In Link We Trust: BFT at the Speed of CFT Using Switches

Lior Zeno, Naama Ben-David, Mark Silberstein

Failures in Distributed Systems

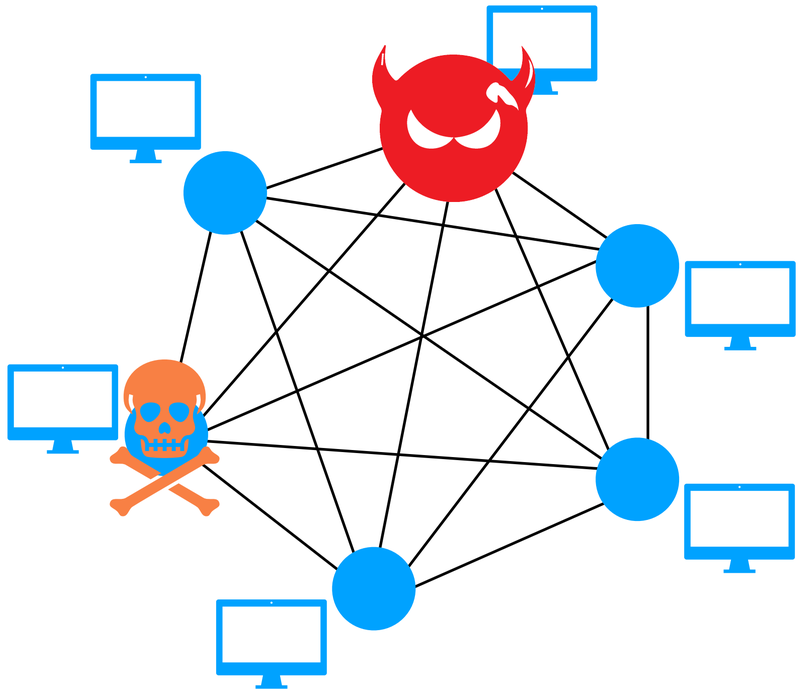
Distributed applications ***must*** be resilient to failures

- Power outages
- Overheating

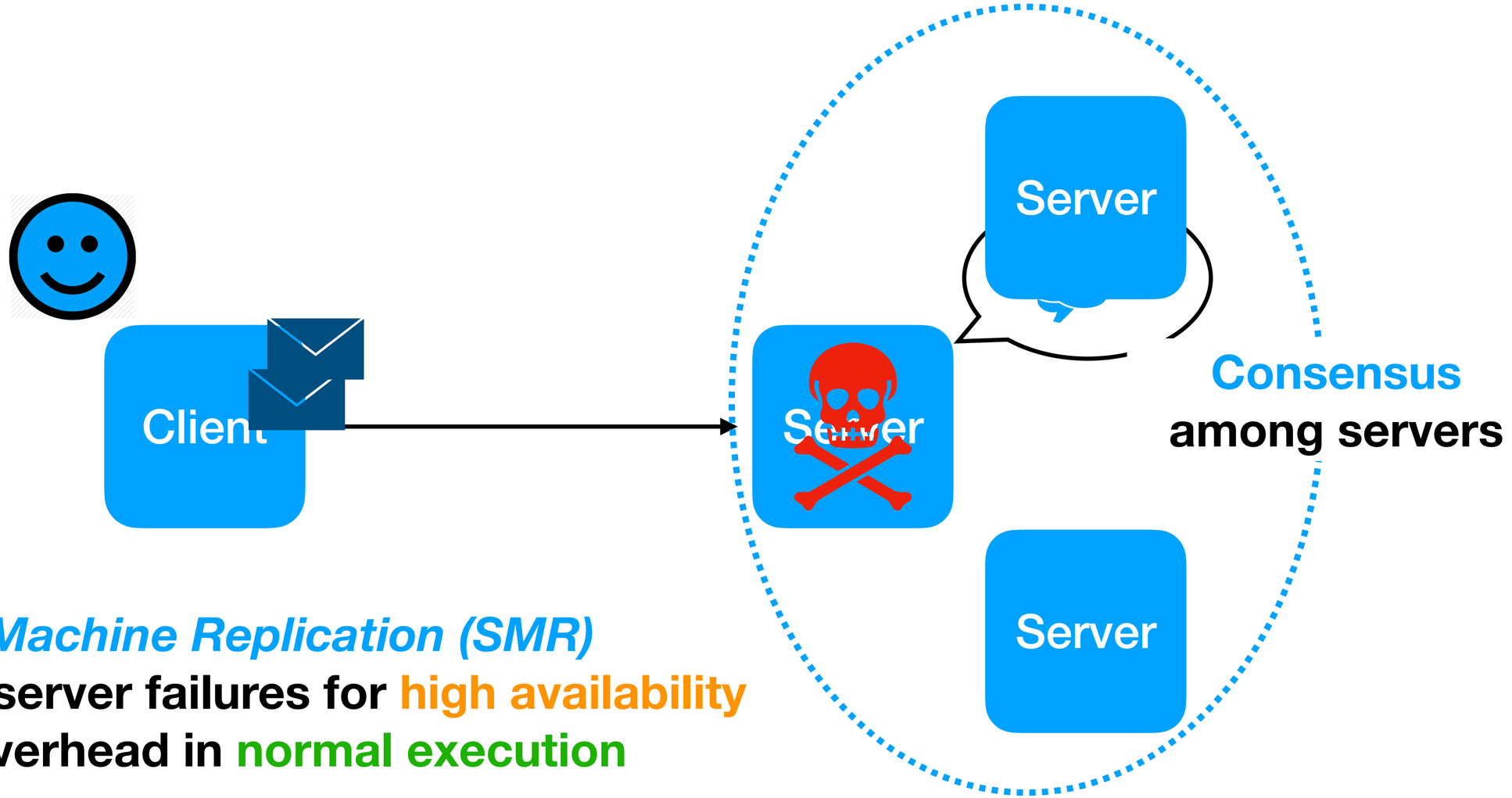
} **Crash** failures

- Bugs
- Attacks

} **Byzantine** failures



State Machine Replication



State Machine Replication (SMR)

Goal: Masks server failures for high availability

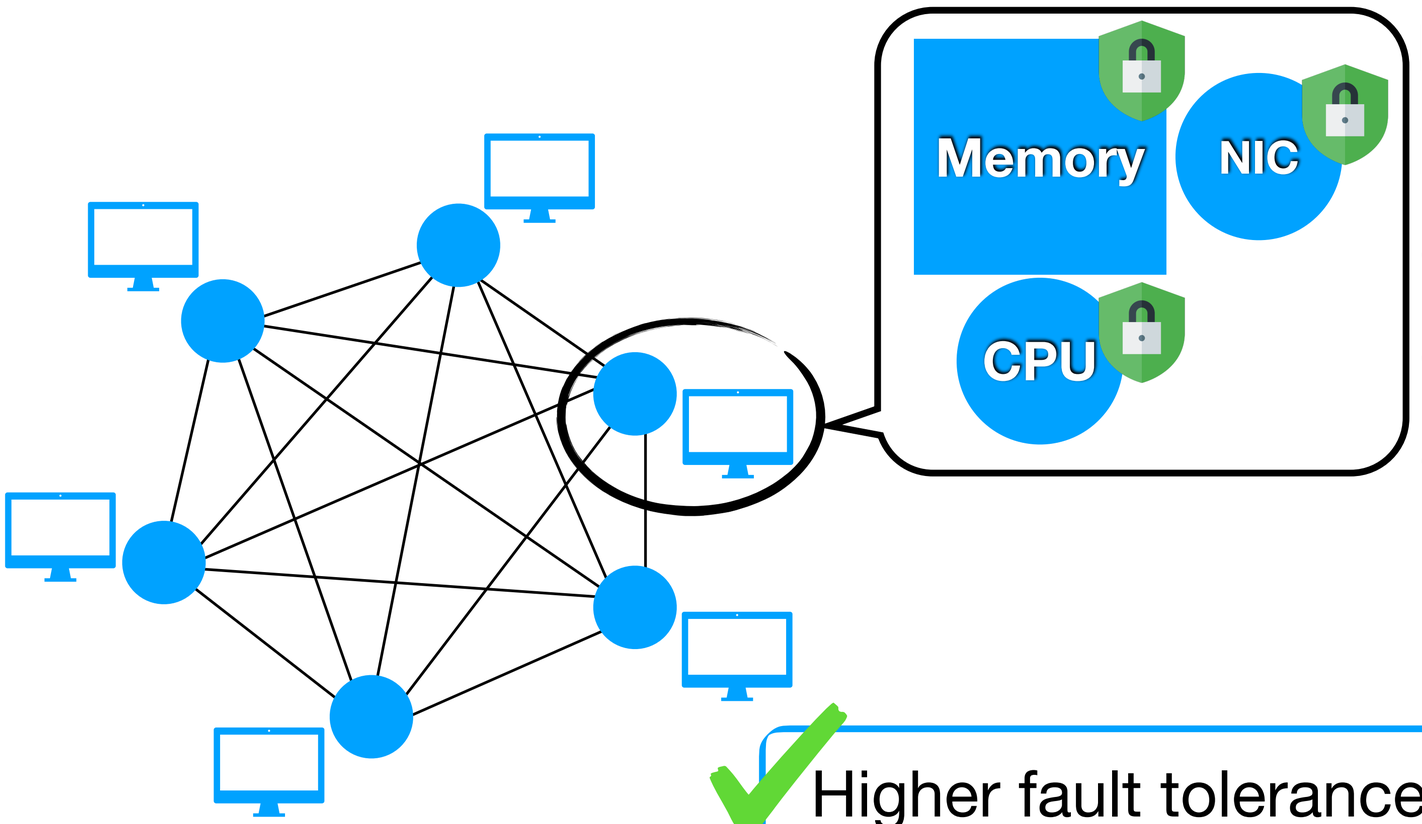
Cost: Overhead in normal execution

Byzantine Fault Tolerance

More **robust** than crash fault tolerance… but also more **expensive**

- 50% more replicas (3f+1 vs 2f+1)
- Often requires heavy cryptographic primitives
- Quadratic vs linear number of messages per leader

How can we Speed Up BFT?



Trusted hardware:

Execute small programs

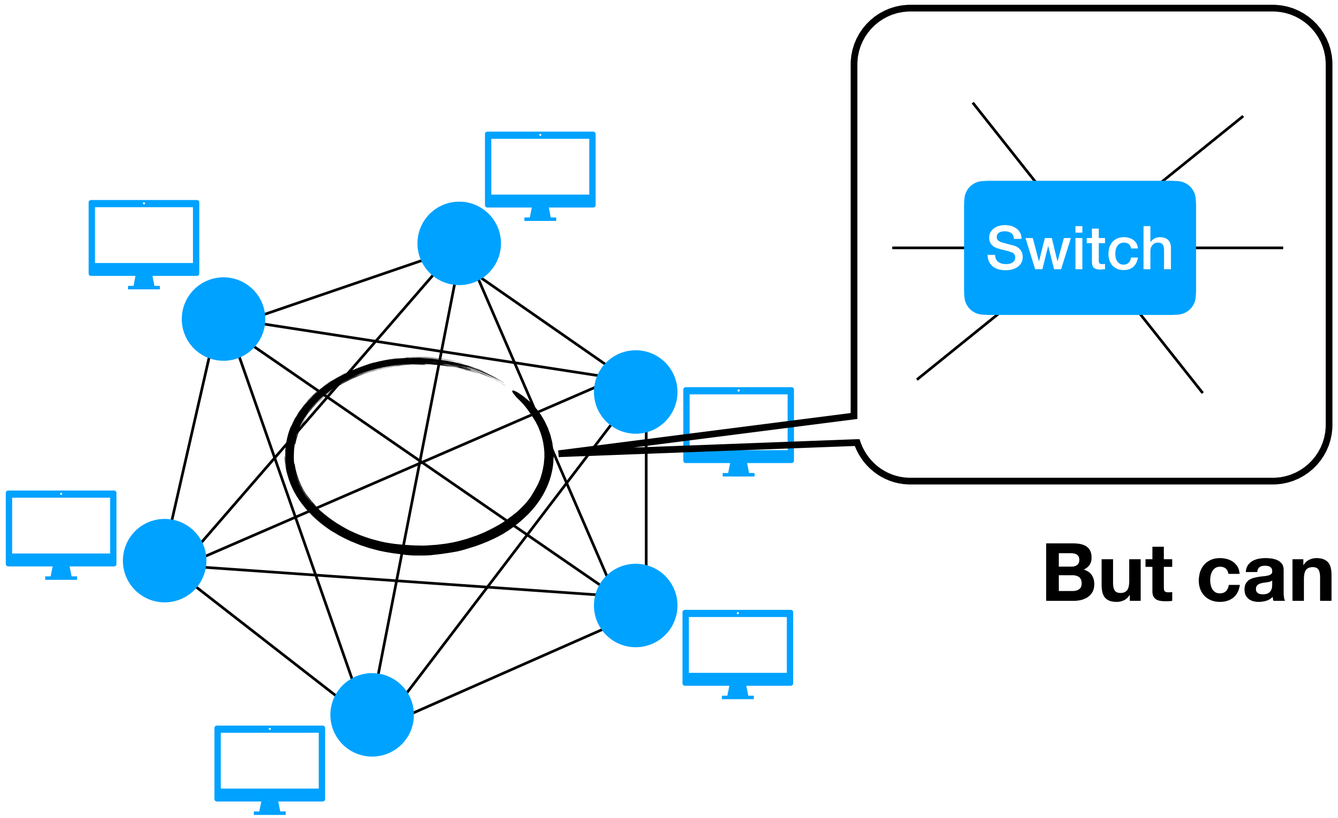
Maintain limited state

- SmartNICs/RDMA [ABGMXZ'23]
- TEEs [VCBLV'11]
- Sticky Bits/Permissions [MRTW'02]

✓ Higher fault tolerance (eliminates equivocation)

✗ Signatures and high communication required [CJKR'12]

How can we Speed Up BFT?

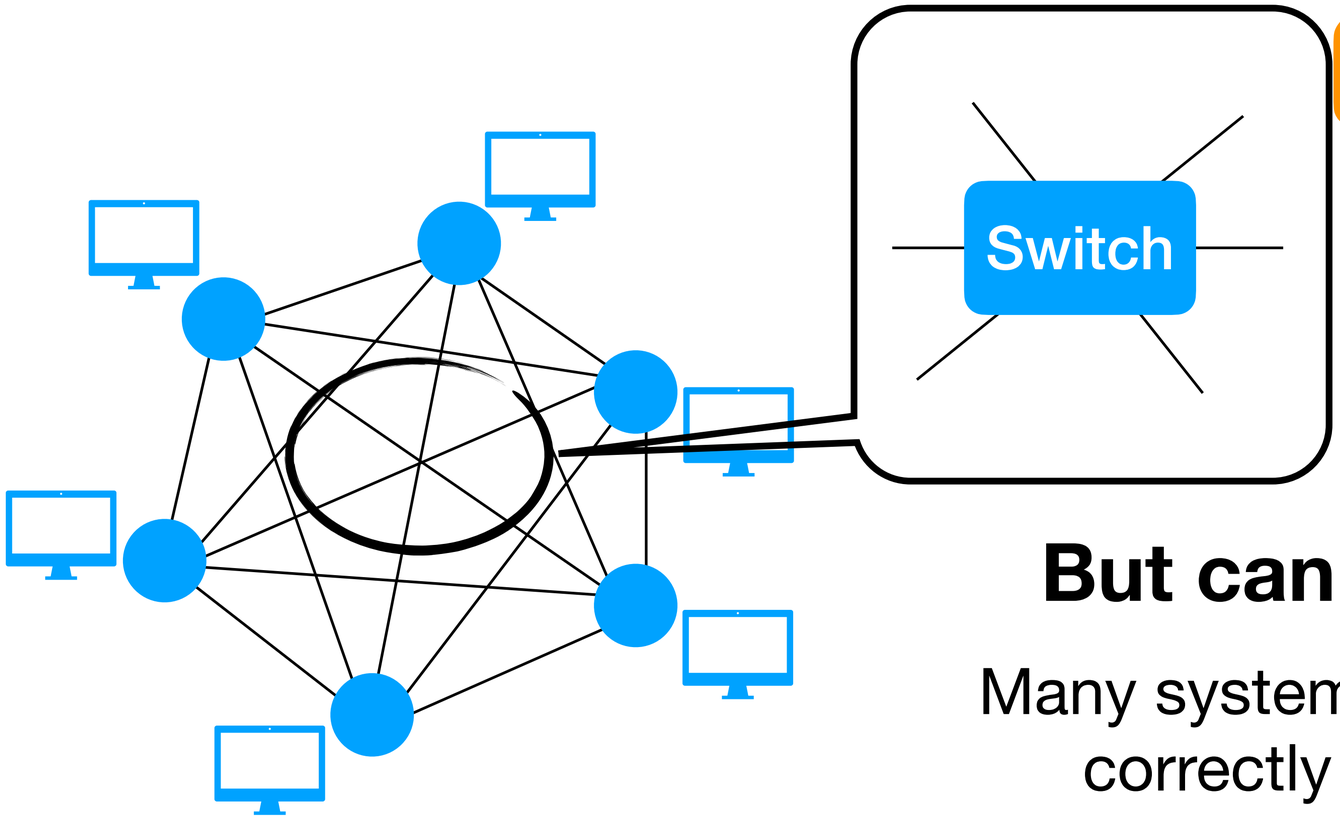


Programmable switch

- All messages routed through it
- Knows the source of a message
- Can execute some computation

But can we trust the switch?

How can we Speed Up BFT?



Through control plane

Programmable switch

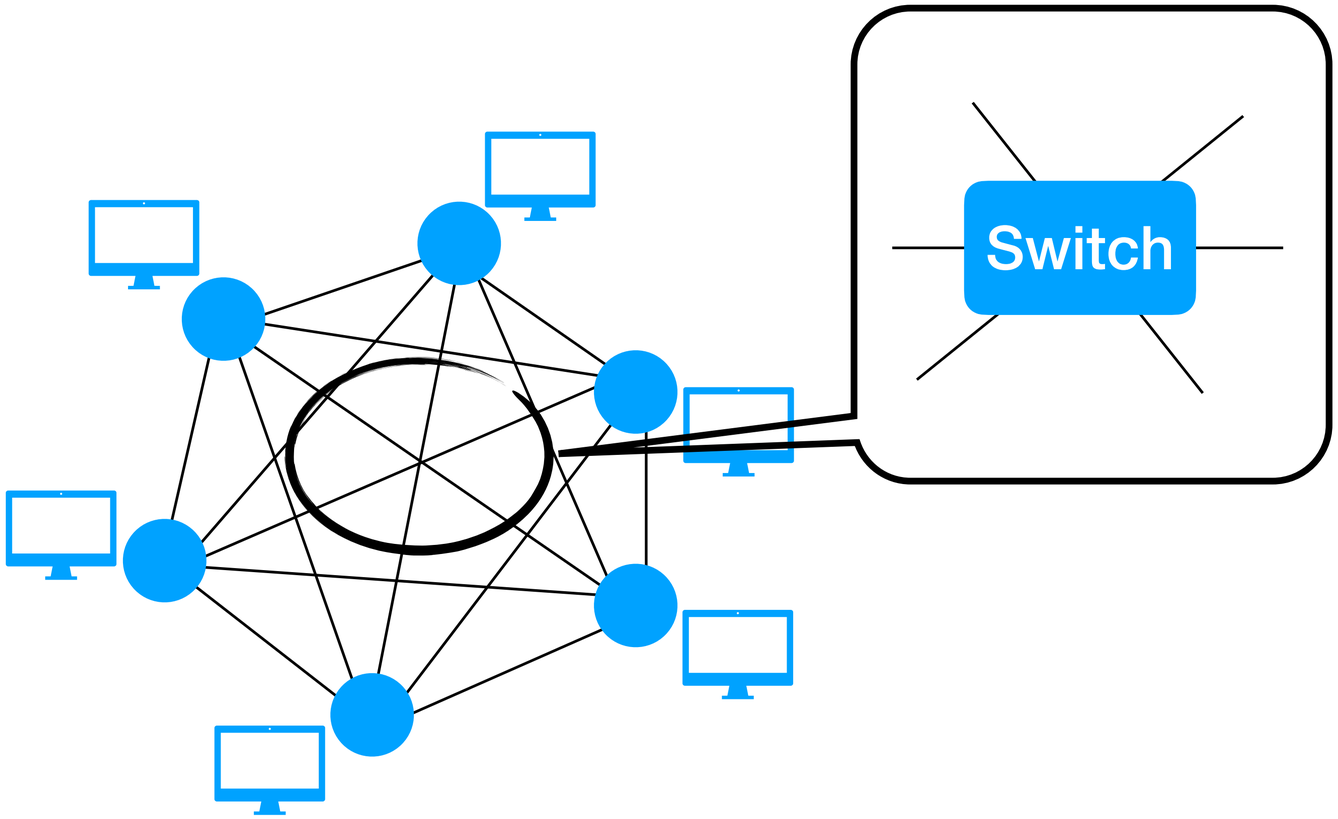
- **All messages routed through it**
- Knows the source of a message
- Can execute some computation

But can we trust the switch?

Many systems implicitly trust the switch to correctly route messages unaltered

Data center tenants do not have control-plane access, which is required to install switch programs

How can we Speed Up BFT?



Programmable switch

- All messages routed through it
- Knows the source of a message
- Can execute some computation

Can we use trusted switches to improve BFT protocols?

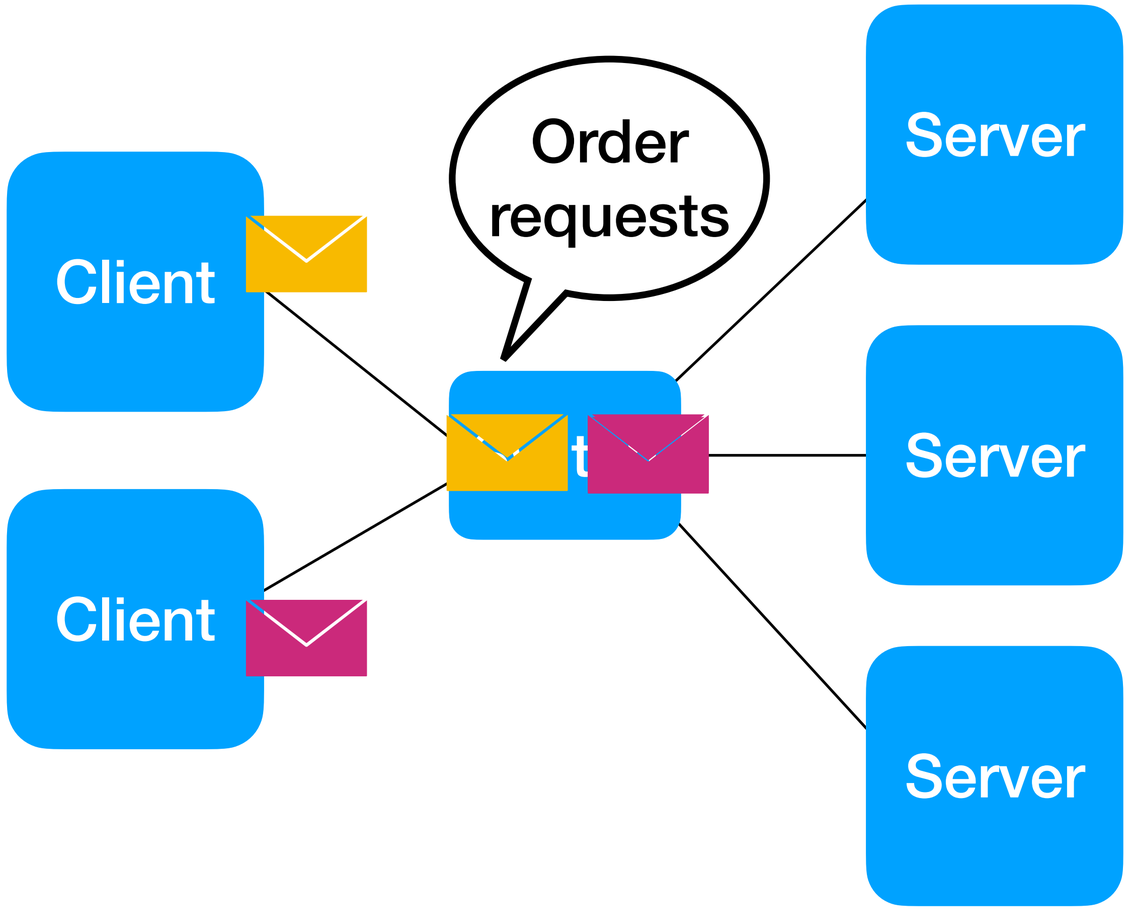
Our Results

Trusted programmable switches
can speed up and increase fault tolerance of BFT protocols

- What programmable switches can and cannot do
- Our protocol - SwitchBFT
- Evaluation

Take 1: Trivial Solution

A centralized trusted entity can solve all our problems!

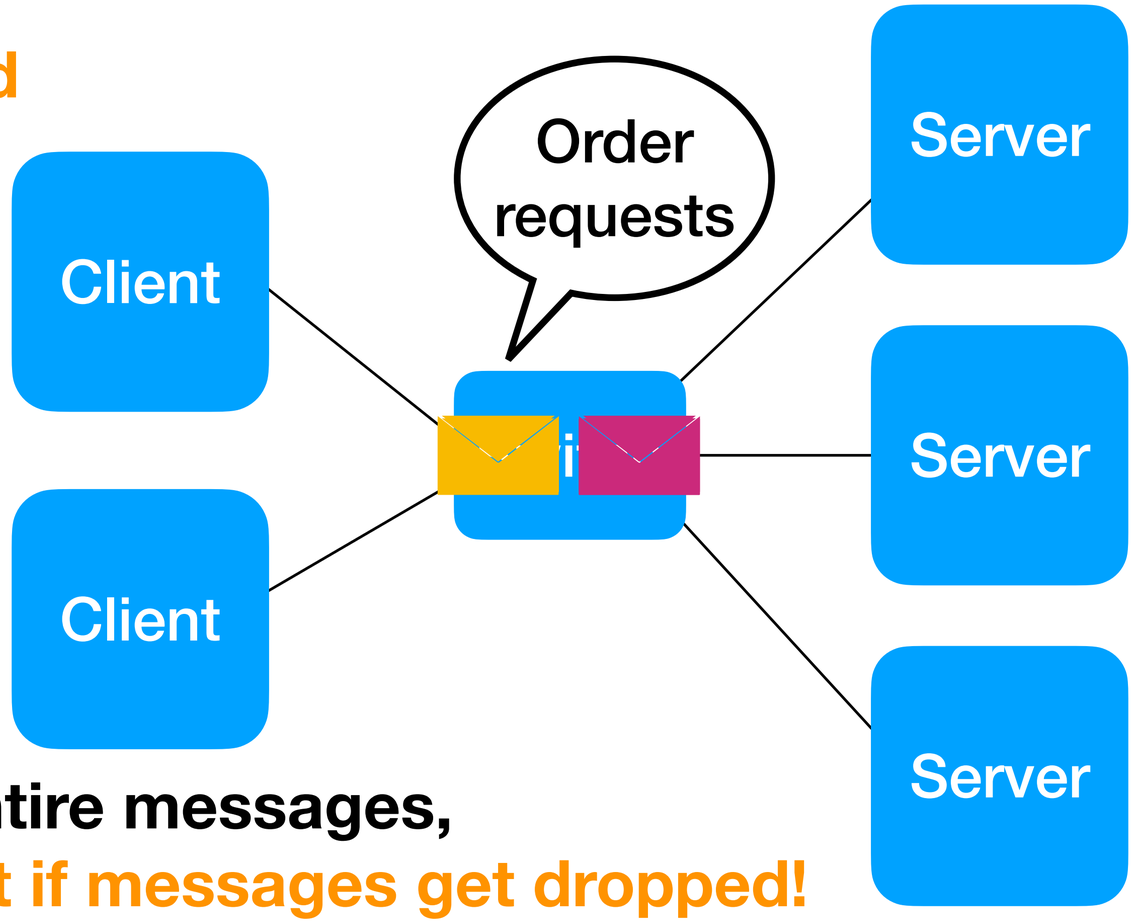


Take 1: Trivial Solution

not

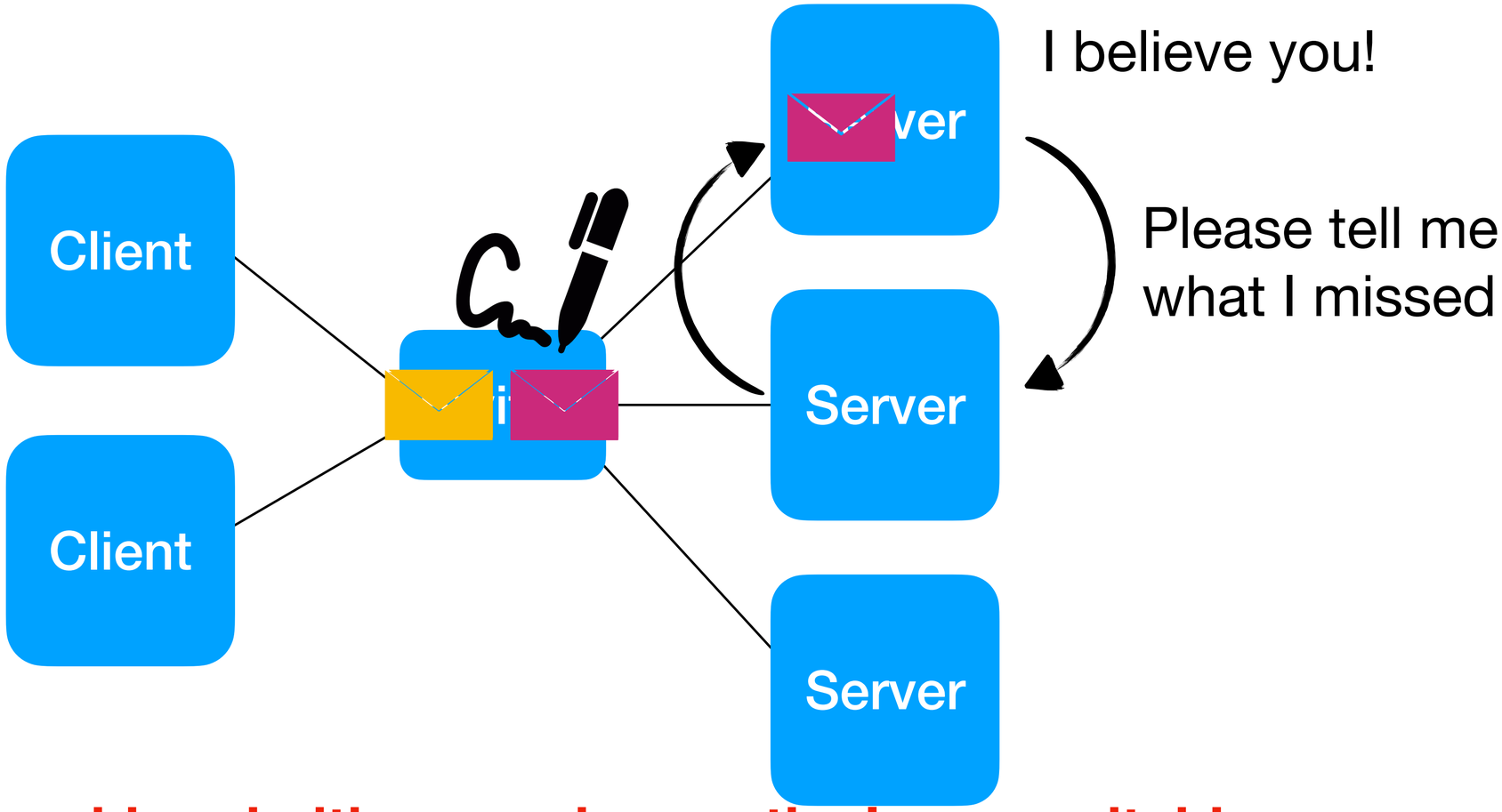
A centralized trusted entity can solve all our problems!

Memory-constrained



If we cannot store entire messages,
we cannot retransmit if messages get dropped!

Take 2: Sign Messages



Cryptographic primitives are impractical on a switch!

Our Results

Trusted programmable switches
can speed up and increase fault tolerance of BFT protocols

✓ What programmable switches can and cannot do

- Our protocol - SwitchBFT
- Evaluation

SwitchBFT: Design Principles

Authenticate packet sources → **Rely on network links instead of cryptography**

Sequence client mes → Trusted Switch → **Commit messages in a single RTT (fast path)**

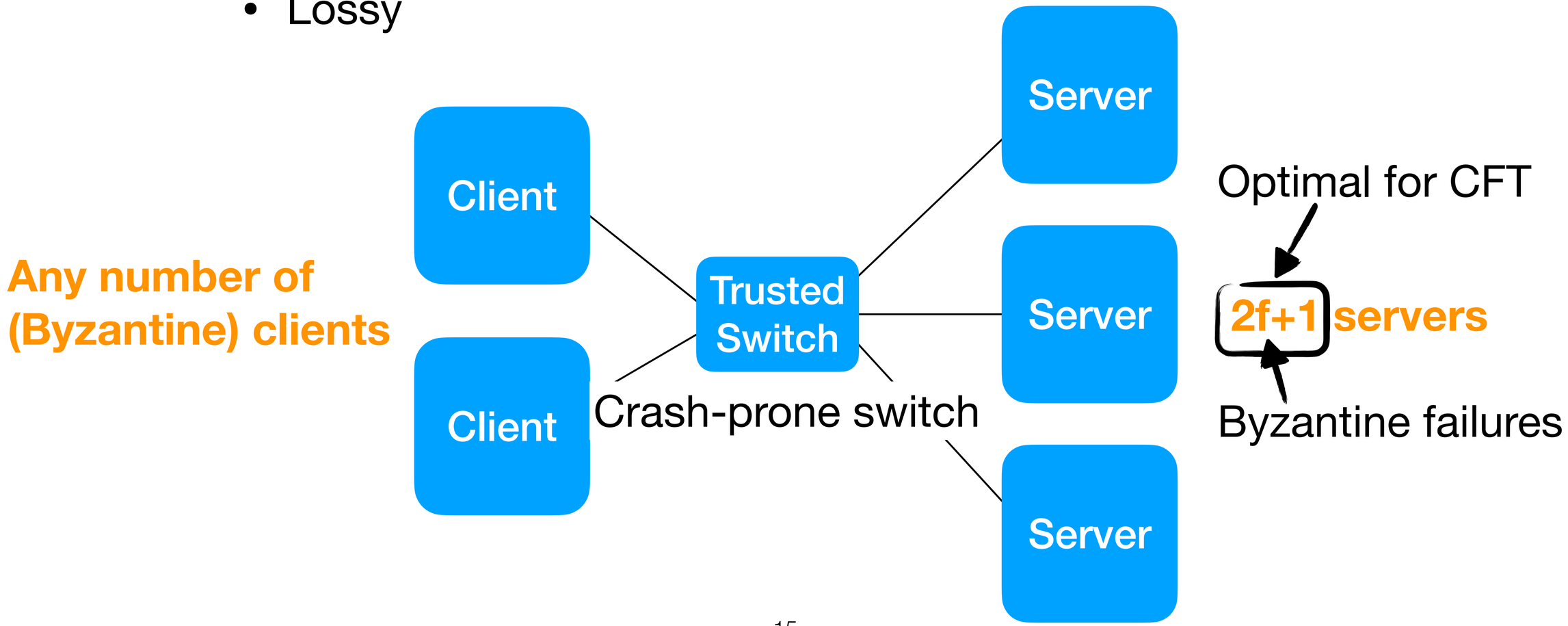
Ensure safety via protocol verification → **Robust performance even when some replicas are faulty**

Same performance and fault tolerance as CFT without signatures or cryptography in the switch

Setting and Assumptions

Network:

- Partially synchronous - don’t know message delay
- Lossy

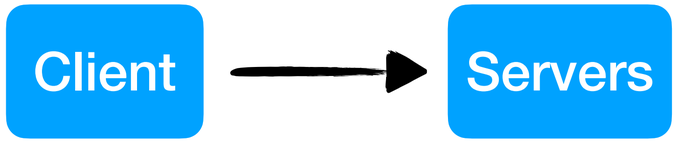


High Level Protocol

- Part 1: Ensure messages arrive in the correct order, with **possible gaps**
- Part 2: Fill gaps

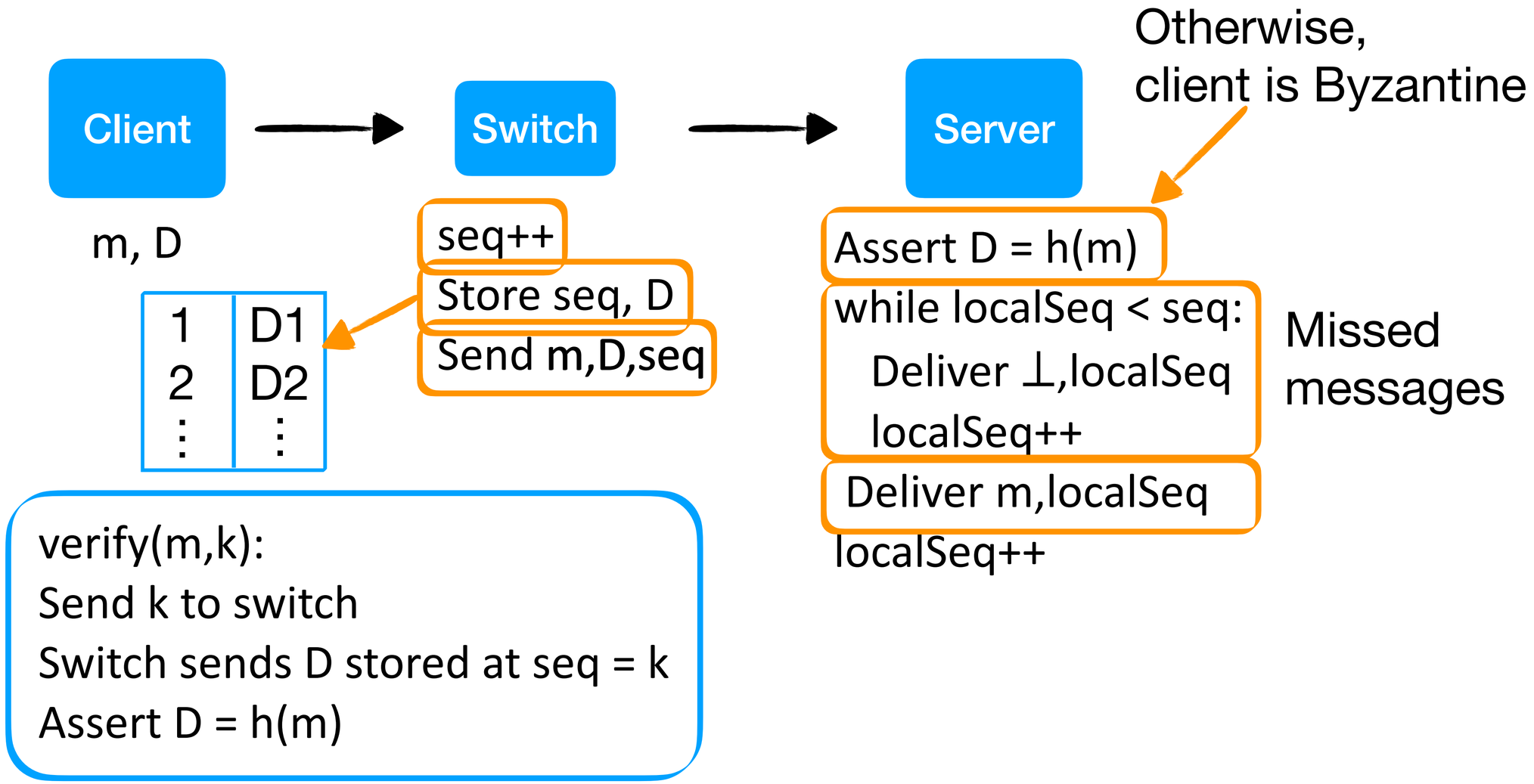
Part 1: VUAB

Verifiable Unreliable Atomic Broadcast



- **Validity:** In an execution without message drops, **everyone receives every message** broadcast by clients exactly once
- **Loss detection:** If **any** server receives a message m broadcast by a client, then **all servers** receive either **m or ⊥**
- **Total ordering:** If two servers receive a message in the **k^{th} round**, it's **the same message**
- **Verifiability:** **verify(m,k)** returns true if some server received m as its k^{th} message

VUAB — Protocol



High Level Protocol

✓ Part 1: VUAB — messages arrive in order and are verifiable, but with gaps

- Part 2: Fill gaps

Handling Message Loss

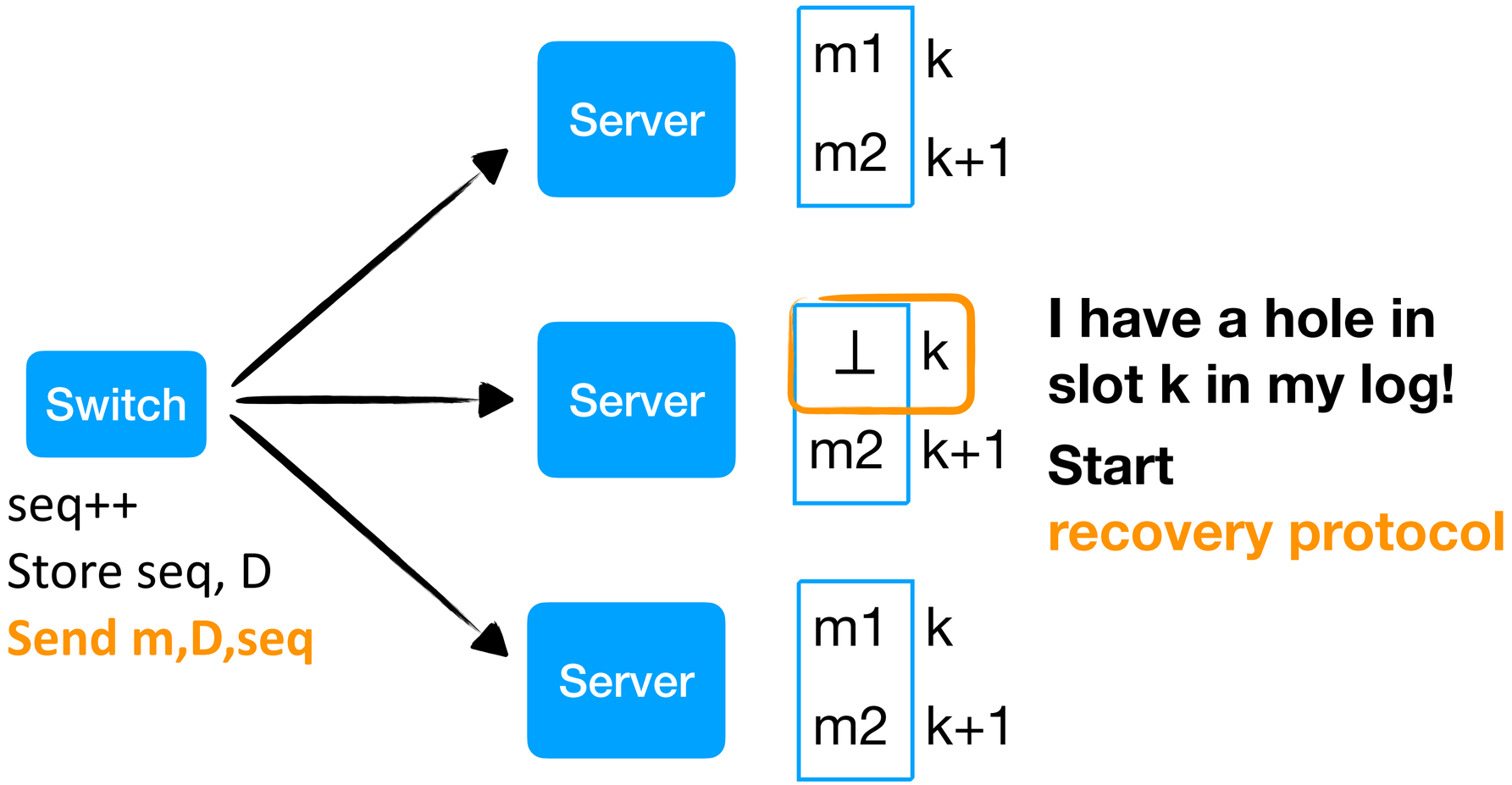
m, D

seq++
Store seq, D
Send m,D,seq

Assert m = h(D)
while localSeq < seq:
  Deliver ⊥,localSeq
  localSeq++
 Deliver m,localSeq
localSeq++

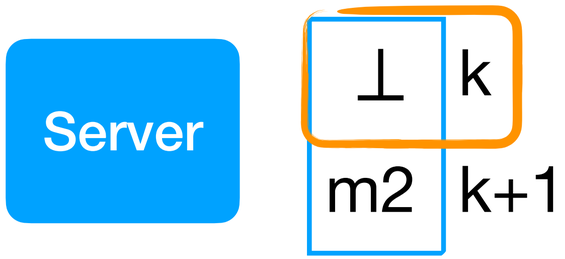
Problem: Since switch does not store messages, it **cannot retransmit** what it forwards from the client

Handling Message Loss



Servers must **agree** on their **entire log**

Handling Message Loss



I have a hole in slot k in my log!

Start recovery protocol

Problem: some servers might succeed in recovering while others do not

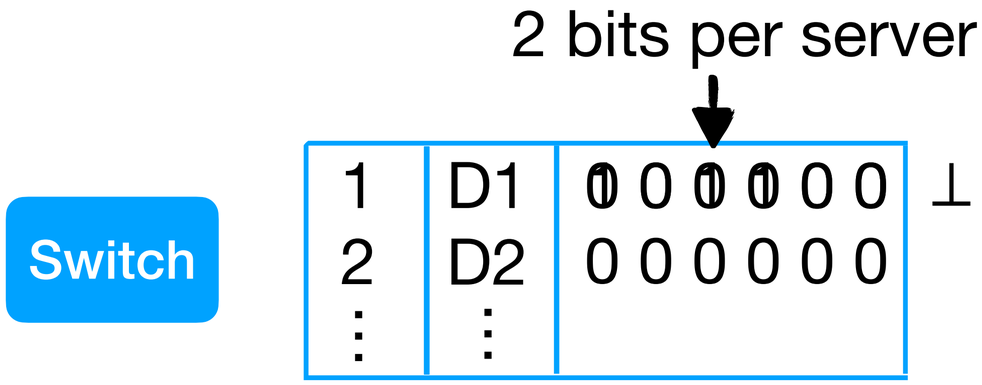
Basic Idea:

- Request message in slot k from all servers
- If any of them sends back a message m, verify(m,k) using VUAB
  - If returns true, adopt that message
- If receive at least f+1 ⊥ replies, stop trying to recover

Handling Message Loss

Problem:
some servers might succeed in recovering while others do not

Solution:
Rely on switch to ensure consistent recovery output!



What has this server replied to recovery of this message?

0 - no reply yet

1 - replied ⊥

2 - replied non-⊥

Switch ignores repeated responses from the same server if they conflict

Once f+1 servers reply ⊥, switch locks decision, replies to requests directly

More Details in the Paper

- Switch failure handling
- Log trimming
- Formal correctness proof
- Optimizations and implementation details
- Scalability analysis

Implementation and Setup

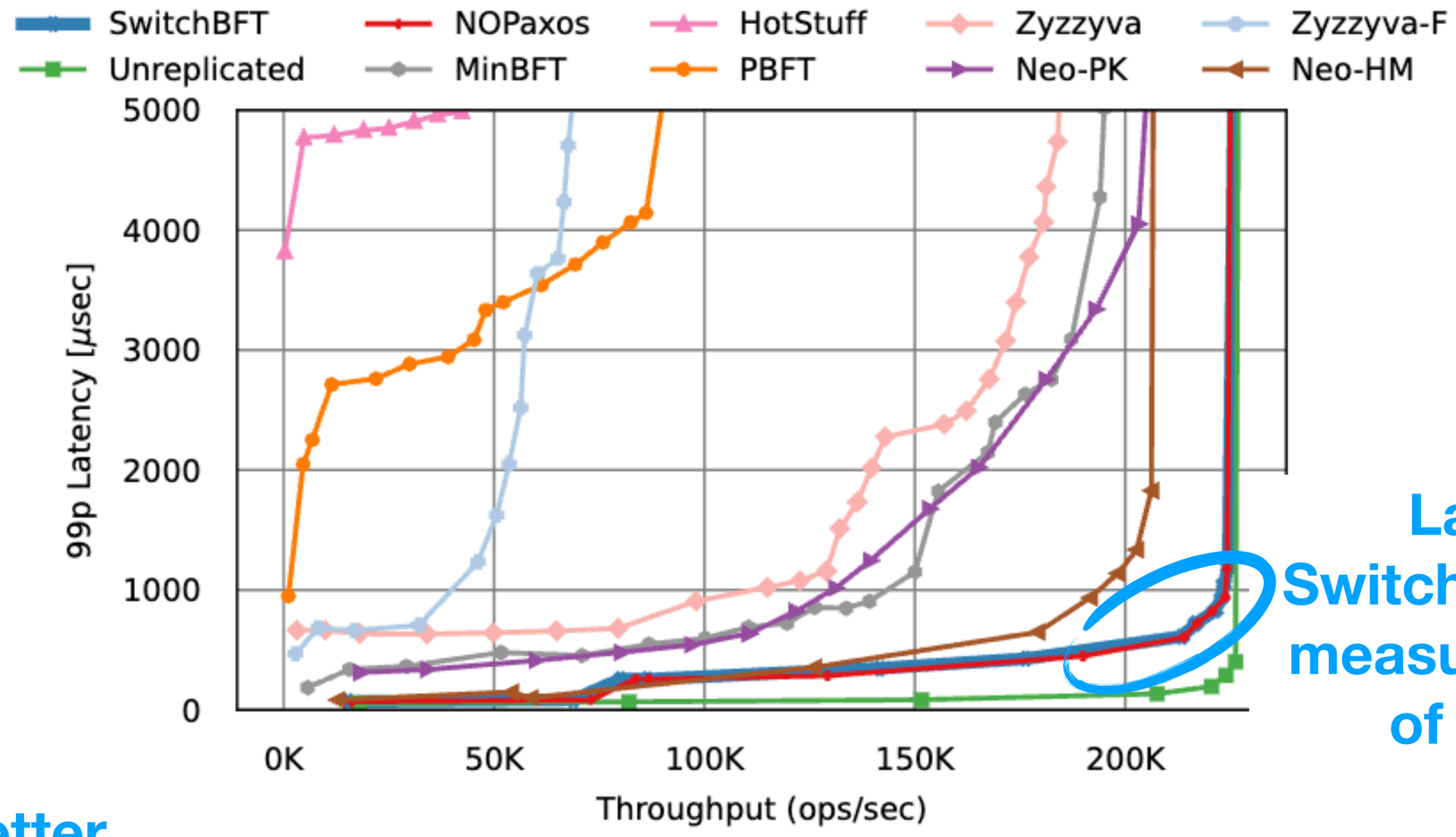
- Implemented on Intel Tofino switches using P4
- 2 Intel Xeon Silver machines, 16 cores each, running clients and servers
- Intel E810 100Gbps NICs

Baselines for Evaluation

- NOPaxos [LMSSP'16]: CFT replication with an in-network sequencer
- NeoBFT [SJKLL'23]: uses cryptography on the switch (with FPGA)
- MinBFT [VCBLV'13]: uses SGX
- Zyzzyva [KADCW'07], HotStuff [YMRGA'19], PBFT [CL'99]: BFT systems with no trusted hardware

Experimental Evaluation

End-to-End: Just replication

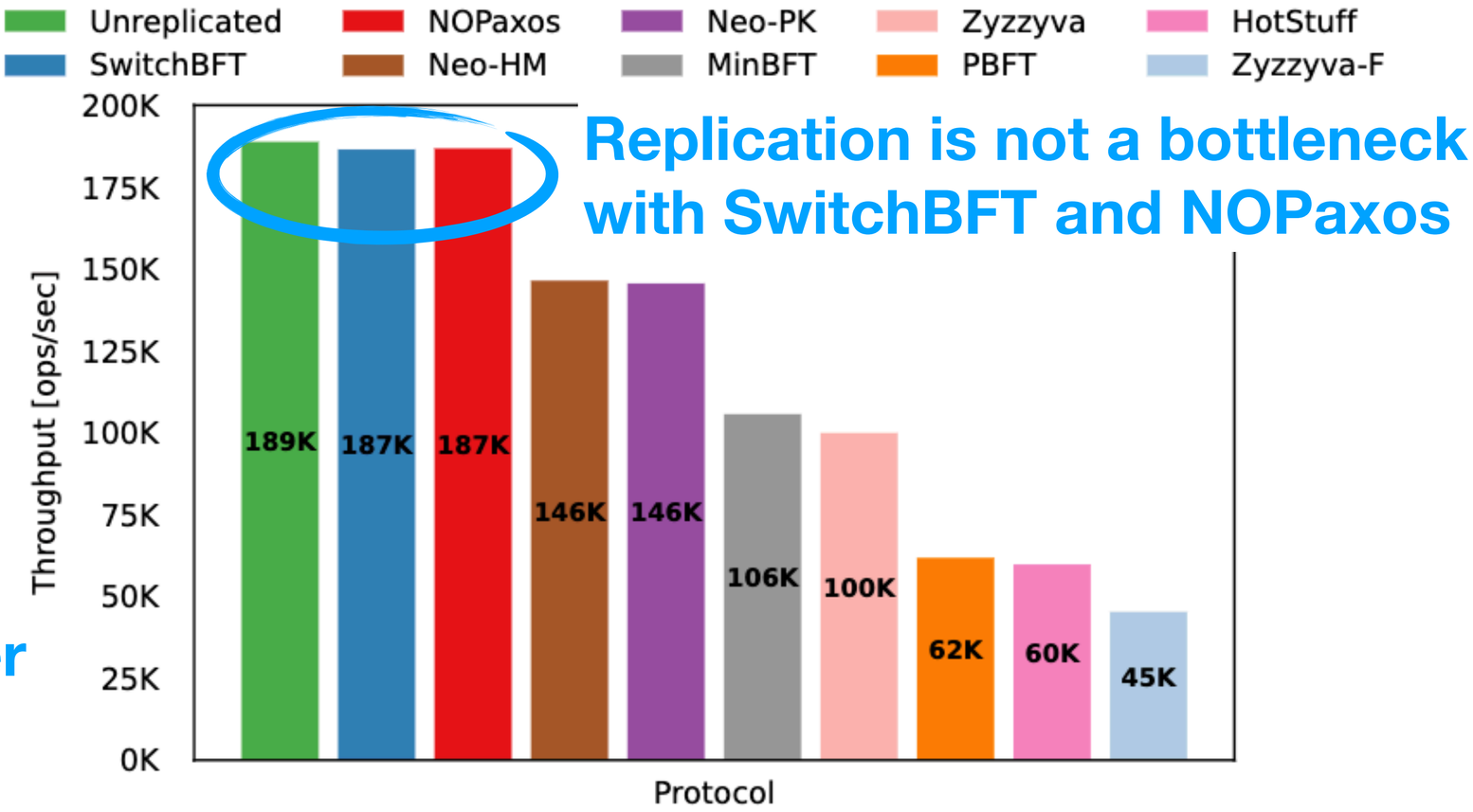


Latency of SwitchBFT is within measurement error of NOPaxos

This way is better

Experimental Evaluation

End-to-End: Replicated Key-Value Store

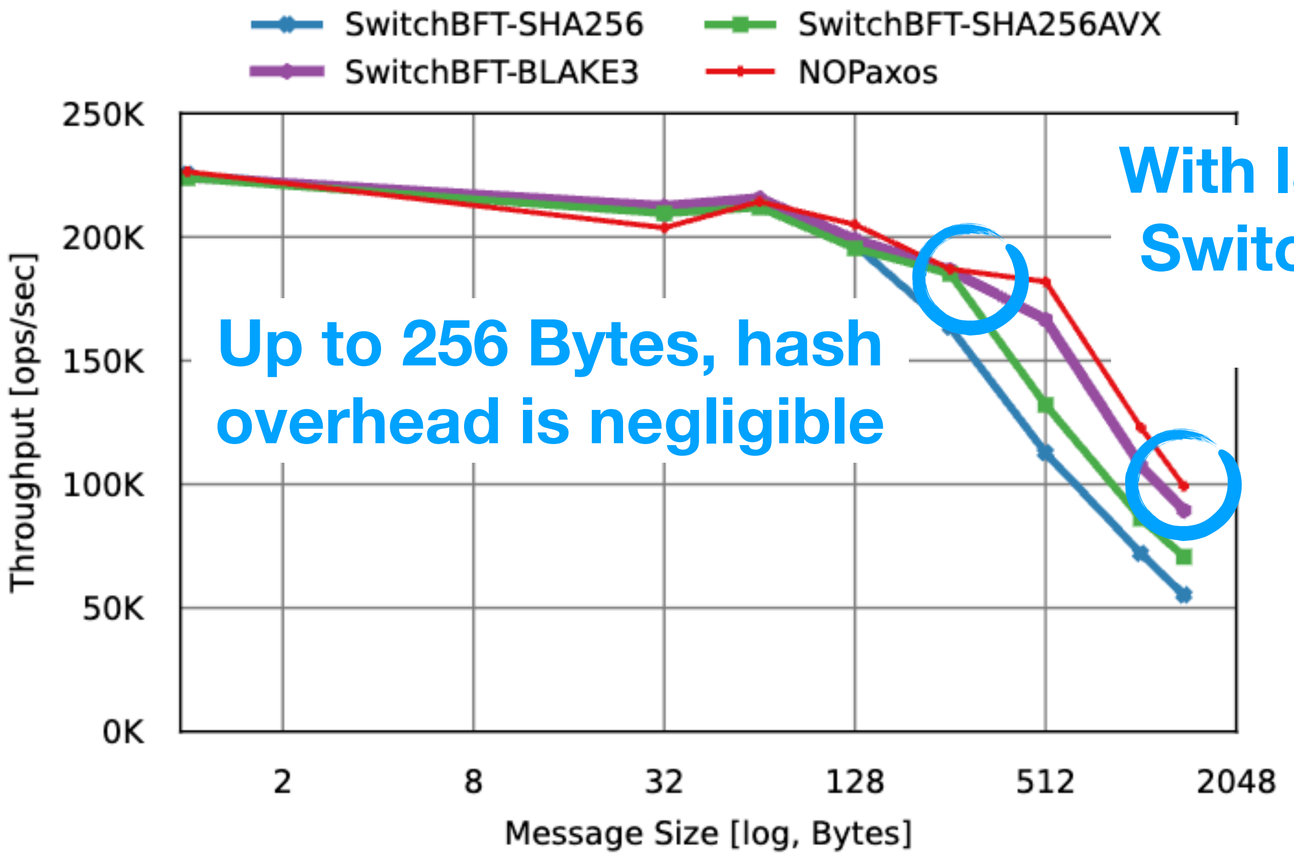


Replication is not a bottleneck with SwitchBFT and NOPaxos

This way is better

Experimental Evaluation

Varying Message Size and Hash Function



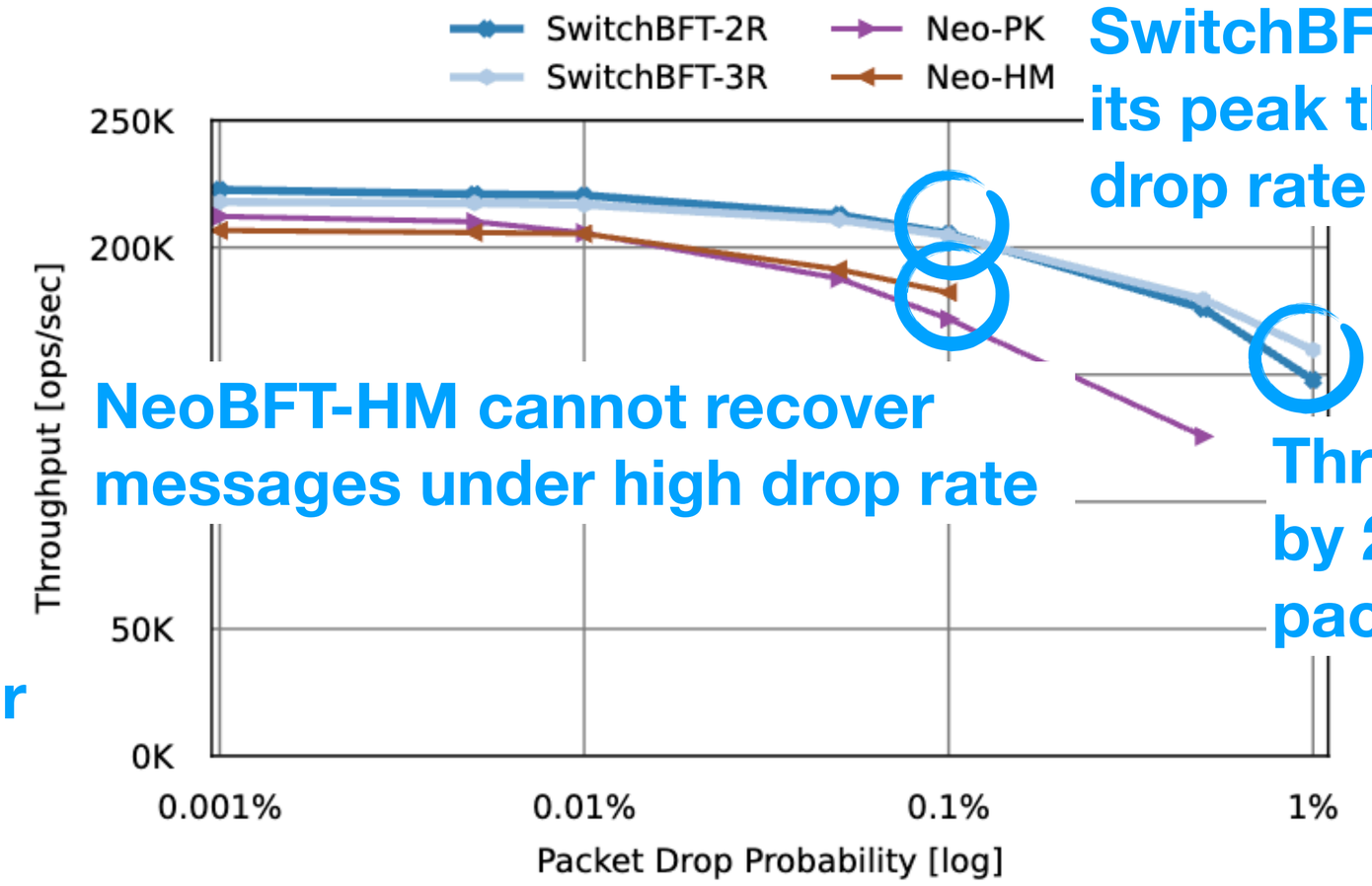
Up to 256 Bytes, hash overhead is negligible

With largest message size, SwitchBFT is 14% slower than NOPaxos

This way is better

Experimental Evaluation

Varying Packet Drop Rate



SwitchBFT maintains 93% of its peak throughput at 0.1% drop rate

NeoBFT-HM cannot recover messages under high drop rate

Throughput drops by 20% at high packet drop rate

This way is better

Summary

Trusted programmable switches
can speed up and increase fault tolerance of BFT protocols

- Programmable switches constitute good trust boundary for BFT systems
- SwitchBFT provides fast replication with 2f+1 fault tolerance and no signatures via ***packet source authentication*** and ***protocol verification***

Thank you!
Questions?

lzeno@nvidia.com