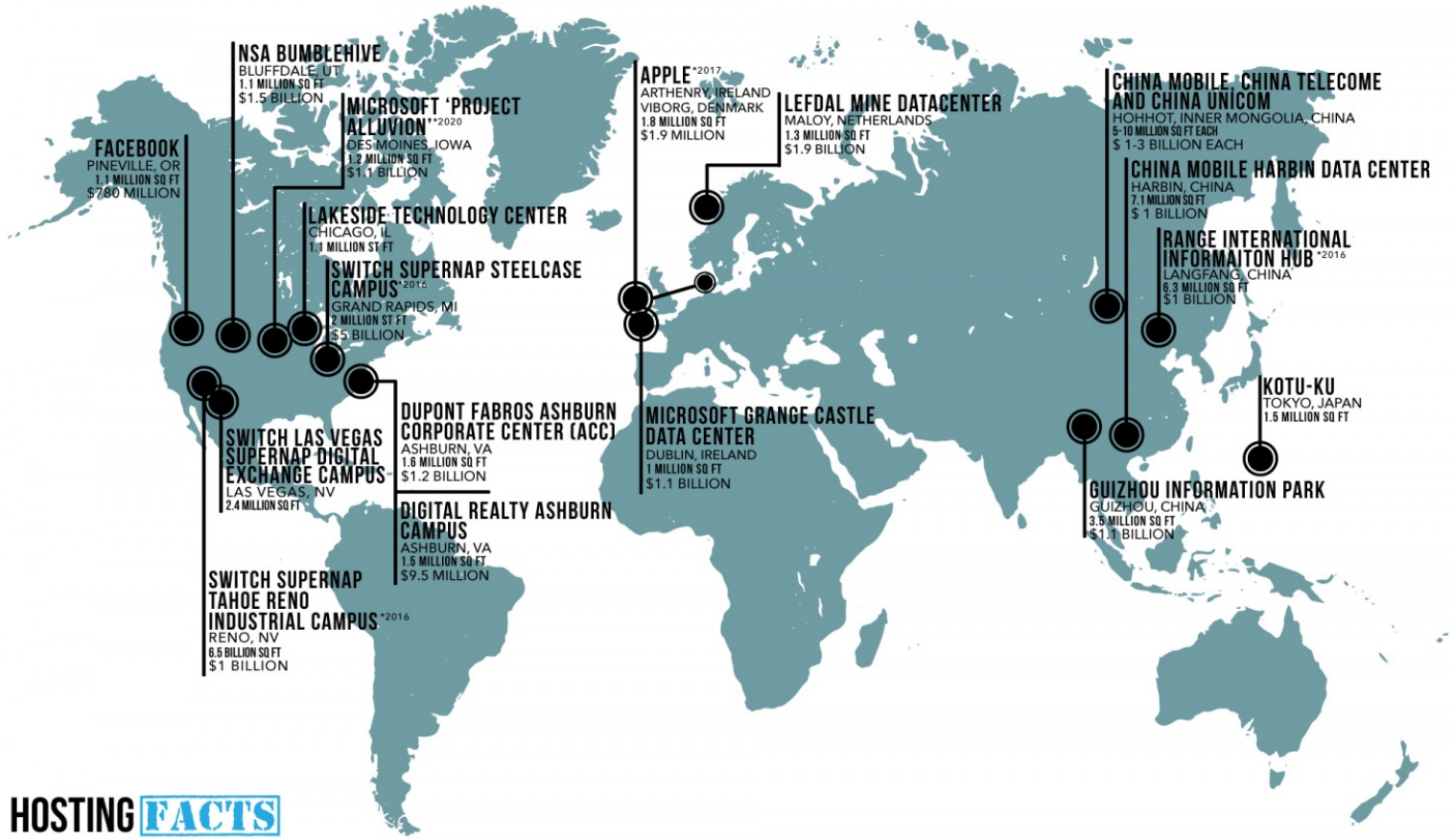
Enabling ECN for Datacenter Networks with RTT Variations

Junxue ZHANG, Wei BAI (Microsoft Research), Kai CHEN

SING Group@Hong Kong University of Science and Technology

SingLab HKUST | 香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

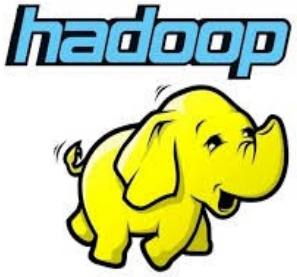
Datacenters Around the World



Largest datacenters in the world

Inside the Datacenters

- Network requirements of applications
  - Desire low latency for short messages
  - Desire high throughput for large messages
  - Desire good burstiness tolerance to avoid frequent packet drops



Inside the Datacenters

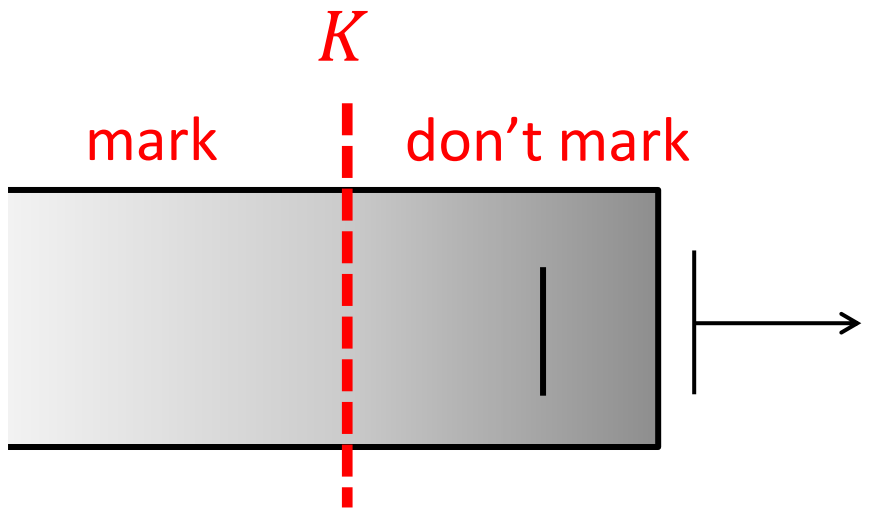
- Network requirements of applications
  - Desire low latency for short messages
  - Desire high throughput for large messages
  - Desire good burstiness tolerance to avoid frequent packet drops

Meet the requirements

- ECN-based Transports
  - Achieve low latency & high throughput simultaneously
  - Achieve good burstiness tolerance

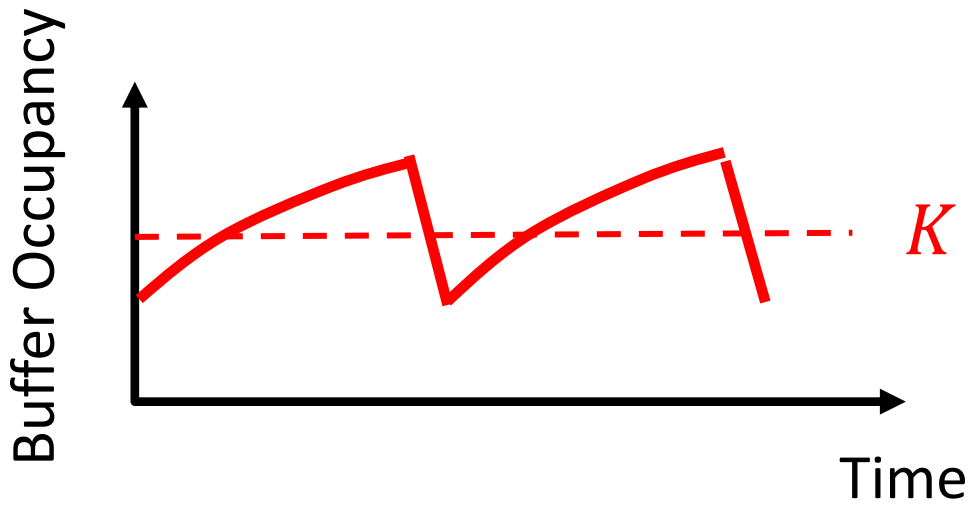
ECN-based Transports

- Packets get marked when queue length $L > K$
- Instantaneous queue length is used to allow good burstiness control



ECN-based Transports

- Packets get marked when queue length $L > K$
- Instantaneous queue length is used to allow good burstiness control
- K should be carefully chosen to achieve both low latency & high throughput



ECN-based Transports

- Packets get marked when queue length $L > K$
- Instantaneous queue length is used to allow good burstiness control
- K should be carefully chosen to achieve both low latency & high throughput
- To achieve this, we set K as follows:

$$K = \lambda \times C \times RTT$$

DCTCP (SIGCOMM '10), TCN (CoNEXT '16)

ECN-based Transports

- Packets get marked when queue length $L > K$
- Instantaneous queue length is used to allow good burstiness control
- K should be carefully chosen to achieve both low latency & high throughput
- To achieve this, we set K as follows:

$$K = \lambda \times C \times RTT$$

Determined by Congestion Control Algorithm
e.g., $\lambda = 1$ with regular ECN-based TCP

ECN-based Transports

- Packets get marked when queue length $L > K$
- Instantaneous queue length is used to allow good burstiness control
- K should be carefully chosen to achieve both low latency & high throughput
- To achieve this, we set K as follows:

$$K = \lambda \times C \times RTT$$

Fixed link capacity

ECN-based Transports

- Packets get marked when queue length $L > K$
- Instantaneous queue length is used to allow good burstiness control
- K should be carefully chosen to achieve both low latency & high throughput
- To achieve this, we set K as follows:

$$K = \lambda \times C \times RTT$$

Base RTT is stable inside data centers ? NO
DCTCP (SIGCOMM '10), ECN*(CoNEXT '12), …

RTT Variation inside Data Centers

- Base RTT is composed of:
  - Transmission delay
  - Propagation delay
  - Processing delay

RTT Variation inside Data Centers

- Base RTT is composed of:
  - Transmission delay
  - Propagation delay
  - Processing delay

Transmission delay is small due to high link capacity
e.g., for a 1.4KB packet, the delay is 1.4μs when link capacity is 10Gbps

RTT Variation inside Data Centers

- Base RTT is composed of:
  - Transmission delay
  - Propagation delay
  - Processing delay

Propagation delay is small due to short cable length inside DC
e.g. The delay of a 1KM cable is only 3.3 μs.

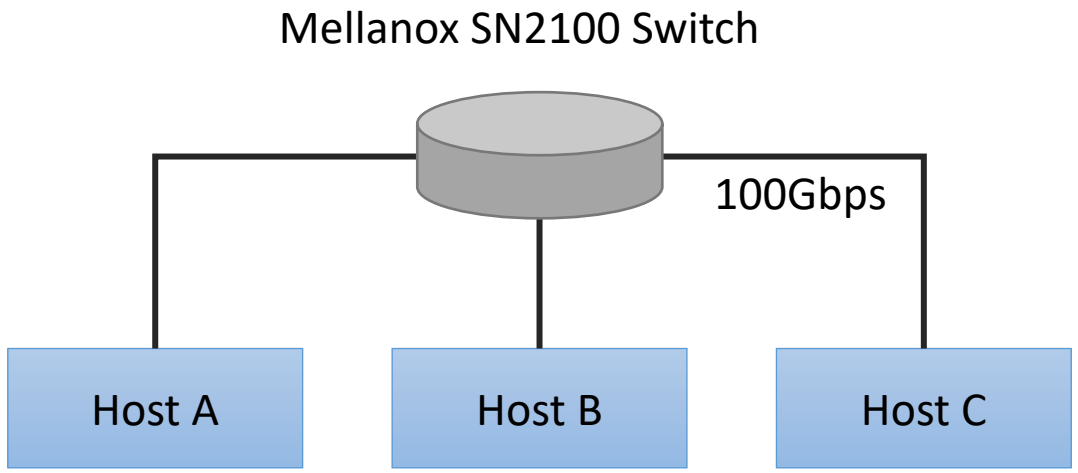
RTT Variation inside Data Centers

- Base RTT is composed of:
  - Transmission delay
  - Propagation delay
  - Processing delay

The processing delay has large variation up to 100 μs or even more caused by Kernel Scheduling, Middlebox, Hypervisor, …
Ananta(SIGCOMM'13), Duet(SIGCOMM'14), …

RTT Variation Caused by Processing Delay

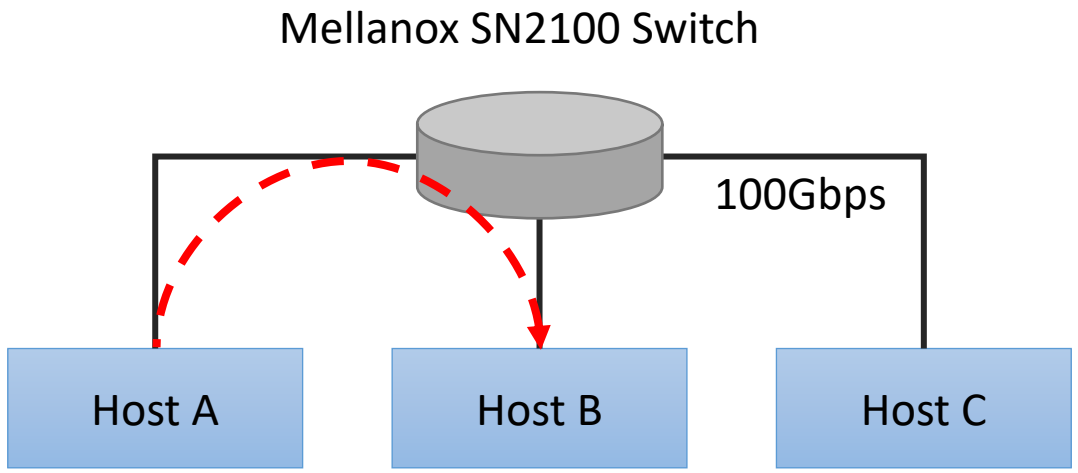
- Testbed Settings
  - 3 Servers are connected to a Mellanox SN2100 switch
  - Links are 100Gbps
  - We use DCTCP on each host



RTT Variation Caused by Processing Delay

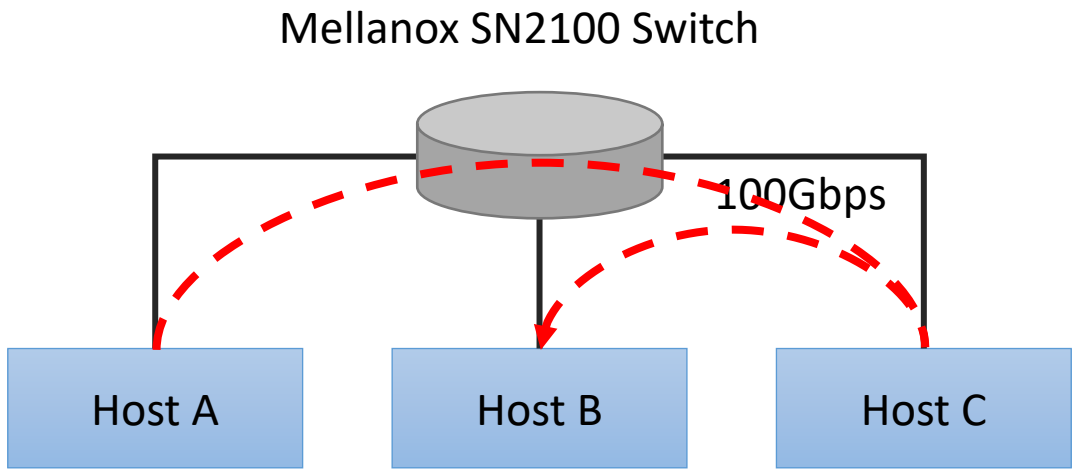
- Case 1: Network Stack
  - Host A is installed with Apache Server
  - Host B uses ApacheBench to fetch webpage from Host A
  - We use TCP PROBE on Host B to probe the RTT

#	Mean	STD	90 Percentile	99 Percentile
1	39.3	12.2	59.0	79.0



RTT Variation Caused by Processing Delay

- Case 2: Network Stack + SLB
  - Host C is installed with Linux Virtual Server (LVS) as a Software Load Balancer (SLB)



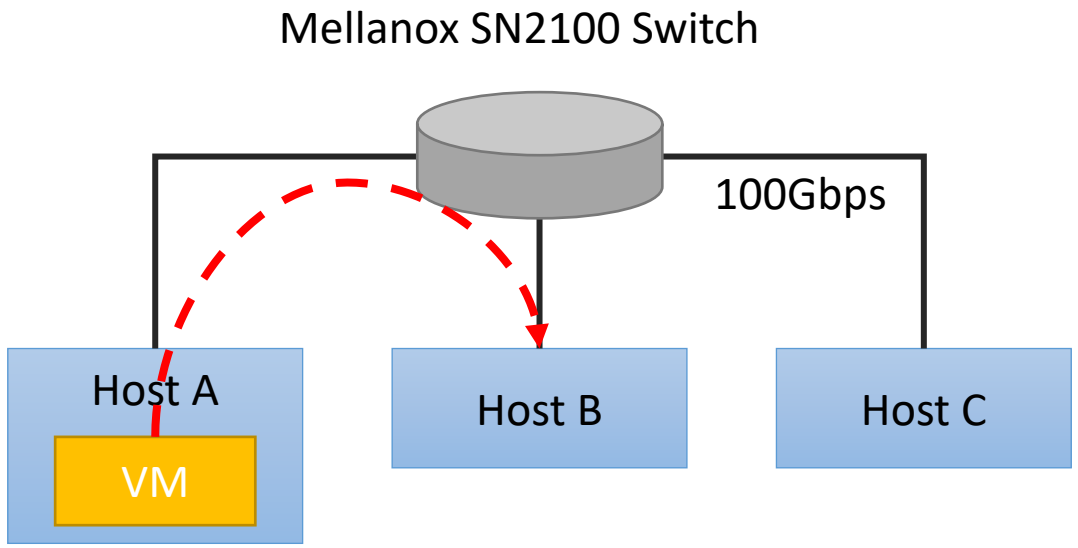
#	Mean	STD	90 Percentile	99 Percentile
1	39.3	12.2	59.0	79.0
2	63.9	18.3	87.0	121.0

Increased by 30μs

RTT Variation Caused by Processing Delay

- Case 3: Network Stack + Hypervisor
  - Host A is installed with KVM
  - A quad-core virtual machine (VM) launches on Host A
  - The VM is installed with Apache Server

#	Mean	STD	90 Percentile	99 Percentile
1	39.3	12.2	59.0	79.0
2	63.9	18.3	87.0	121.0
3	69.3	18.8	91.0	130.0

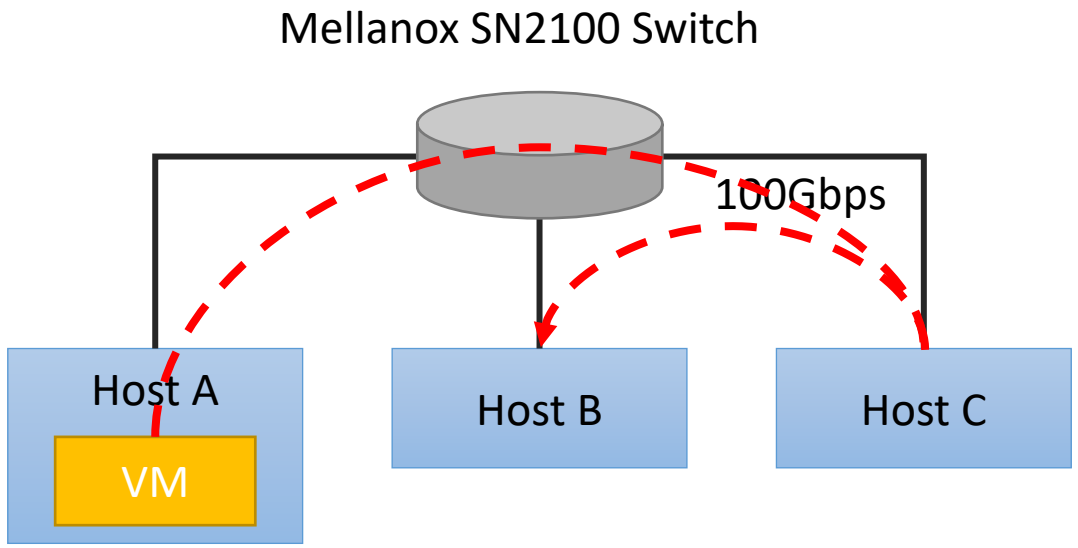


RTT Variation Caused by Processing Delay

- Case 4: Network Stack + SLB + Hypervisor

#	Mean	STD	90 Percentile	99 Percentile
1	39.3	12.2	59.0	79.0
2	63.9	18.3	87.0	121.0
3	69.3	18.8	91.0	130.0
4	99.2	23.0	129.0	161.0

Almost 1.5X



RTT Variation Caused by Processing Delay

- RTT Variations are mainly caused by varying processing delay
  - Spatial: Flows may traverse through different networking components, e.g. middlebox, hypervisor, resulting in different RTTs.
  - Temporal: Different components may add varying delay at different time due to changing workload
  - Testbed is with very simple settings

#	Mean	STD	90 Percentile	99 Percentile
1	39.3	12.2	59.0	79.0
2	63.9	18.3	87.0	121.0
3	69.3	18.8	91.0	130.0
4	99.2	23.0	129.0	161.0

RTT Variation Caused by Processing Delay

- RTT Variations are mainly caused by varying processing delay
  - Spatial: Flows may traverse through different networking components, e.g. middlebox, hypervisor, resulting in different RTTs.
  - Temporal: Different components may add varying delay at different time due to changing workload
  - Testbed is with very simple settings

How to calculate the threshold

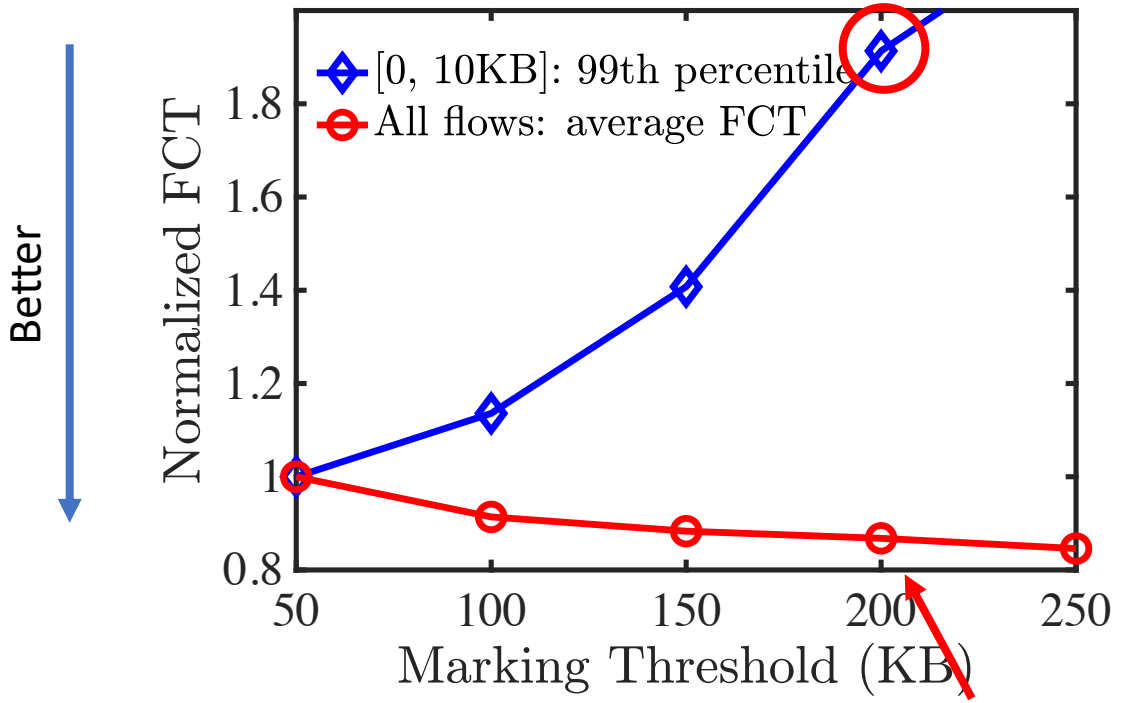
$$K = \lambda \times C \times RTT$$

Current practice is to use high percentile RTT to derive the threshold

ECN*(CoNEXT '12), …

Performance Degradation under RTT Variations

All results are normalized to results achieved by threshold of 50KB

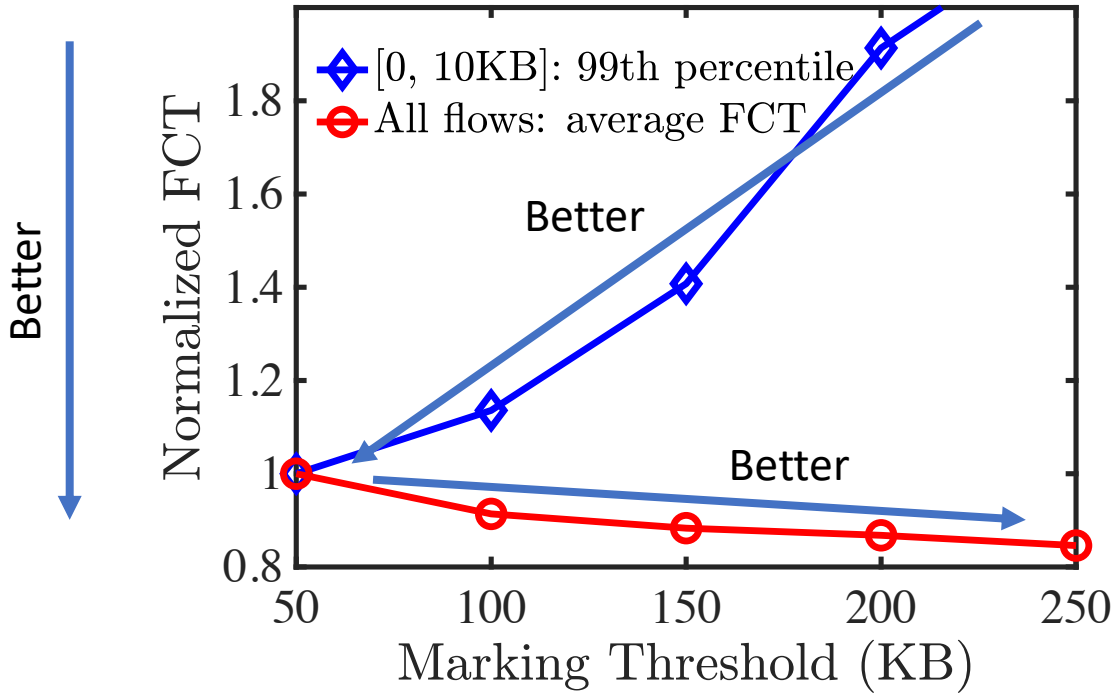


Performance degradation for short flows

Current Practice, the threshold is derived based on high percentile RTT

Performance Degradation under RTT Variations

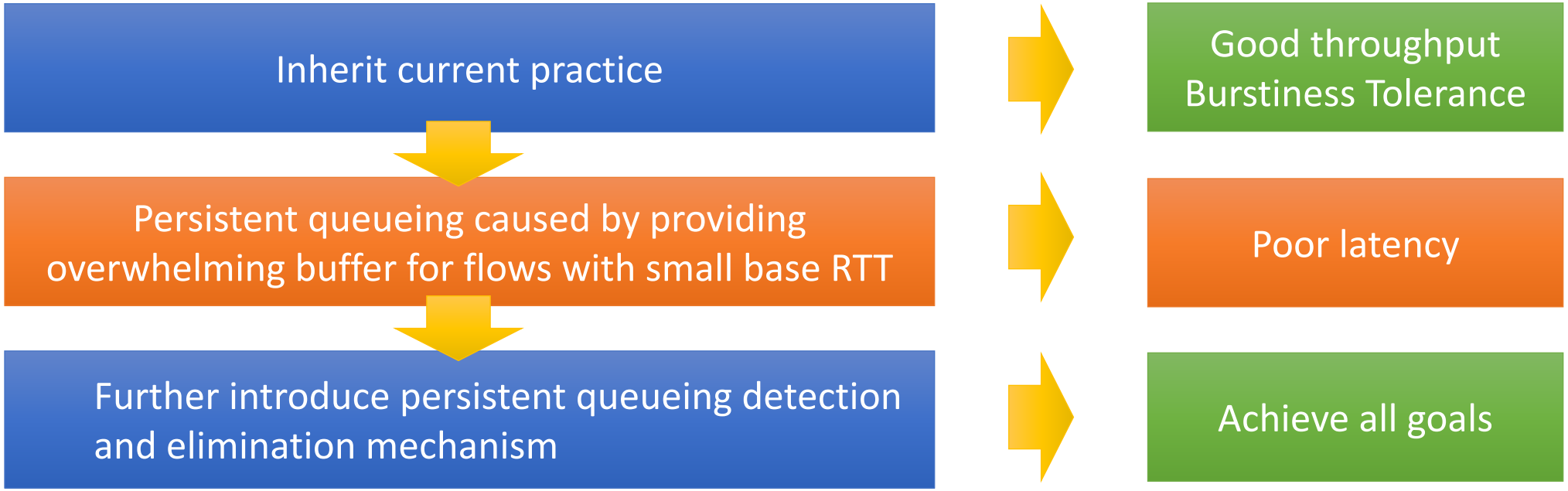
All results are normalized to results achieved by threshold of 50KB



Observation: Setting threshold based on high/low percentile results in either unacceptable throughput loss or long latency when RTT variations exist

ECN#

- ECN# is simple yet effective

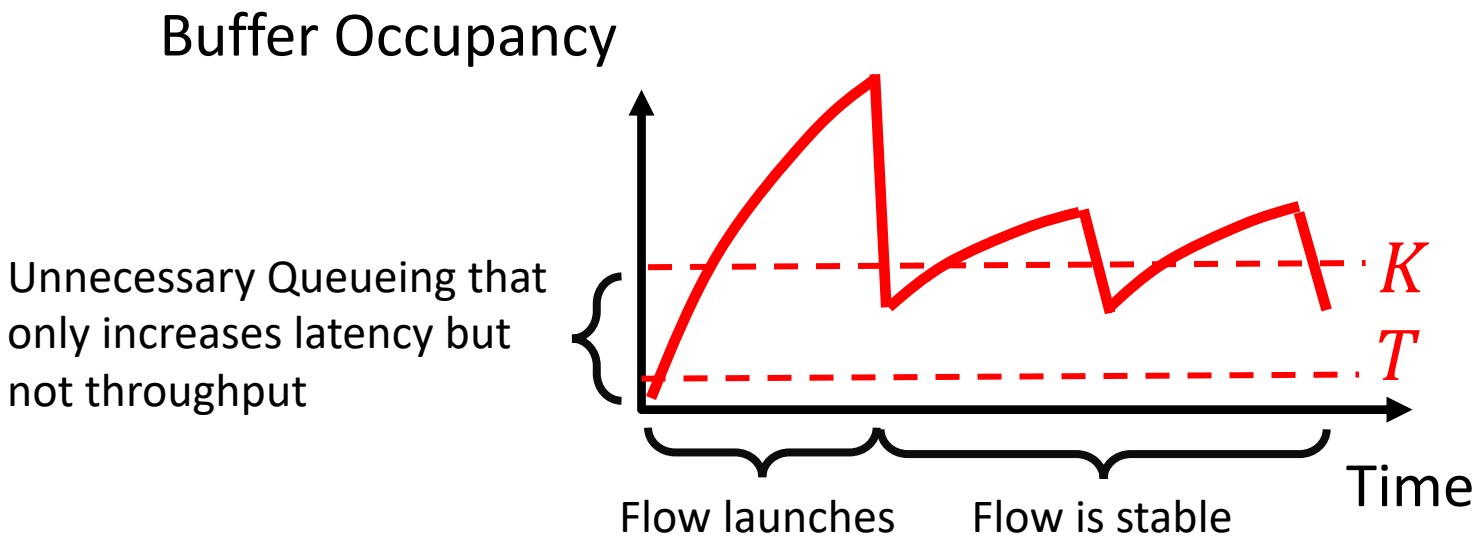


ECN$^{\#}$ in Details

- ECN marking based on instantaneous queueing
  - Marks if instantaneous queueing $L > K$
  - K is derived based on high percentile RTT
  - Two advantages:
    - Not hurt throughput
    - Good burstiness control

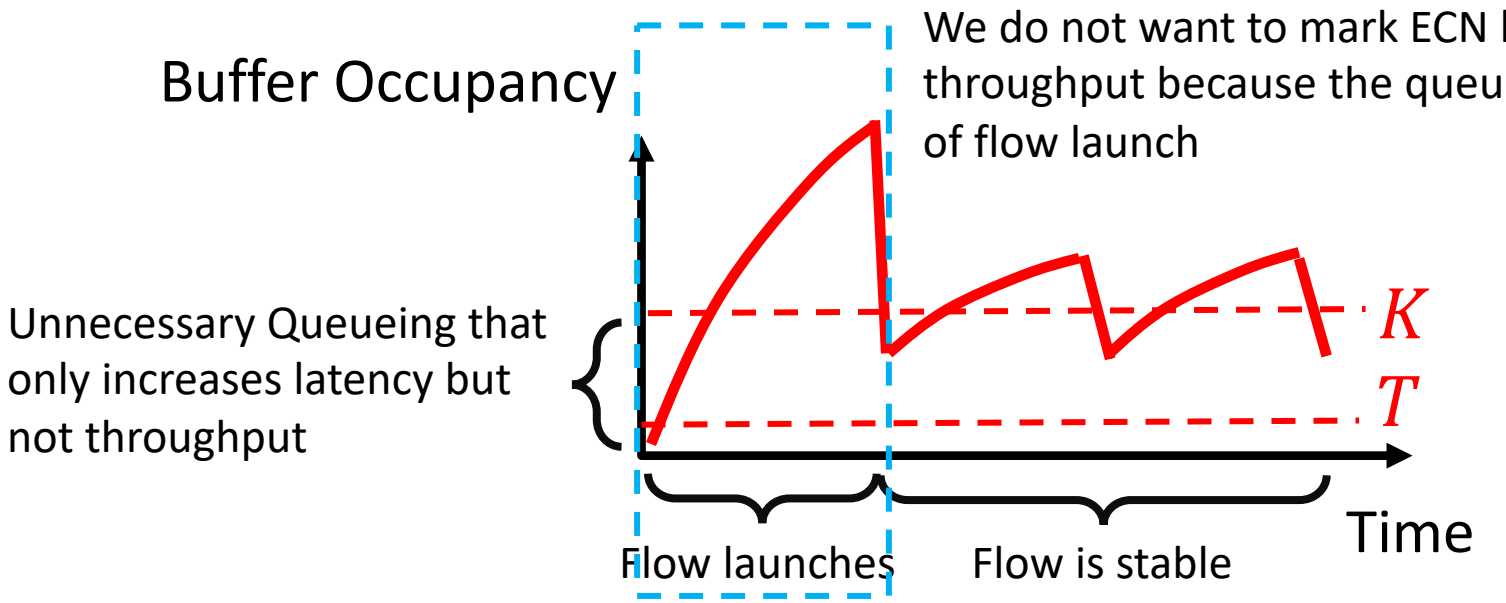
ECN# in Details

- ECN marking based on instantaneous queueing
- ECN marking based on persistent queueing
  - Compare the minimal queueing over an interval I with threshold T



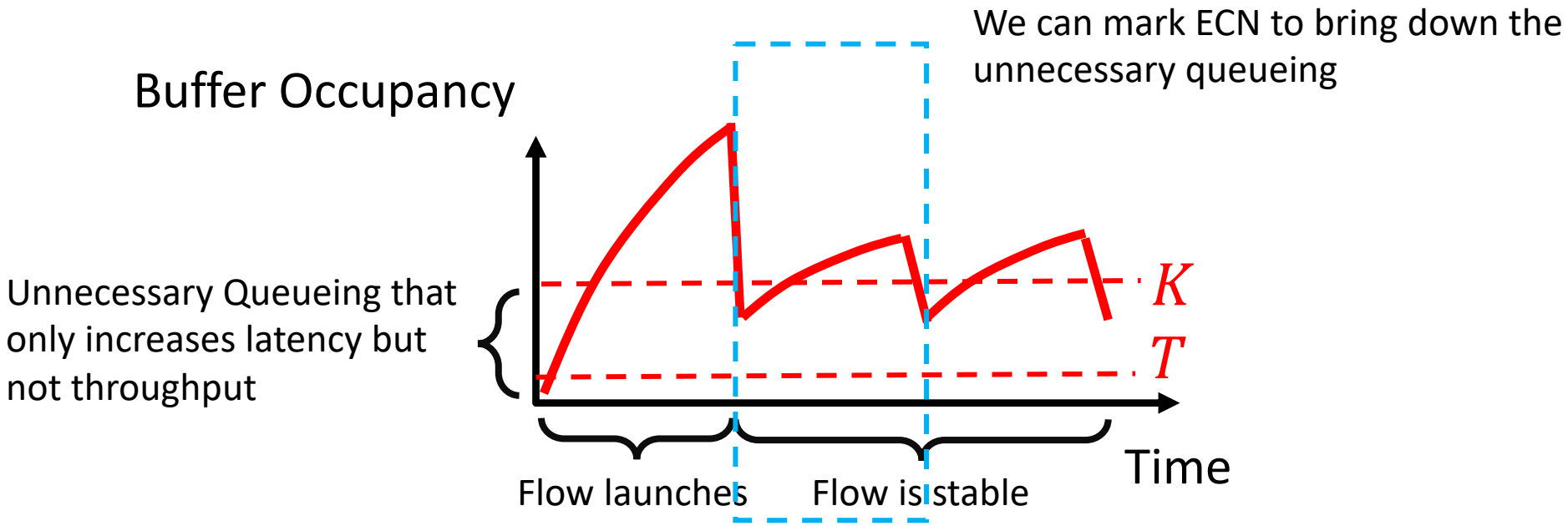
ECN# in Details

- ECN marking based on instantaneous queueing
- ECN marking based on persistent queueing
  - Compare the minimal queueing over an interval I with threshold T



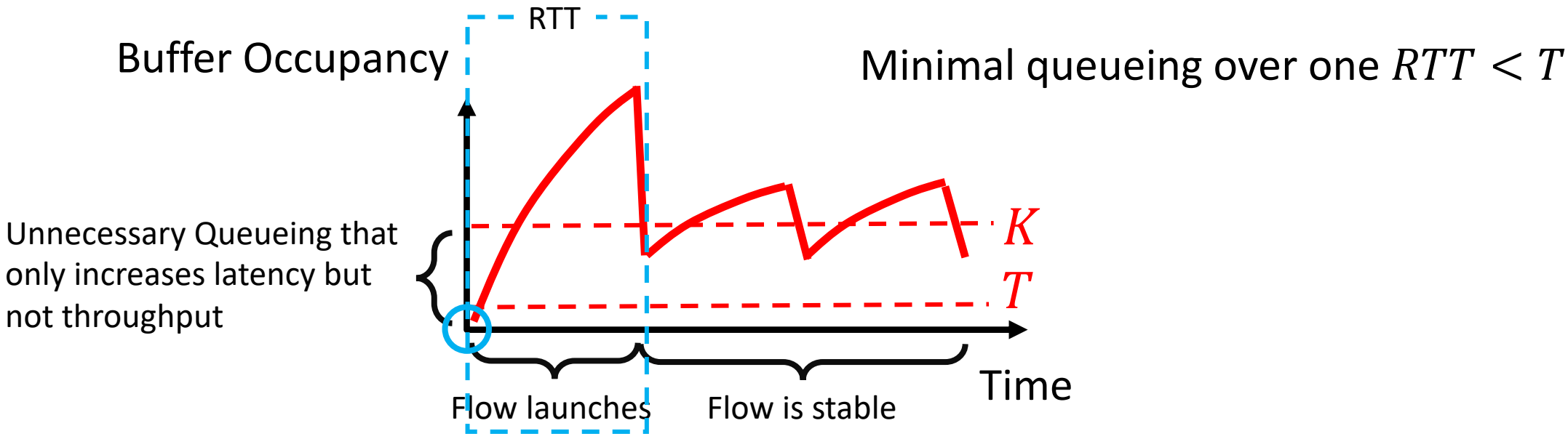
ECN# in Details

- ECN marking based on instantaneous queueing
- ECN marking based on persistent queueing
  - Compare the minimal queueing over an interval I with threshold T



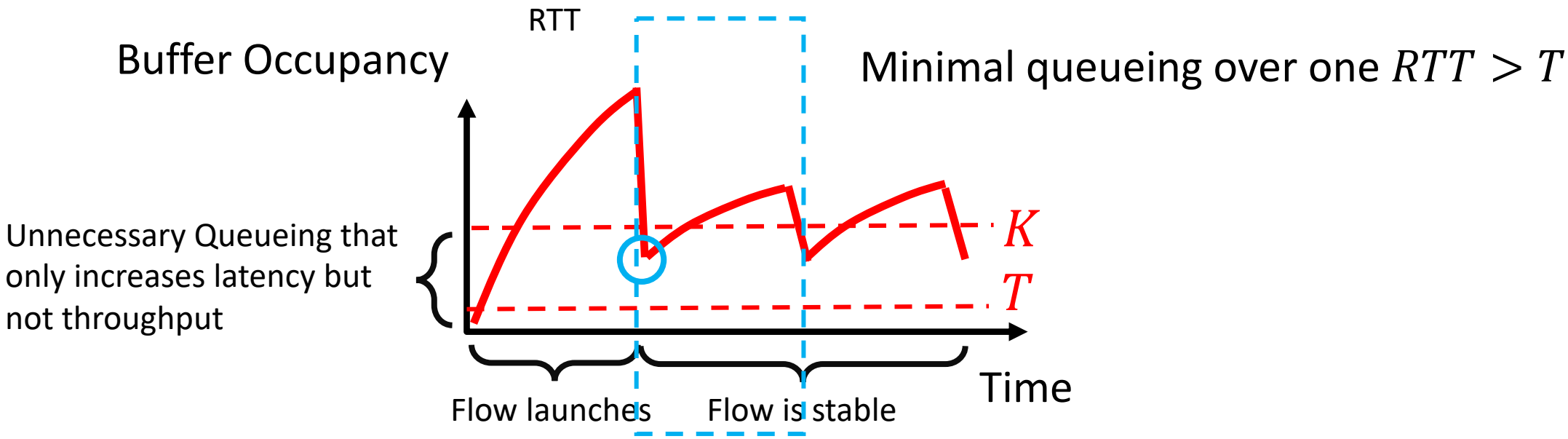
ECN# in Details

- ECN marking based on instantaneous queueing
- ECN marking based on persistent queueing
  - Compare the minimal queueing over an interval I with threshold T



ECN# in Details

- ECN marking based on instantaneous queueing
- ECN marking based on persistent queueing
  - Compare the minimal queueing over an interval I with threshold T



ECN$^{\#}$ in Details

- ECN marking based on instantaneous queueing
- ECN marking based on persistent queueing
  - Compare the minimal queueing over an interval I with threshold T
  - We set I as high percentile RTT and T a relatively small value
  - This strategy eliminates unnecessary queueing by marking packets conservatively.

ECN# in Details

- ECN marking based on instantaneous queueing
- ECN marking based on persistent queueing
- ECN# marks packets when either one is satisfied

The advantage of ECN#

- Low latency ⟵ Marking based on persistent queueing to eliminate unnecessary queuing
- Good burstiness control ⟵ Aggressive marking of instantaneous marking
- High throughput ⟵ The instantaneous marking threshold is calculated based on high percentile RTT

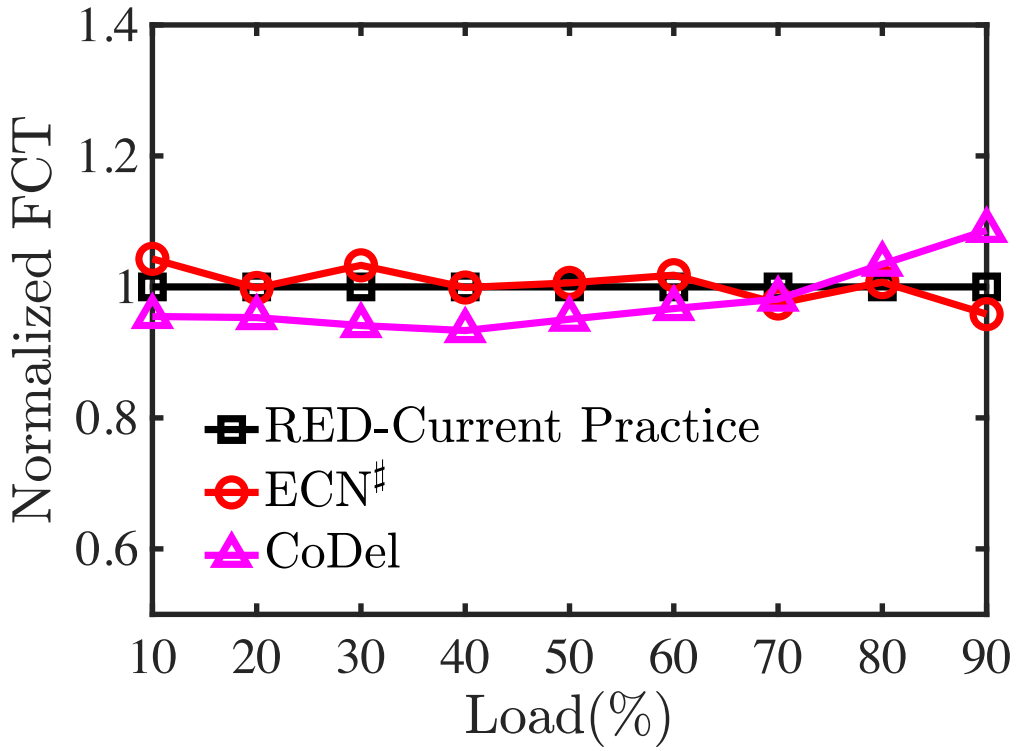
Hardware Implementation

- We implement ECN# on Barefoot Tofino switches
  - Emulate high precise system time
  - Update switch states at line rate

Evaluation

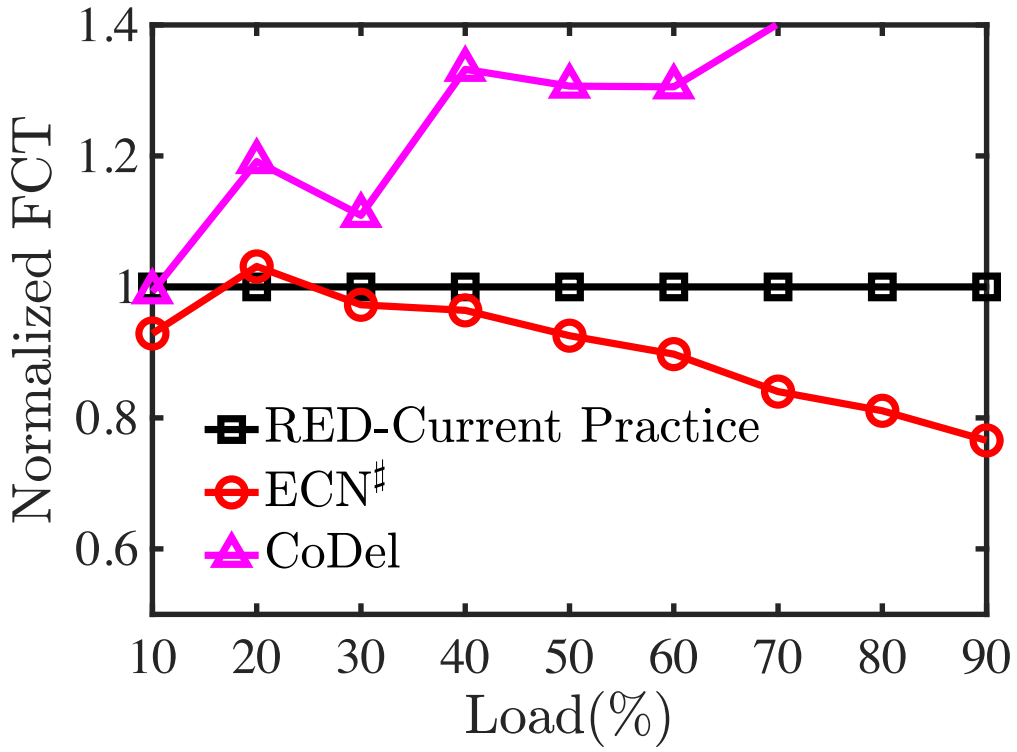
- Simulation + testbed evaluation
- Testbed setup
  - 8 servers are connected to a Barefoot Tofino switch
  - DCTCP is used at all endhosts.
  - NETEM is used to add delay at endhosts to emulate RTT variations
- Scheme compared
  - Current practice: RED with threshold calculated based on high percentile RTT
  - CoDel

Realistic Traffic: FCT of All Flows



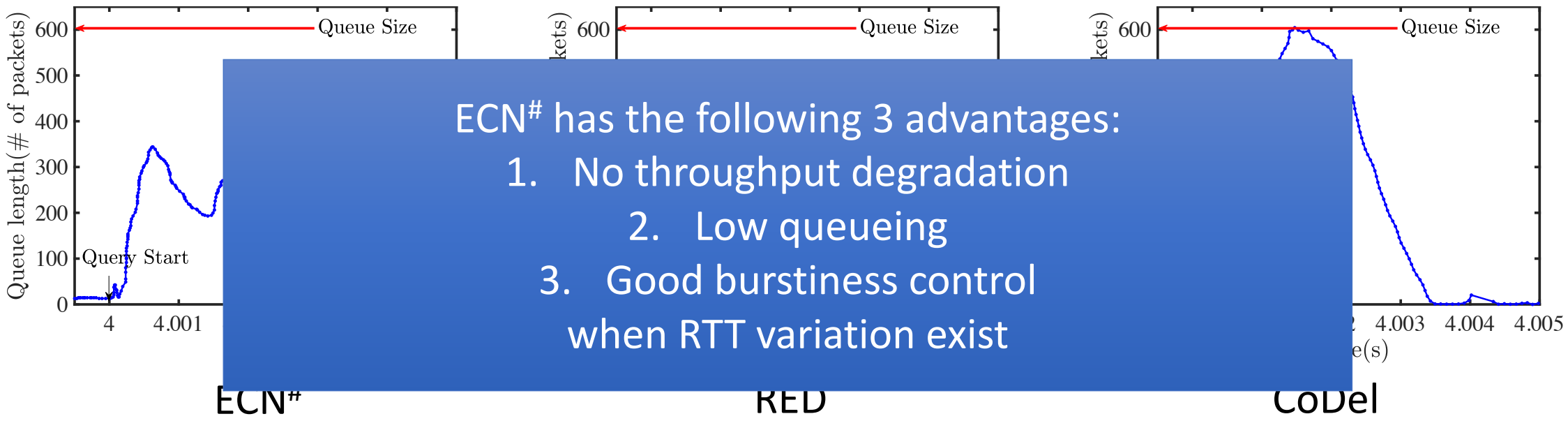
Consider overall throughput of the network, ECN# can achieve comparable performance as current practice of RED and CoDel.

Realistic Traffic: FCT of Short Flows ($< \mathbf{100KB}$)



For latency-sensitive short flows, ECN# can achieve much better performance.
CoDel achieves the worst due to frequent packets drops

Simulation: Microscopic View of Queues



ECN# has the following 3 advantages:

1. No throughput degradation
2. Low queueing
3. Good burstiness control when RTT variation exist

1. Compared to RED, ECN# can effectively eliminate unnecessary queueing
2. Compared to CoDel, ECN# has good burstiness control

Conclusion

- Detect the problem of RTT variations
- ECN$^{\#}$: a simple yet effective ECN solution for datacenters with RTT variations
  - Leverage instantaneous ECN marking to have good burstiness control
  - Use ECN marking based on persistent queueing to eliminate unnecessary queueing caused by RTT variations
- Code: https://github.com/snowzjx/ns3-ecn-sharp

README.md

Very happy to see that our simulator helps more and more papers

ns3 Simulator for ECN#

Papers that use this simulator

Enabling ECN for Datacenter Networks with RTT Variations (CoNEXT 19)

Resilient Datacenter Load Balancing in the Wild (SIGCOMM 17)

PURR: a primitive for reconfigurable fast reroute: hope for the best and program for the worst (CoNEXT 19)

Download and Compile

1. Ubuntu + gcc-4.9 has been verified to compatiable with the project.

```
docker run -it gcc:4.9
```

2. Clone the proejct.

```
git clone git@github.com:snowzjx/ns3-ecn-sharp.git
```

3. Configuration.

```
cd ns3-ecn-sharp
```

```
./waf -d optimized --enable-examples configure
```

4. If you want to enable the debug mode for logging, can pass `-d debug` to the configuration.

Thanks