LAB OF COMPUTER COMMUNICATION & NETWORKING
Networking Education, Research and Innovation
The Henry and Merilyn Taub Computer Science Department, Technion

DCTCP VS. SWIFT

DATA CENTER CONGESTION CONTROL COMPARISON

AUTHORS: YASMIN MITKAL , OMAR GARAH, AMIR WATTED
ADVISOR: ERAN TAVOR

ABSTRACT

Efficient congestion control in data centers is critical to achieving low latency and high throughput under shallow-buffer conditions. We compare two modern protocols—DCTCP (Data Center TCP), which uses ECN-based proportional reactions, and Swift, a new and evolving delay-based congestion controller via NS-3 simulations and packet-level analysis.

OBJECTIVE

Compare DCTCP and Swift as two data center congestion control protocols under realistic data center conditions.

PROJECT OVERVIEW

- *Understand the core principles of DCTCP and Swift.*
- *Ramp up and configure the NS-3 simulator.*
- *Build a fat-tree topology in NS-3.*
- *Analyze the existing implementation of DCTCP in NS-3.*
- *Implement the Swift protocol within NS-3.*
- *Design and run simulations under varying load levels and traffic patterns.*
- *Compare DCTCP and Swift performance across different scenarios.*

PROTOCOLS OVERVIEW

DCTCP: *ecn-based*

- *Switches mark packets with ECN when queue > threshold K*
- *Receiver echoes ECN marks; sender tracks fraction **α** of marked packets*
- *Congestion window updated: cwnd ← cwnd × (1 – α/2)*
- *Retains standard TCP features: slow start, additive increase, loss recovery*

(1) $\alpha \leftarrow (1-g) \times \alpha + g \times F$

(2) $cwnd \leftarrow cwnd \times (1 - \alpha/2)$

SWIFT: *delay-based*

- *RTT as Congestion Signal: reacts to end-to-end delay rather than ECN marks*
- *Topology-Based Target Delay: per-flow RTT goal that scales with path hops and active flow count (topology- and load-aware)*
- *ACK-Driven AIMD: if RTT ≤ target → additive increase; if RTT > target → multiplicative decrease proportional to delay excess*
- *Hardware-Based Delay Breakdown: uses NIC timestamps to separate fabric vs. endpoint queuing delays*
- *Incast Pacing: supports sub-packet cwnd (<1) with precise pacing to absorb large bursts*

Algorithm 1: SWIFT REACTION TO CONGESTION

```
Parameters: ai: additive increment, β: multiplicative decrease
  constant, max_mdf: maximum multiplicative decrease factor
cwnd_prev ← cwnd
bool can_decrease ←                      ▹ Enforces MD once every RTT
  (now − t_last_decrease ≥ rtt)
On Receiving ACK
  retransmit_cnt ← 0
  target_delay ← TargetDelay()           ▹ See S3.5
  if delay < target_delay then           ▹ Additive Increase (AI)
    if cwnd ≥ 1 then
      cwnd ← cwnd + (ai / cwnd) · num_acked
    else
      cwnd ← cwnd + ai · num_acked
  else                                   ▹ Multiplicative Decrease (MD)
    if can_decrease then
      cwnd ← max(1 − β · ((delay−target_delay) / delay),
On Retransmit Timeout
  retransmit_cnt ← retransmit_cnt + 1
  if retransmit_cnt ≥ RETX_RESET_THRESHOLD then
    cwnd ← min_cwnd
  else
    if can_decrease then
      cwnd ← (1 − max_mdf) · cwnd
On Fast Recovery
  retransmit_cnt ← 0
  if can_decrease then
    cwnd ← (1 − max_mdf) · cwnd
cwnd ←                                   ▹ Enforce lower/upper bounds
  clamp(min_cwnd, cwnd, max_cwnd)
if cwnd ≤ cwnd_prev then
  t_last_decrease ← now
if cwnd < 1 then
  pacing_delay ← rtt / cwnd
else
  pacing_delay ← 0;
```

Target Delay :

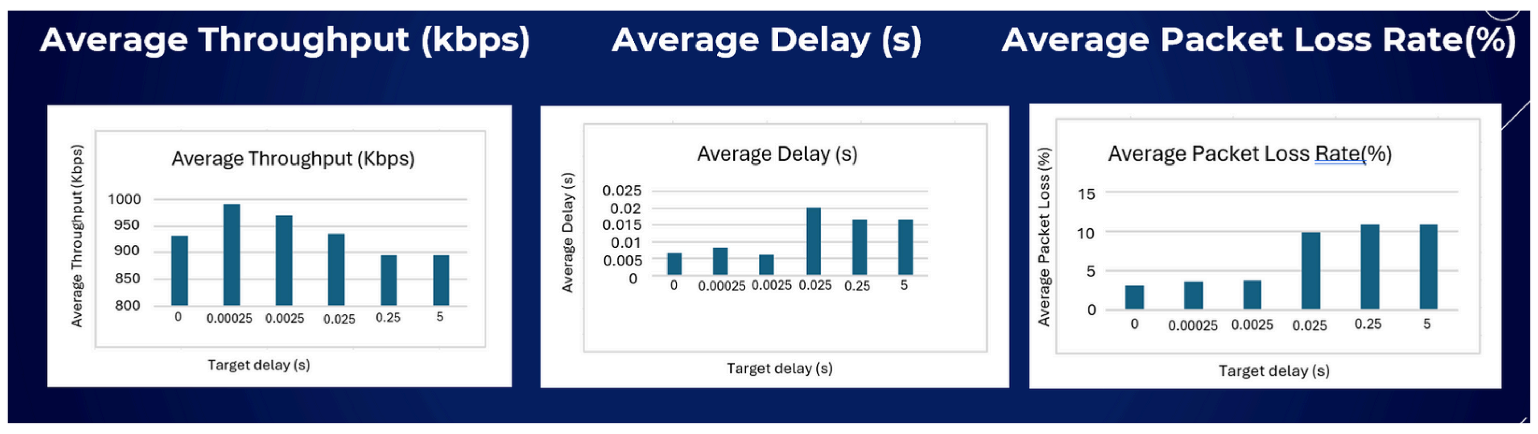
$$t = base\_target + \#hops \times \hbar + max(0, min(\frac{\alpha}{\sqrt{fcwnd}} + \beta, fs\_range))$$

SWIFT IMPLEMENTATION

- *Built from the ground up within NS-3's TCP stack.*
- *In-depth NS-3 TCP integration: analyzed TcpSocketBase, TcpSocketState, and congestion modules to embed Swift logic*
- *Customized congestion ops: implemented Swift's AIMD in a new TcpCongestionOps subclass, overriding PktsAcked and IncreaseWindow*
- *Extensive validation: unit tests, trace callbacks, and packet-level debugging ensured correctness and performance*

TARGET DELAY CONFIGUARATION

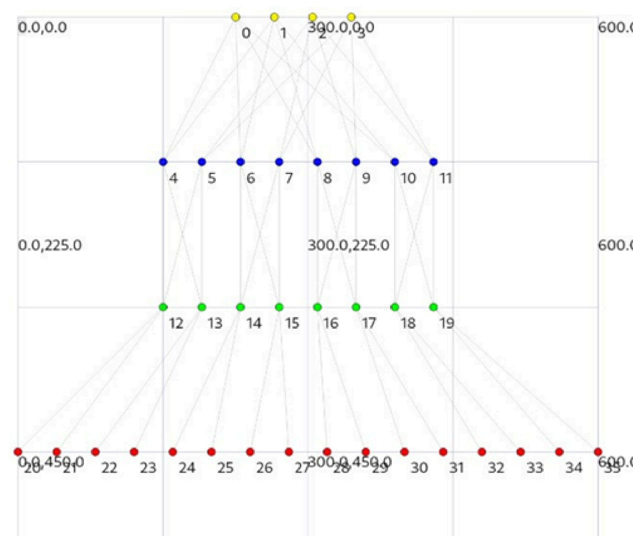
To configure Swift's target delay, we experimented with various fixed delay values. We observed how each value impacted throughput, latency, and packet loss.



- *Best Strategy: Using the minimum observed RTT (baseRTT) as the target delay achieved the best balance across all metrics.*
- *Key Insights:*
  - *Confirms Swift paper's trade-off:*
    - *Larger delays increase throughput but cause higher latency and packet loss.*
    - *Smaller delays reduce latency but limit throughput.*
- *Conclusion: Fine-tuning the target delay is crucial. Static delays must be chosen carefully to avoid under-utilizing bandwidth or overloading the network.*

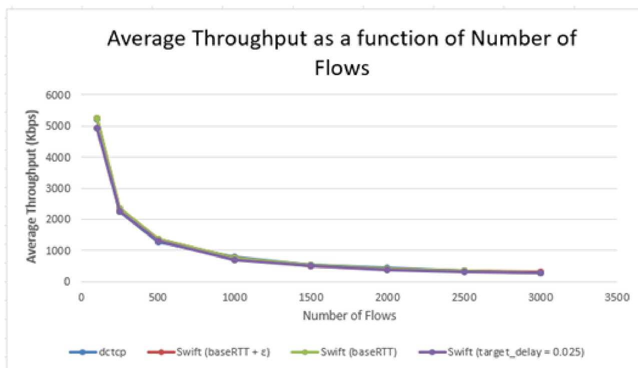
METHODOLOGY & EXPERIMENTAL SETUP

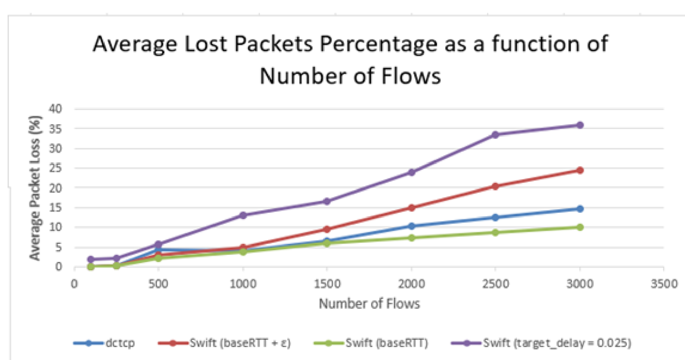
- *Fat-tree topology in NS-3.*
- *Multiple traffic modes.*
- *Metrics: Throughput, Delay, Packet Loss.*
- *On-off application*
- *Averaged over 3 runs per data point.*
- *3 swift variants:*
  - *Static target delay – 25ms*
  - *Target delay is base rtt (min rtt observed)*
  - *Target delay is Base rtt + ∈ ~ 1ms*

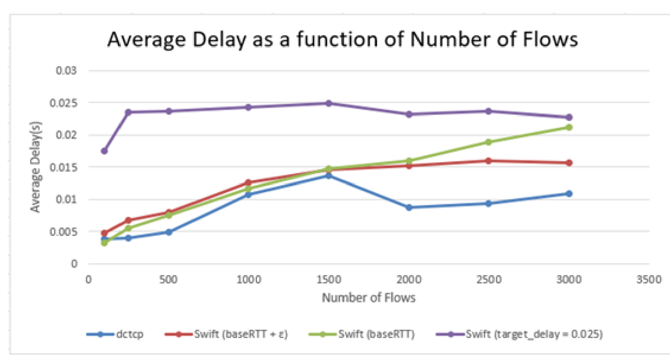


RESULTS

Results for K=12 (432 hosts) and a 7 seconds simulation runtime - random hot spot:







CONCLUSIONS

- *Swift (baseRTT) and DCTCP showed similar throughput. Swift had lower packet loss, DCTCP maintained lower delay.*
- *Results match Swift's original paper claims on throughput and loss. However, delay improvements were not fully replicated in our setup.*
- *Performance is highly sensitive to parameters and target delay tuning.*
- *Target Delay Trade-off: Smaller delays reduce queuing and loss, larger ones boost throughput early but degrade performance under load. baseRTT provides a good balance.*
- *Overall Insight: Swift's delay-based approach is promising, requires refined tuning of delay targets and pacing.*

ACHEIVEMENTS

- *Implemented Swift in NS-3 and uploaded it to GitHub to support the community.*
- *Fine-tuned Swift parameters for improved performance and stability.*
- *Developed automation tools to streamline simulation and result analysis.*
- *Built a solid base for future projects involving Swift and congestion control protocols.*
- *Compared a new, developing protocol (Swift) against an industry-established protocol (DCTCP).*