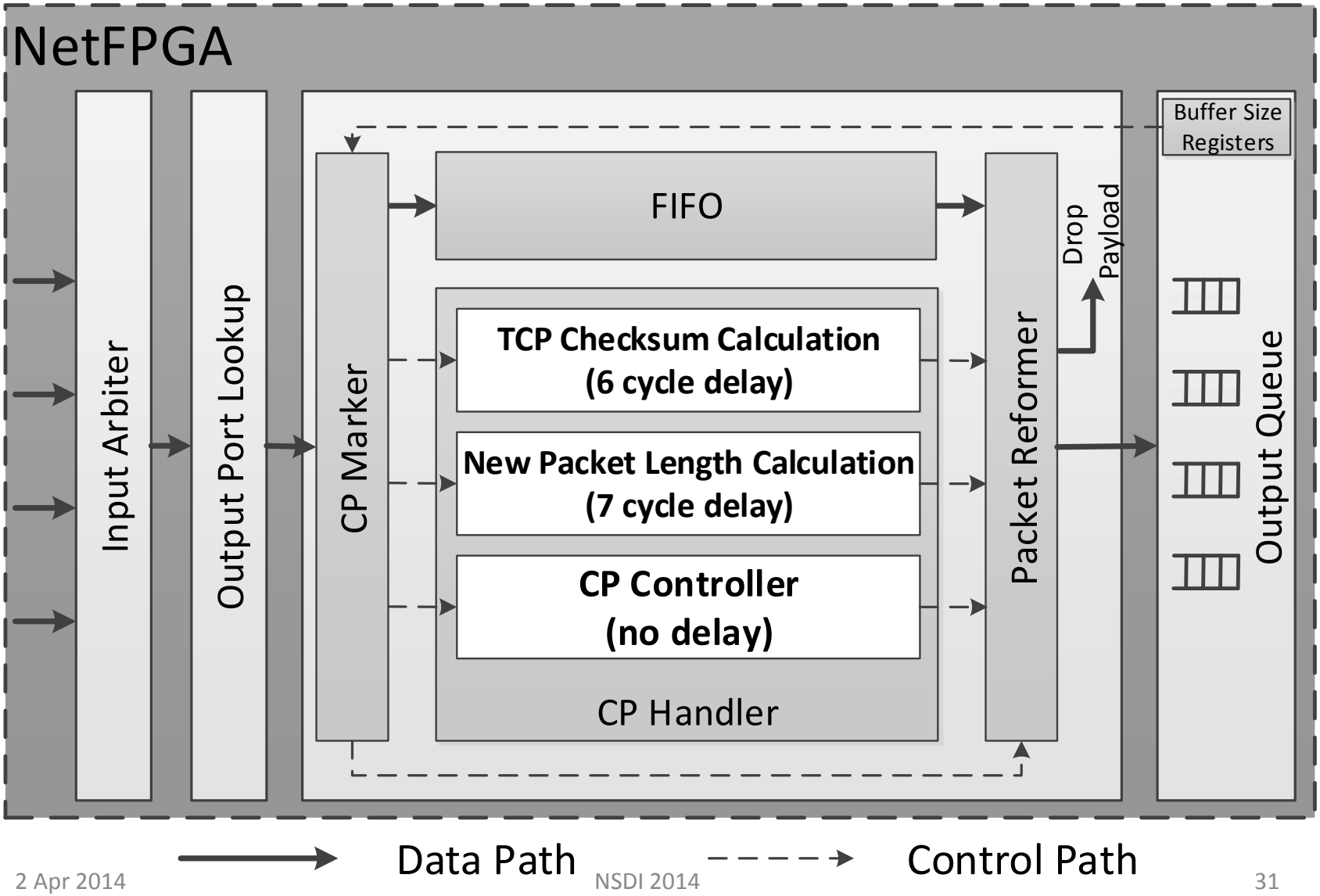
(Rapid Precise

Packet Loss Notification)

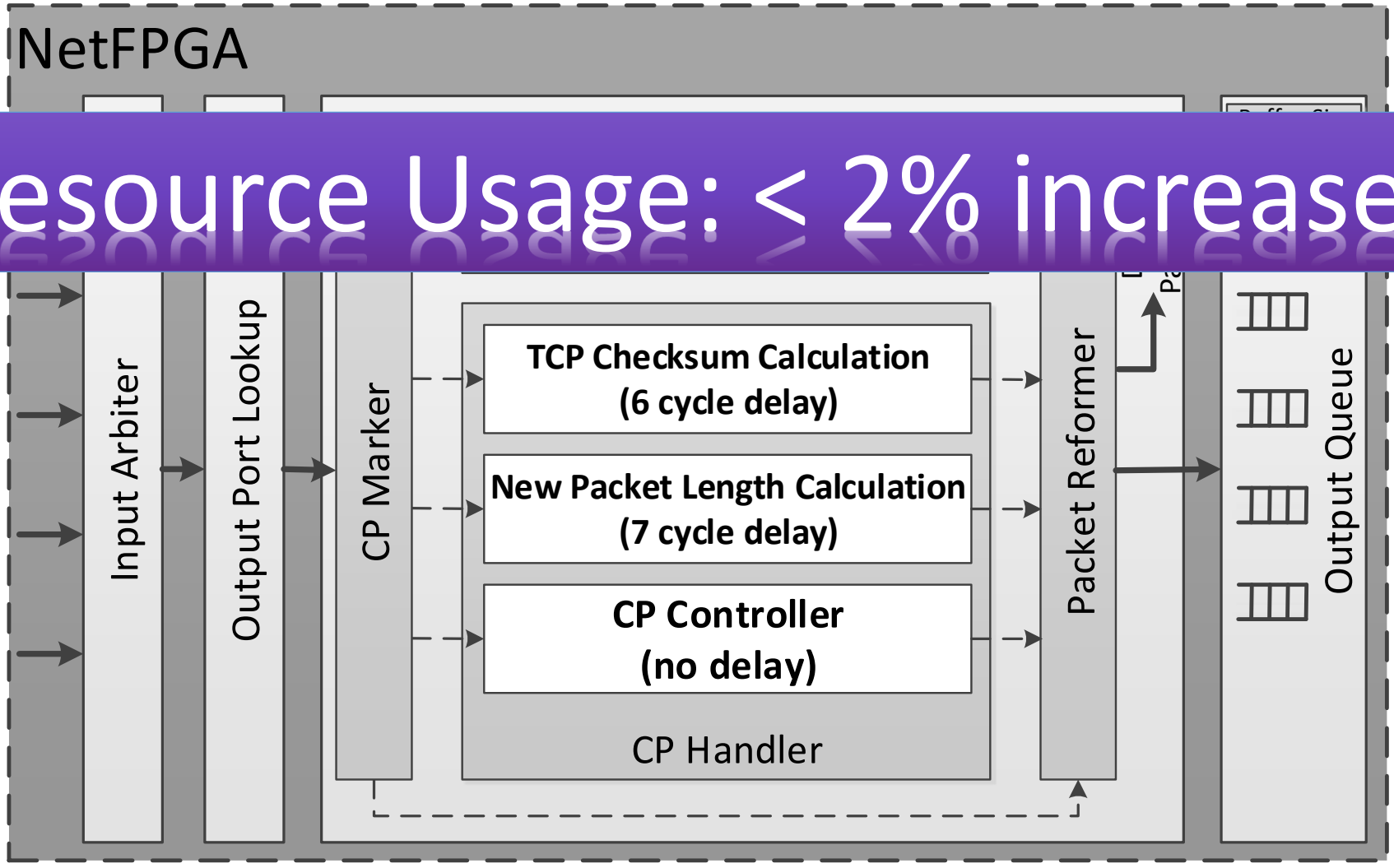
Distinguish accurately

Reduce Packet Loss Detection Time

Implementation

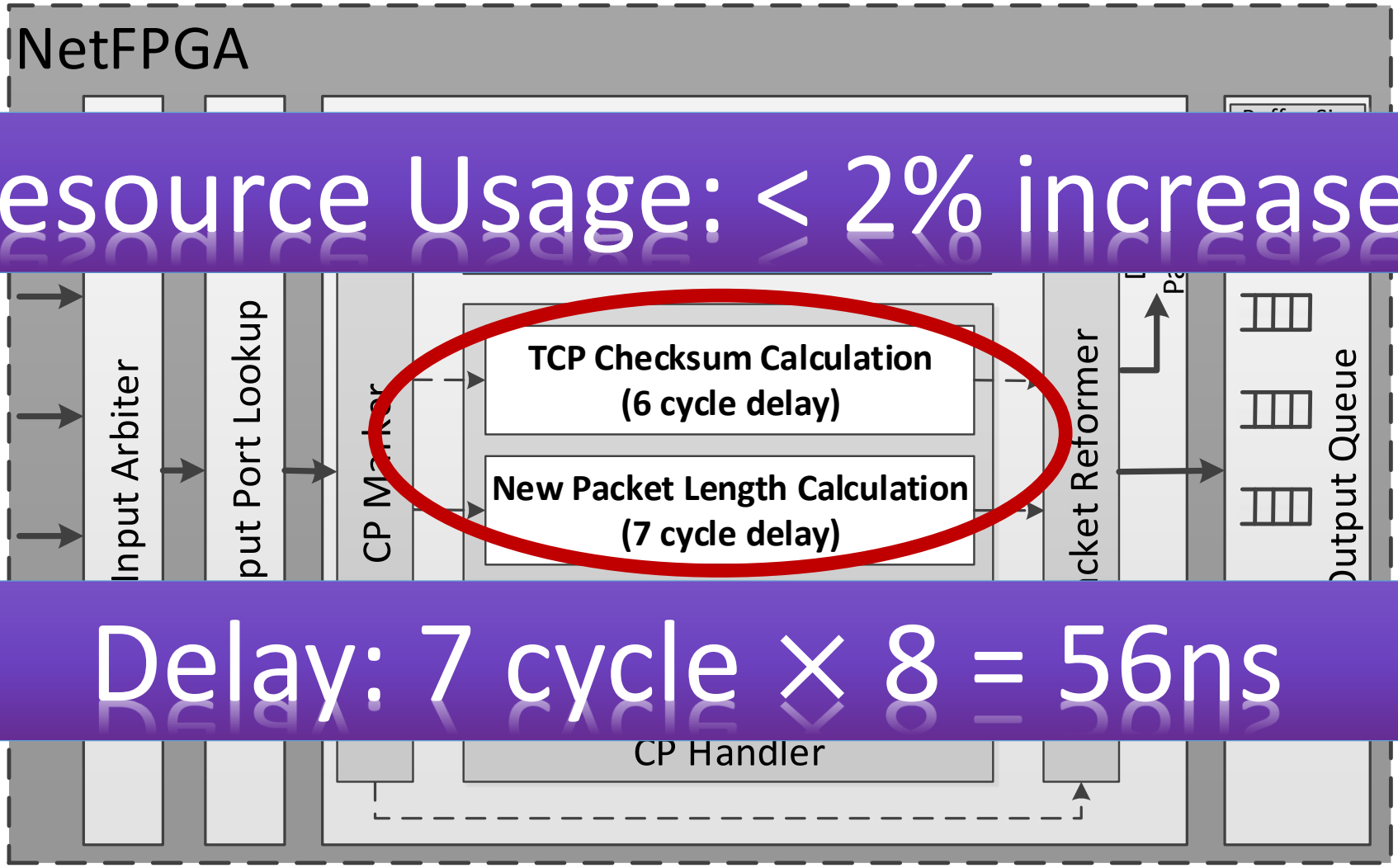


Implementation



Resource Usage: < 2% increase

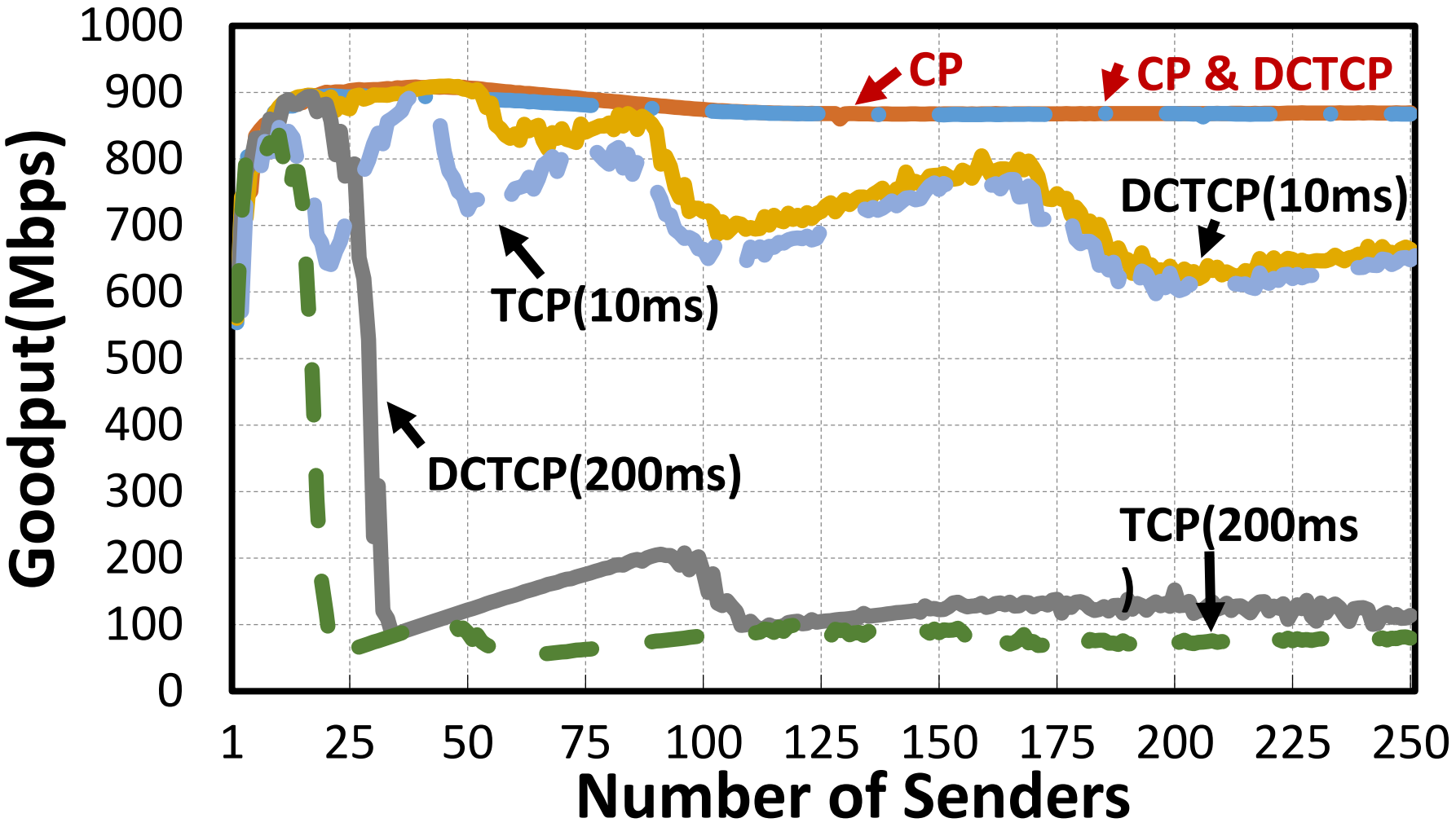
Implementation



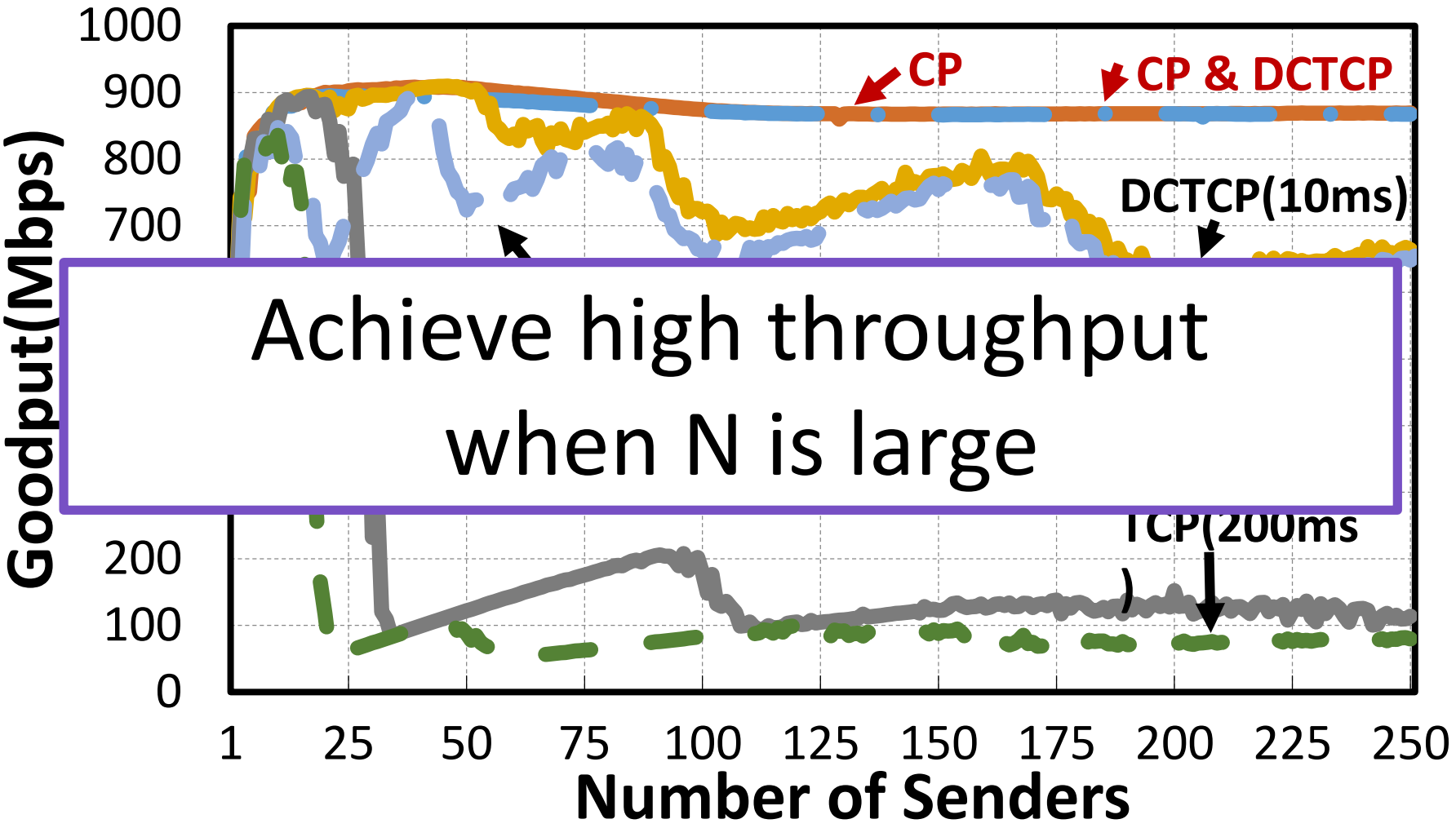
Resource Usage: < 2% increase

Delay: 7 cycle × 8 = 56ns

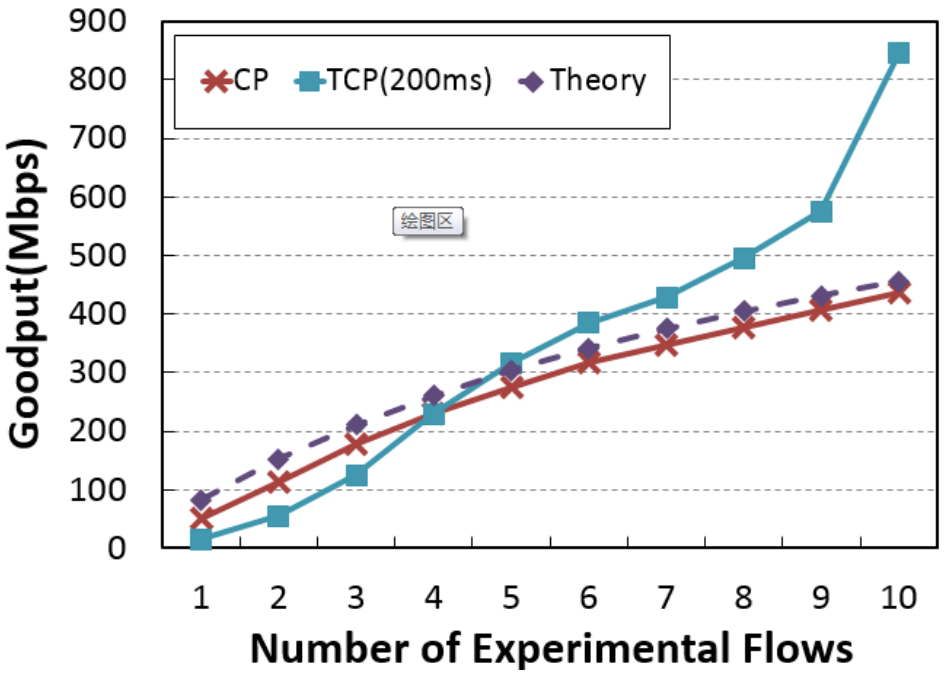
Evaluation: TCP Incast

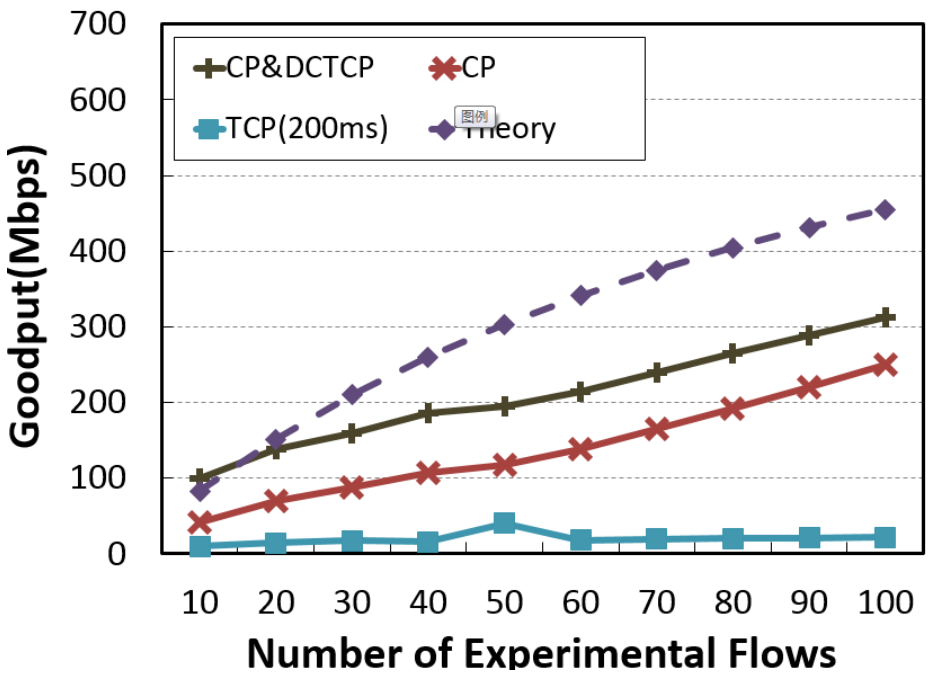


Evaluation: TCP Incast



Evaluation: TCP Unfairness

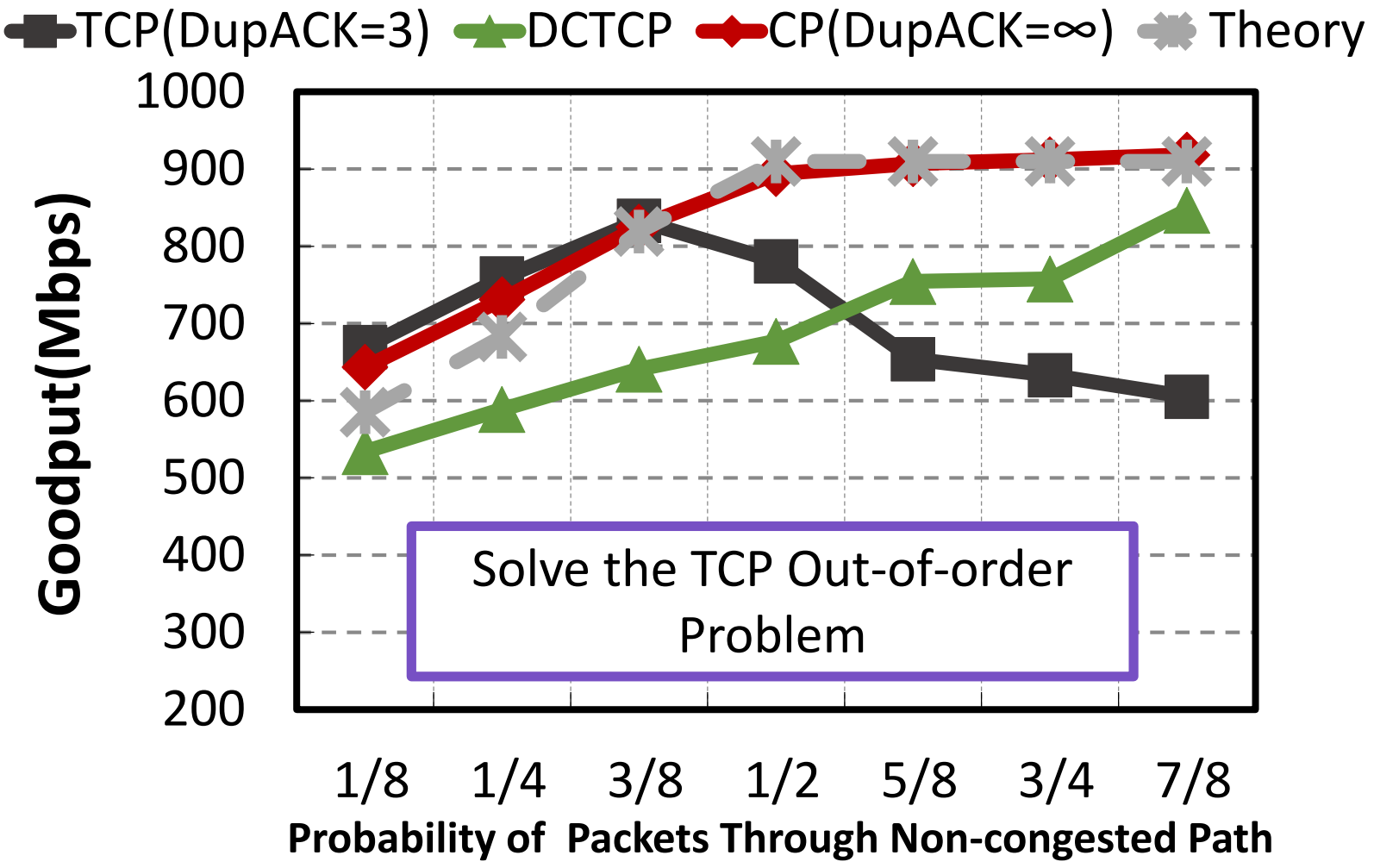




Low Packet Loss Ratio:
Avoid TCP Outcast

High Packet Loss Ratio:
Ease Low Throughput

Evaluation: TCP Out-of-order Problem



Evaluation: Query Completion Time

		DCTCP	TCP
Realistic Traffic (384KB)	99th	10.2ms→5.7ms (44.34%)	11.3ms→7.0ms (38.09%)
	99.9th	36.5ms→34.1ms (6.58%)	47.7ms→42.3ms (11.40%)
10X Traffic	99th	31ms→25ms (18.53%)	117ms→75ms (35.71%)
	99.9th	52ms→39ms (26.21%)	482ms→113ms (76.55%)

Conclusion

- CP
  - is a TCP-complementary method
  - can solve many TCP performance issue
  - is compatible with other TCP protocol in DCNs
- Source Code Release
  - Linux Patch + NetFPGA implementation
  - http://qos.cs.tsinghua.edu.cn/~chengp/CP/index.html

Thank you!