Reducing Latency in Multi-Tenant Data Centers via Cautious Congestion Watch

The 49th International Conference on Parallel Processing (ICPP) 2020

Ahmed M. Abdelmoniem, Hengky Susanto, and Brahim Bensaou

Outline

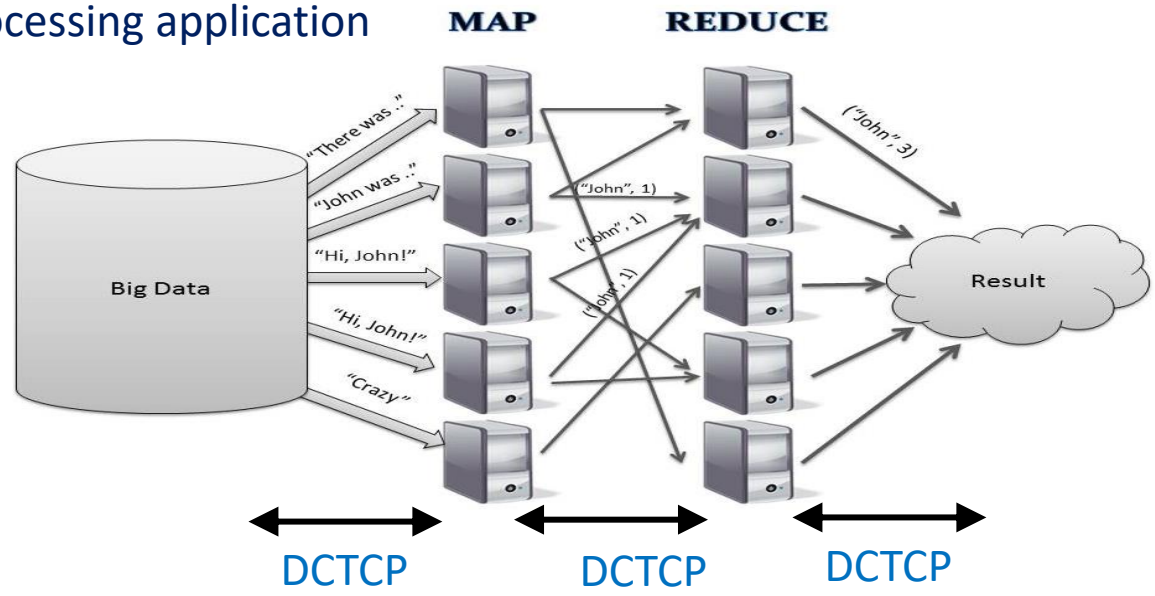
- Introduction and background
- Preliminary Investigation
- Exploring solution design space
- The solution (Hwatch) design
- Evaluation
- Conclusion

Living in *Big Data* Era

- Massive amount of data being generated, collected, and processed daily.
  - Data for recommendation, machine learning, business intelligence, scientific research, etc.
- To accelerate the processing speed, this massive amount of data *must be processed in parallel*.

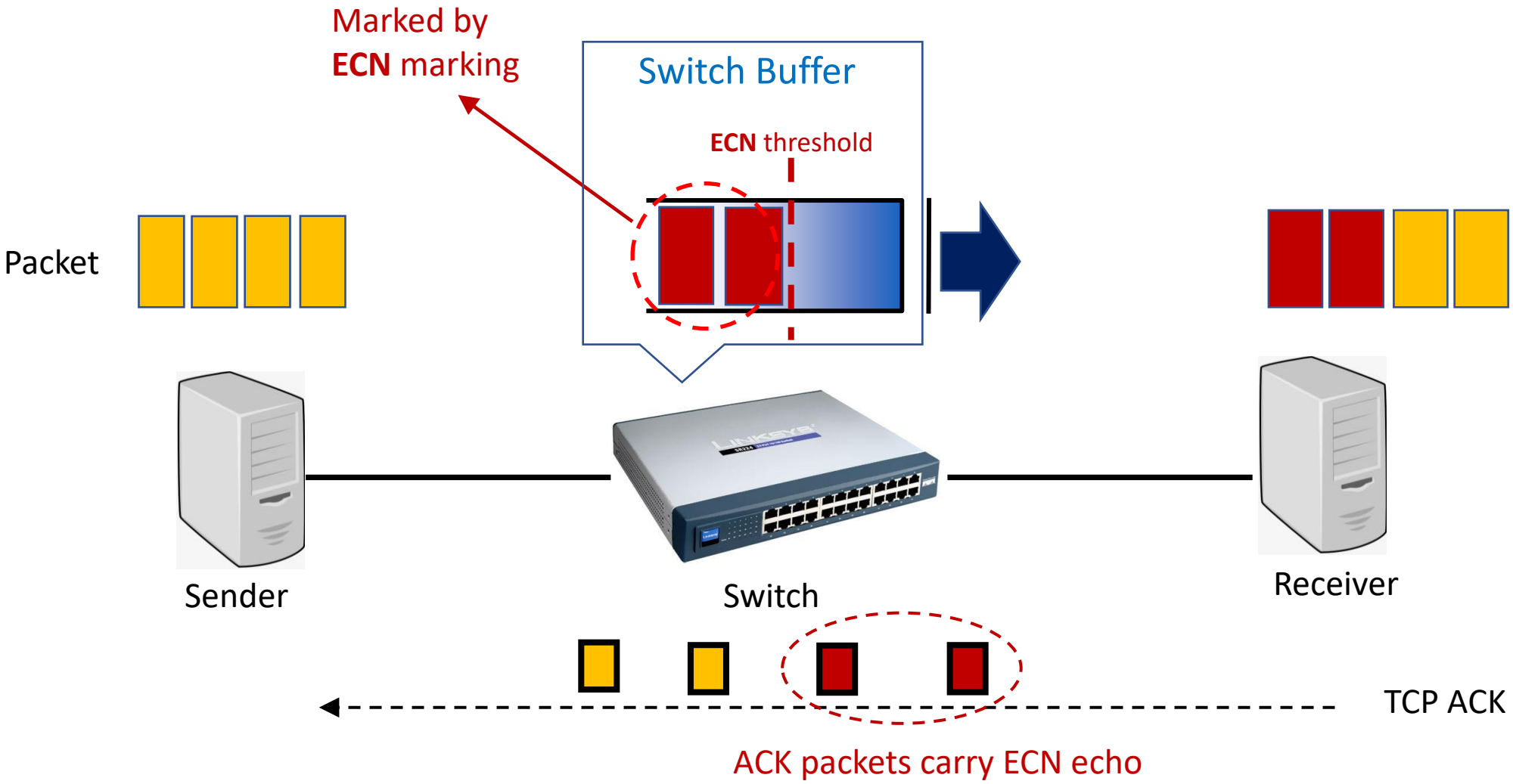
Congestion Control in Data Center

Data parallel processing application

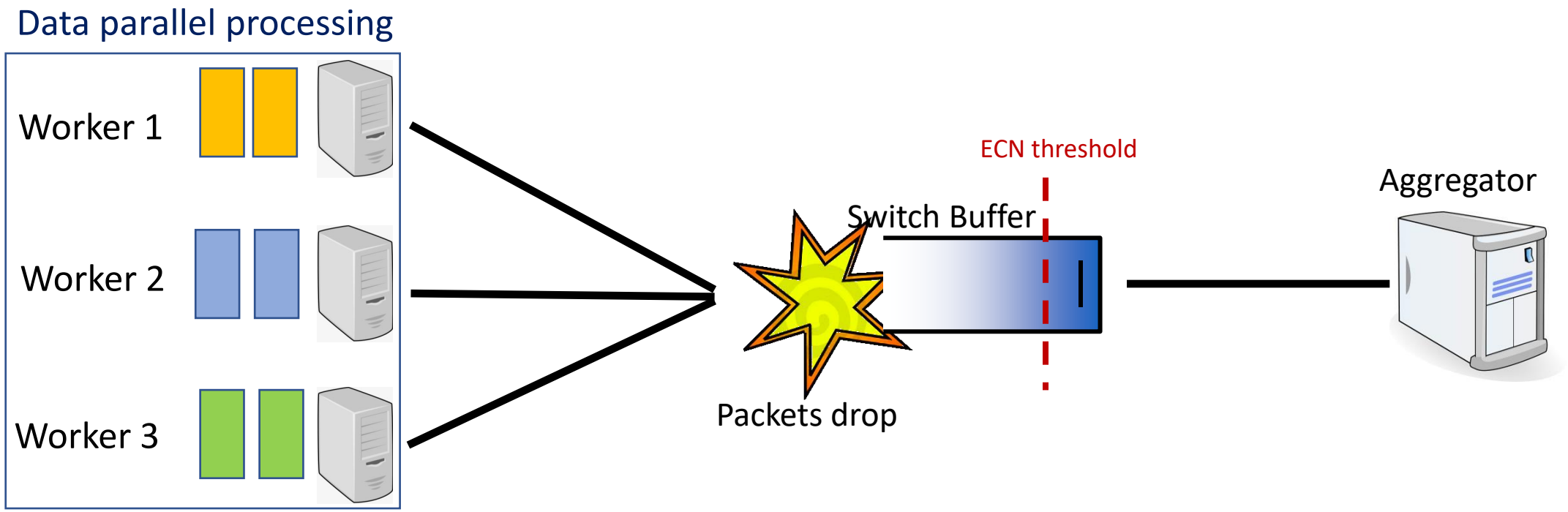


- Data parallel processing is generally conducted in a data center.
- **Modern datacenter employ *Data Center TCP (DCTCP).***
  - DCTCP is a TCP-like congestion control protocol designed for data center networks.
  - DCTCP leverages Explicit Congestion Notification (ECN) to provide multi-bit feed-back to the end host.

DCTCP Overview



Incast Problem in Datacenter Network

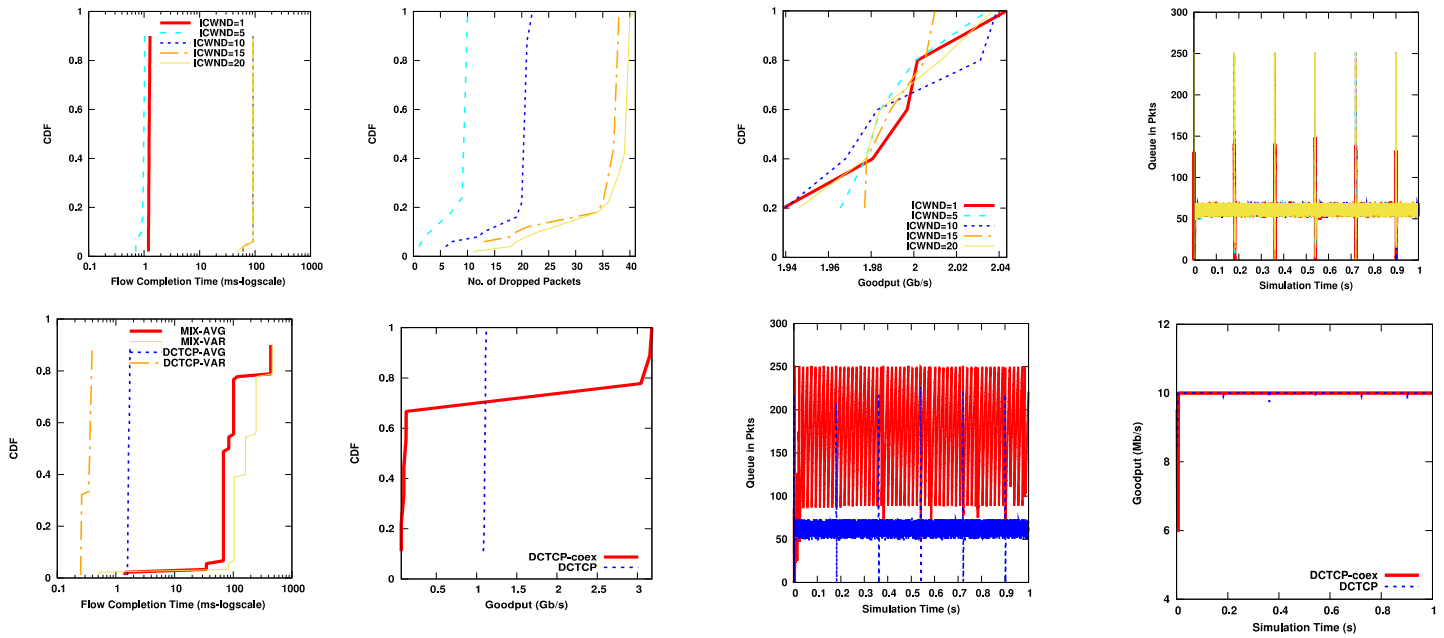


- Incast is bloated buffer incident.
  - Caused by burst of packets arriving at the same time.
  - *Common occurrence* for data parallel processing in data center network.
  - Often leads to network and application level performance degradation.

Preliminary Investigation

Better understanding of how well DCTCP coping with the incast problem.

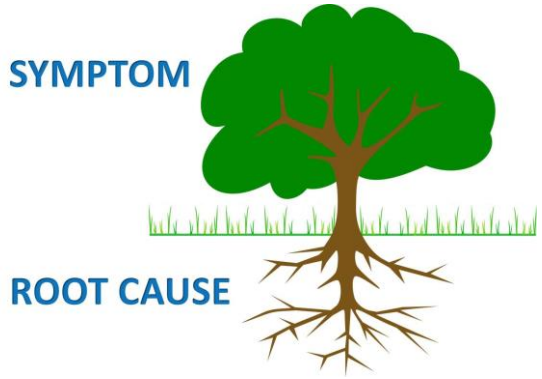
Investigating Incast Problem



Conduct preliminary experiment via NS2 simulator in dumbbell network topology.

- Short-lived flows are sensitive to the choice of the initial sending window size.
- DCTCP marking results in the more aggressive acquisition of the available bandwidth.
  - Favor large flows (e.g. data backup traffic) over short-lived flow (e.g. search query) that may degrade the performance short-lived flow .

The Root of The Performance Degradation



- *Observation 1* : Short-lived flows (e.g. 1 to 5 packets) do not have enough packets to generate 3 DUPACKs to trigger TCP fast re-transmission scheme.
  - Or the sender may lose the entire window of packets.
  - Sender must rely on TCP RTO to detect packet drops. (Default Linux's RTO = 200 *ms* to 300 *ms*).
- *Observation 2*: *Incast problem primarily affects short-lived flows*.
  - Today's data parallel processing applications (E.g. Map-Reduce) generate many short-lived flows.

Our findings show that, short-lived flows in *DCTCP still suffer from packet losses due to the use of large initial congestion windows* that results in incast problem.

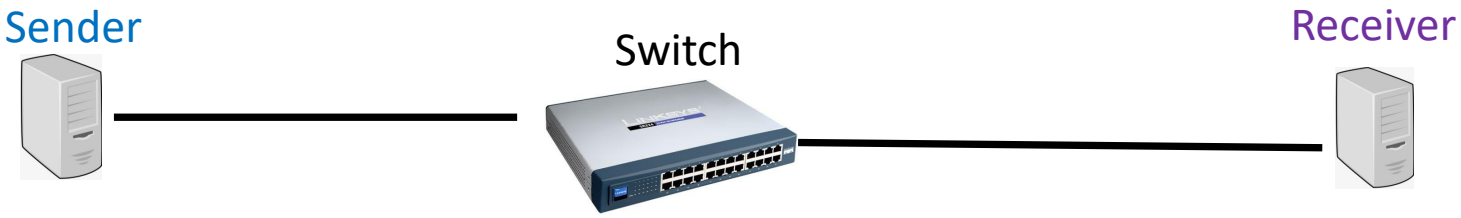
Research Question

What is the optimal choice of initial congestion window for DCTCP that mitigates incast problem, while minimizing the average completion time of short-lived flows?

Exploring Solution Design Space

To provide a holistic view of the problem and better understanding on the complex interactions between network components (switch, sender, and receiver).

Exploring The Design Space at Sender and Receiver



Due to nature of TCP based protocol, there is a wealth of information available at the **sender**.

- Number of transmitted packets, congestion window size, RTT, .., etc.

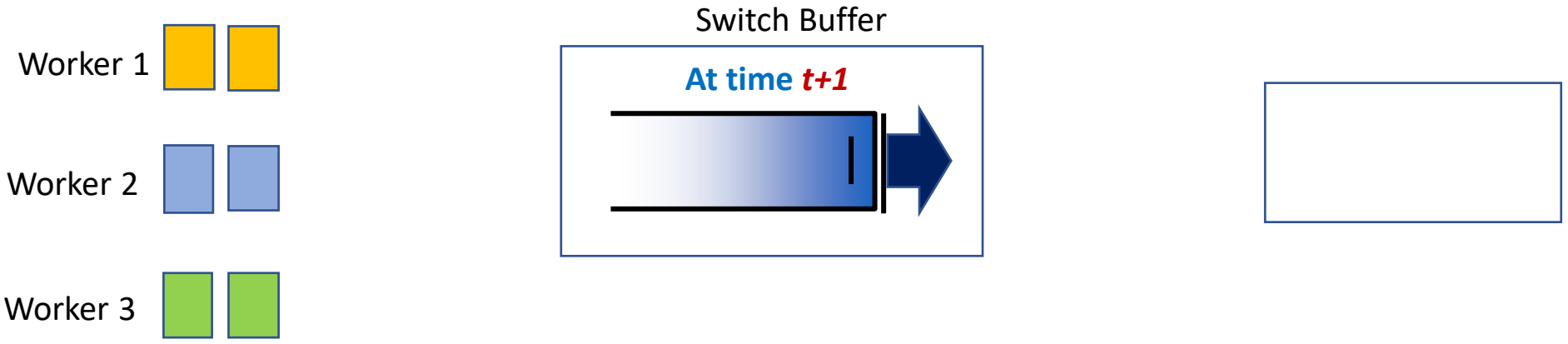
A **receiver** has a natural position to evaluate information carried by packets with ECN marking from inbound traffic.

Combining information gathered from both the sender and receiver provides a sender with a richer and more holistic view of the network condition.

- For instance, the number of packets dropped can be approximated by subtracting the number of packets received by the receiver from the total number of transmitted packets by the sender.

Exploring The Design Space at The *Switch*

Observation : Transmitting packets at different time helps to mitigate packets drop.

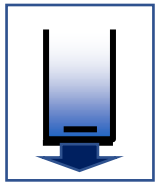


Visualizing Queue Management as *Bin-Packing*

Worker 1

Worker 2

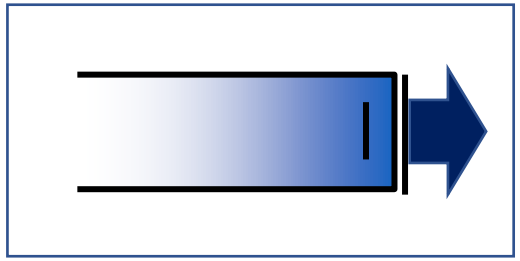
Worker 3

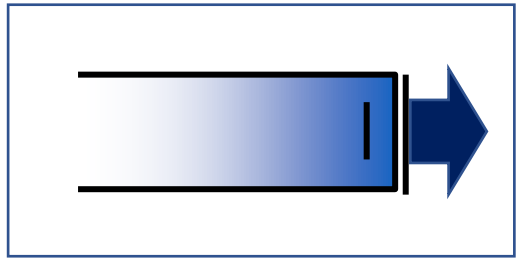


Visualize switch buffer as the **bin.**

Visualize packet as the **items** that are packed into a bin.

Switch Buffer





At time *t*

At time *t+1*

Visualizing Incast As Bin-Packing

- Different perspective in understanding incast problem.
  - Starting point to think how to *transmit a bulk data of short-lived flows,* which only consists of few packets.
- Allowing us to inherit wisdom from earlier study on bin-packing problem.
  - Our solution draws inspiration from the classic solution, Next Fit Algorithm.
- Emulate Bin-Packing problem by utilizing ECN used in DCTCP.
  - The ECN setup follows the recommendation for DCTCP.

Theoretical Results

Key insight gained from our theoretical results:

- The initial congestion window size can be approximated by leveraging information base on the number of packets marked and unmarked by ECN.
- Given the number of packets marked by ECN, the first n transmitted packets should be transmitted in at least two batches (rounds).

From Theory to Practice - *Practical* Challenges

Despite encouraging outcomes from our theoretical results, to realize the theory to practice, we must consider the *practical* challenges.

- Incast problem is a distributed online problem.
  - Bin-packing is an offline problem.
- Short-lived flows (e.g. 1 to 5 packets) may be completed before the sender receives the ECN echo via ACK packets.
  - Short-lived flows only learn about in network congestion after receiving ECN echo, which is too late.

System Design Requirements of The Solution

- Improves the performance of short-lived flows with minimal impact on large flows.
- Simple for deployment in data center.
  - Being Independent of the TCP variant.
  - No modifications to the VM’s network stack.
- Addresses the practical challenges.

Our solution:

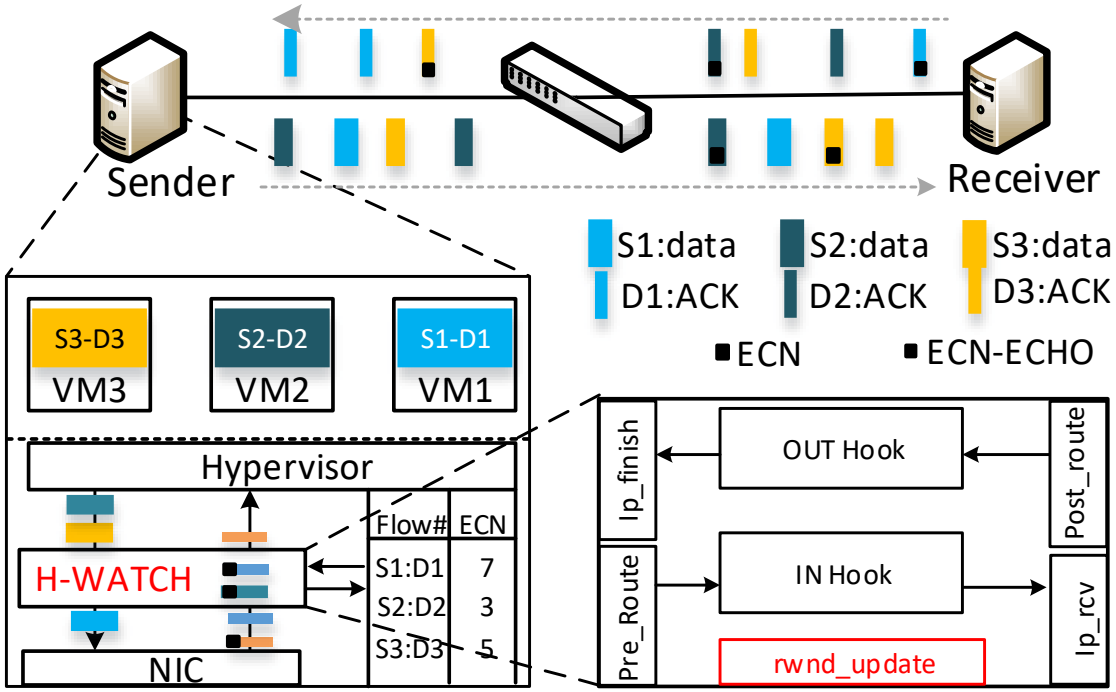
HWatch

(*Hypervisor-based congestion watching mechanism*)

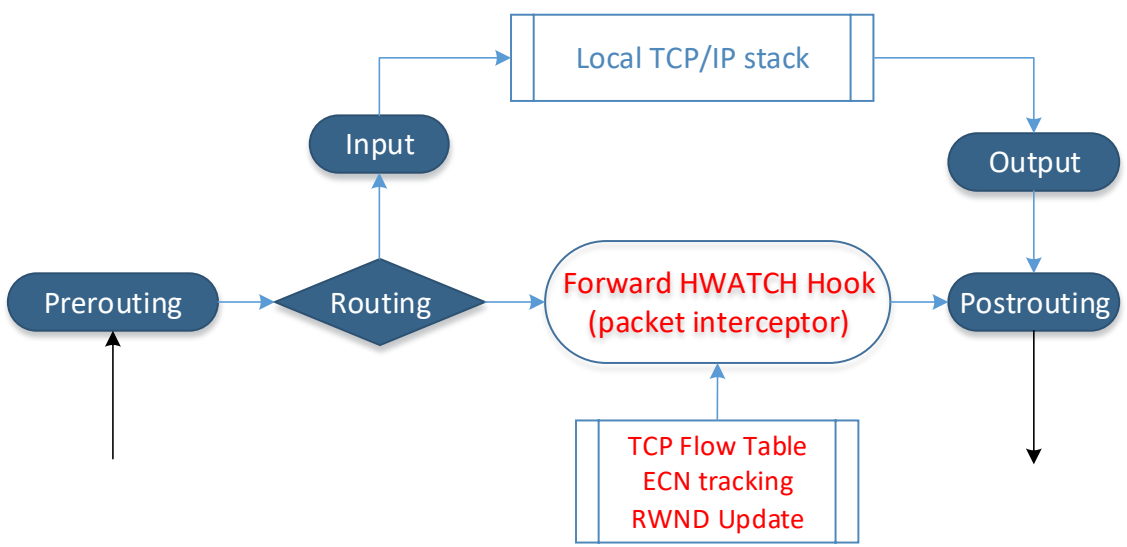
An active network probe scheme, that incorporates insights from our theoretical results, to *determine the initial congestion window* based on the congestion level in the network.

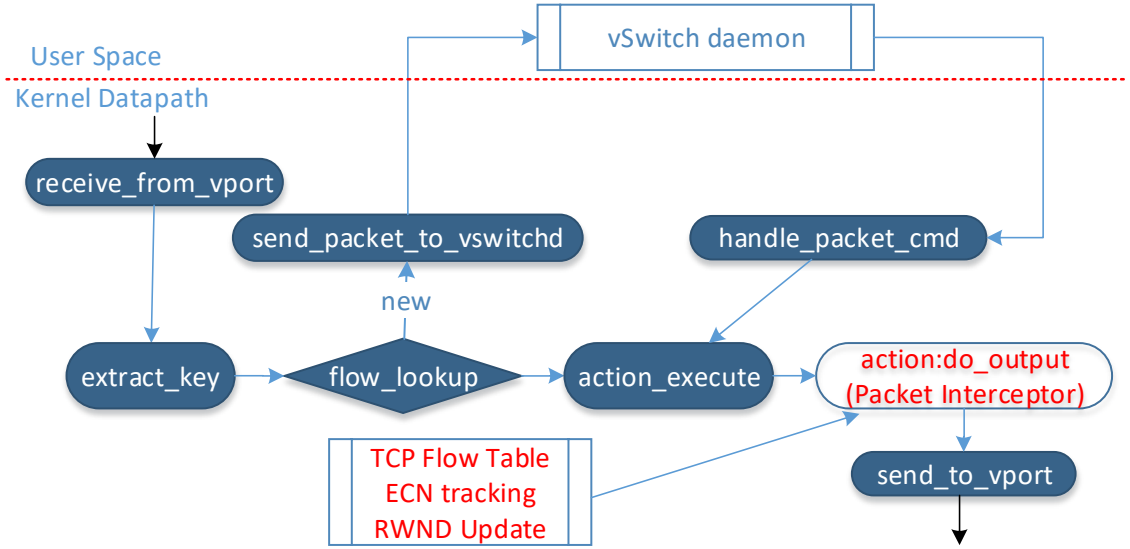
HWatch System Design in a Nutshell

- Injects probe packets at connection start-up during TCP synchronization stage.
- The probe packets carry the ECN marks to the receiver in the case of congestion.
- Receiver sets the receive window (RWND) according to number of probe packets marked with ECN marking
- Conveys RWND to sender via ACK packet.
- Sender sets the *Initial congestion window* (ICWND) according to RWND.



HWatch Implementation Overview



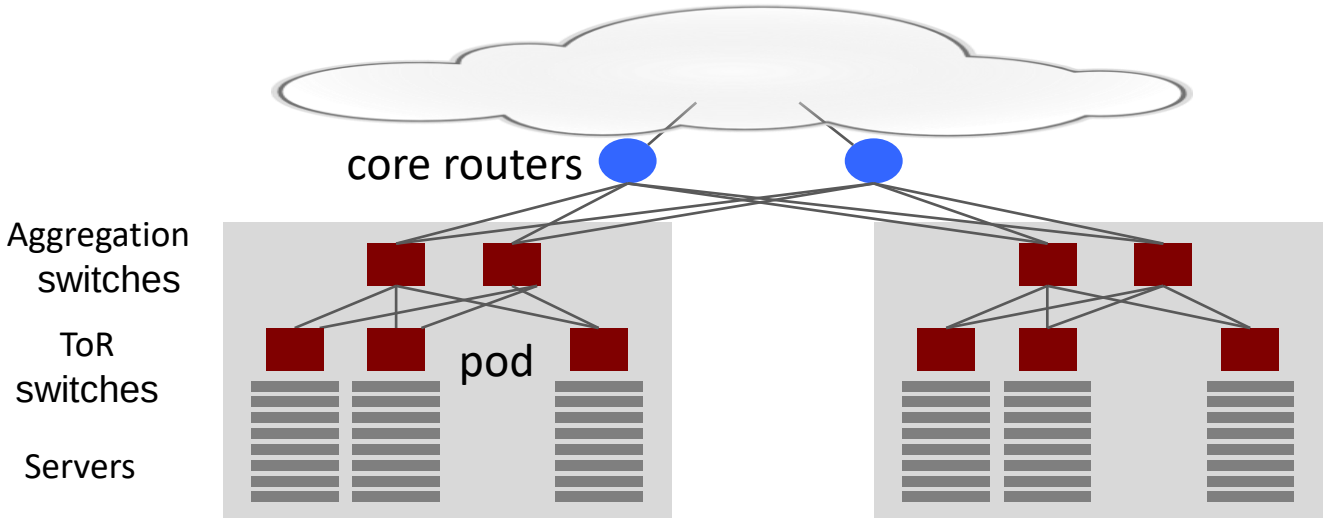


- The Hwatch module is implemented in the shim-layer in both sender and receiver.
  - Hwatch is implemented via Netfilters by inserting hooks in the forward processing path.
- HWatch is realized by modifying the OpenSwitch kernel datapath module.
  - Adjusting the flow processing table.
- By doing so, Hwatch is deployable friendly in the production datacenter.

Evaluation and Analysis

(1) Simulation experiments.
(2) Testbed experiments.

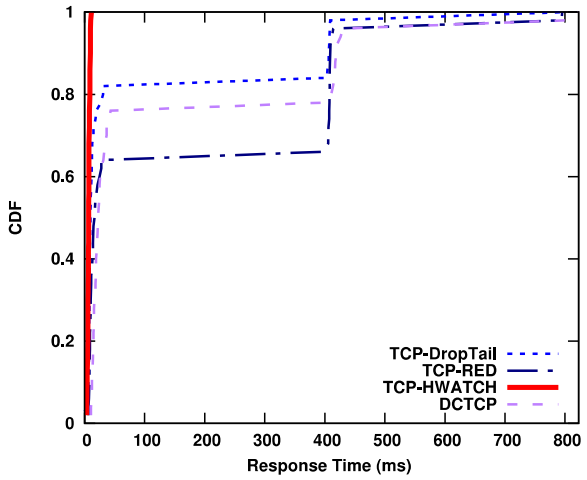
Simulation Experiment Setups



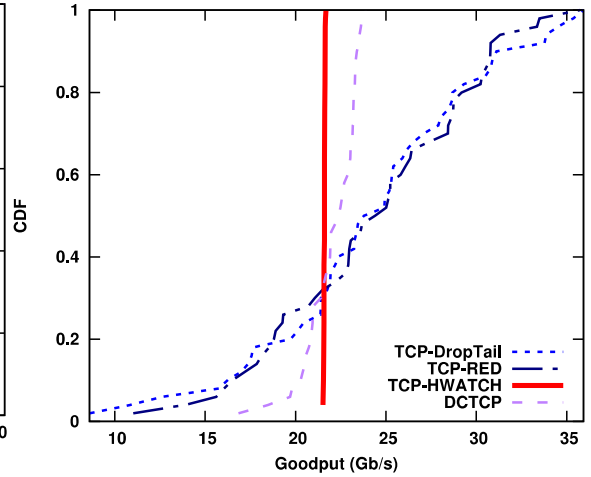
Fat Tree Topology

- NS2 simulator.
- Over 100s of servers connected by commodity switches.
- Compared to different type TCP traffic sources (TCP-DropTail, TCP-Red, and DCTCP).

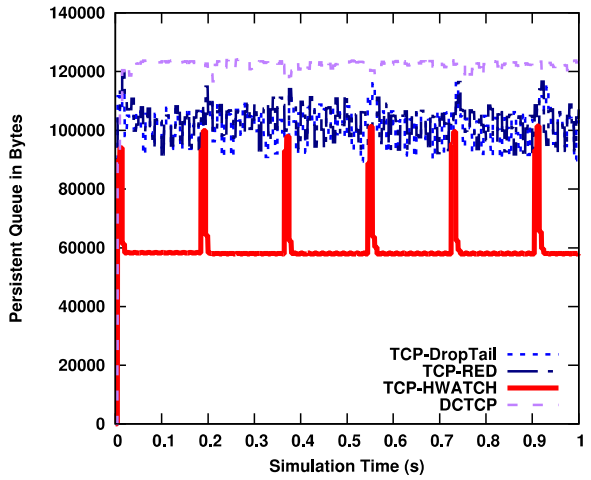
Simulation Experiment Results



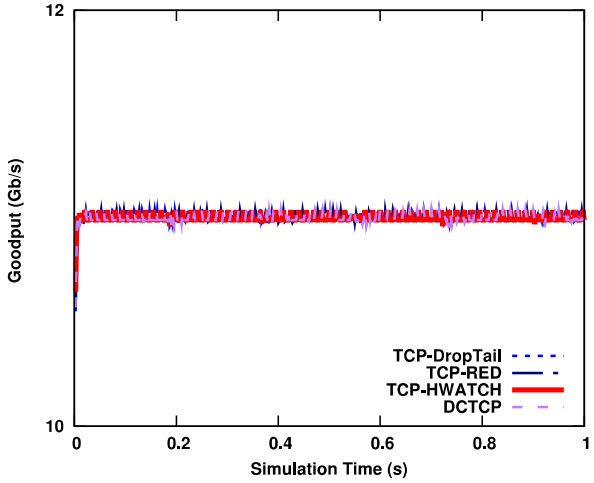
Short-Lived flows: Avg FCT



Long-Lived flows: Avg Goodput



Persistent queue over time

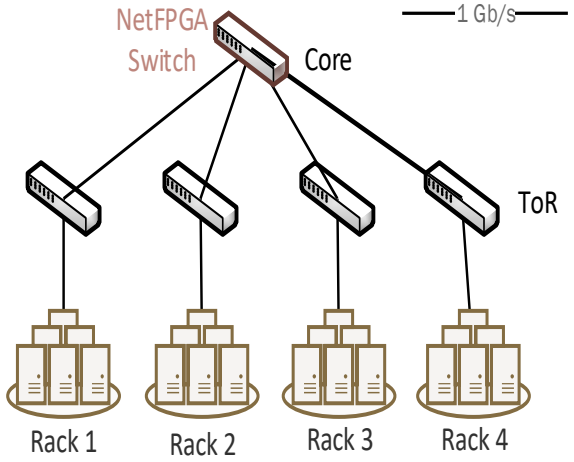


Bottleneck utilization over time

Performance of short-lived and long-lived flows in over 100 sources scenario.

- HWatch improves the performance of short-lived flows by ***up to 10X*** compares to TCP-DropTail, TCP-Red, and DCTCP.
  - Minimal impact to the performance of large-flows.

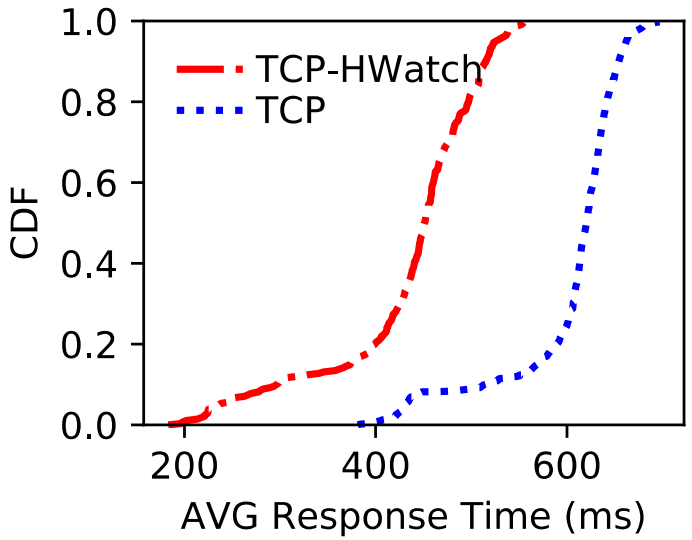
Testbed Experiment Setup



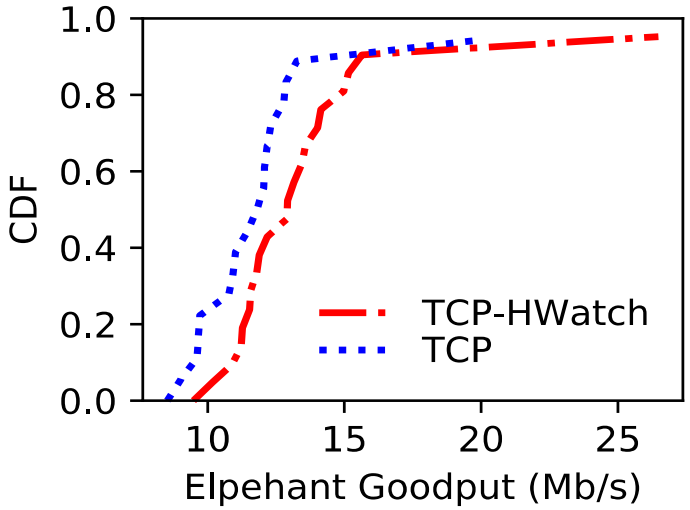
Fat Tree topology

- Build and deploy HWatch prototype in mini data center.
- *Fat Tree* topology connecting 4 Racks of DC-grade servers installed with Incast Guard End-host Module.
- Commodity (EdgeCore) Top of Rack switches.
- Core Switch is PC installed with the NetFPGA Switch.

Testbed Experiments Results



Short-Lived flows: Average FCT



Long-Lived flows: Average Goodput

Test experiments confirms that HWatch mitigates packets drop.

- Improves the performance of short-lived flows by ***up to 100%***.
- Minimal impact to the performance large flows.

Key Insights From HWatch Performance

- Dispersing packets transmission over time, creates smaller size of incasts.
  - Allowing buffer to observed the incoming packets with fewer packet drop.
  - Fewer packet drop allows short-lived flows to achieve faster completion time.
- HWatch stochastically prioritizes the available buffer space to short-lived flows.
  - Because probe packets indirectly reserve buffer space for short-lived flows.
- Probing mechanism functions as incast early warning system allowing long-lived flows that are actively sending data to scale back to release some buffer space for short-lived flows.

Conclusion

- HWatch mitigates incast problem in the data center.
- Hwatch strikes the balance between improving the performance of short-lived flows while imposing minimal impact on the large flows.
- Hwatch is deployment-friendly in production data center.

Thank You

Exploring The Design Space at The *Switch*

Illustration of the switch buffer experiencing ***incast***.