SwiSh: Distributed Shared State Abstractions for Programmable Switches

Lior Zeno, Dan R. K. Ports, Jacob Nelson, Daehyeok Kim, Shir Landau Feibish, Idit Keidar, Arik Rinberg, Alon Rashelbach, Igor De-Paula, Mark Silberstein
Stateful Packet Processing

Fixed-function Switch  
Fixed-function Switch  
Programmable Switch  
Programmable Switch  

Stateless processing  
Stateless processing  
Stateful processing  
Stateful processing  

P4
Current Trend: In-Switch Acceleration

SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs [SIGCOMM 2017]

Offloading Real-time DDoS Attack Detection to Programmable Data Planes [IM 2019]

Heavy-Hitter Detection Entirely in the Data Plane [SOSR 2017]

Cheetah: Accelerating Database Queries with Switch Pruning [SIGMOD 2020]

Designed for a single-switch
The Case for Many-Switch Designs

Scalability

Availability

Locality

Not all information is available on all switches
Example: Reactive Applications (DDoS detector)
Example: Reactive Applications (DDoS detector)
Challenge: Network-Wide DDoS Detector

Mismatch between the control and data plane processing rate

An ad-hoc solution
Our Work: Data Plane Replication

Sketches are replicated entirely in the data-plane with provable consistency guarantees.

Data-plane replication opens the door for new in-switch application designs.
Agenda

• The case for data-plane replication
• SwiSh design and challenges
• Experimental results
SwiSh Design

- Reusable APIs for application developers
- 3 different consistency levels for shared variables

One big switch
In-Switch Replication Protocols

- **Strong Read-Optimized (SRO)**
  - NAT
- **Eventual Write-Optimized (EWO)**
  - Rate limiter
- **Strong Delay-Writes (SDW)**
  - Sketch-based applications
SDW Challenges

• **C1:** What is the most suitable consistency level for replicating sketches?

• **C2:** How to deal with packet drops?
C1: Consistency vs. Performance

How efficient can the protocol be?

What is the right consistency level?
C1: Consistency vs. Performance

Consistency level

Weak

Inconsistent

Switches have different views of the sketch

High bandwidth overhead

Updates
C1: Consistency vs. Performance

Consistency level

Strong

Switches cannot apply updates concurrently

High latency overhead

Chain Replication
Solution: Strong Delayed-Writes (SDW)

- Consistency level: r-relaxed strong linearizability
- Provably correct
- Provable precise error bounds for sketches
- Low latency
- Constant #replication messages
- SDW protocol
SDW Protocol

- Read
- Update
- Sync
SDW Protocol

Reads and writes are applied locally

Read

Update

Sync
SDW Protocol

Window id = 0

Sync

Round-based protocol

Window id = 0

Sync
SDW Protocol

Window id = 0
Sync

Updates

Window id = 0
Sync
SDW Protocol

Window id = 0
Sync

Sync
Window id = 0

Updates

ACK
C2: Dealing with Packet Drops

Common solution: implementing reliable delivery over an unreliable network
C2: Packet Buffering is Expensive
Solution: Reproducible Updates

What if we already merged updates from other switches?

Rebuild updates from the buffer
Solution: Reproducible Updates

Sync-Source

Sync-Merge
SDW Protocol

Once all updates and ACKs are received we can slide the window.
Efficient Register Swapping
In the paper...

• Theoretical proof of SDW consistency guarantees
• Recovery protocols
• Asymmetric topologies
  • Ready phase
• SDW design
• Eventual Write-Optimized (EWO)
  • Eventual consistency (low read/write latency)
• Strong Read-Optimized (SRO)
  • Strong consistency
Evaluation

• Three real-world application:
  • NAT
  • Rate limiter
  • DDoS Detector

• Microbenchmarks and scalability analysis

• Recovery time
Evaluation
Super-spreader Detector

#(S, dst) > 1K -> Block

Sends 10K packets with the same source IP to different destinations

We measure how many packets are received
Push Design

#(S, dst) > 0.5K -> Notify(S)

#Updates(S) == 2 -> Block(S)
Pull Design

#(S, dst) > 0.5K -> Store(S)

#Occurrences(S) == 2 -> Block(S)
Data Plane-Only Design

SDW replicates the sketch
Super-spreader Detector: Results

More packets are passing

Received / Sent (%)

Number of DDoS sources per second

Ideal

More sources to process

More sources to process
Super-spreader Detector

Packet drops at the controller

The controller blocks sources that won’t be used in the future

SDW performs ideally
Conclusions

• Data plane replication is essential for reactive in-switch applications
• SwiSh provide reusable APIs for building distributed in-switch applications
• SwiSh provides a provably correct SDW protocol for sketch replication
• SwiSh is practical, performant and fault tolerant
• Rethink distributed in-switch applications design

Thank you!
Questions?
liorz@campus.technion.ac.il