Closed-loop Network Performance Monitoring and Diagnosis with SpiderMon

Weitao Wang, Xinyu Crystal Wu, *Praveen Tammana, Ang Chen, T. S. Eugene Ng

Root Causes of Performance Problems are Unpredictable

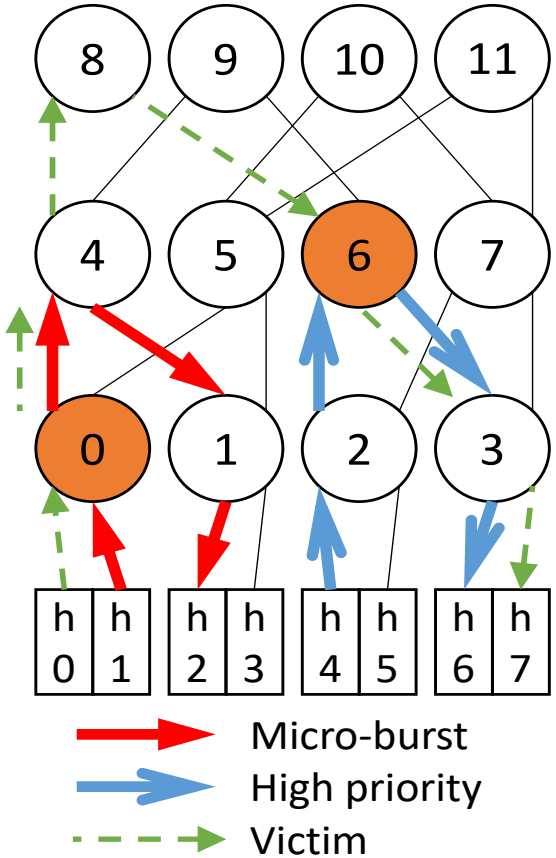
Sporadic	Network-wide	Transient
Random flow suffers at different locations	Problem may involve multiple switches	Last for a short time and disappear quickly

Root Causes of Performance Problems Are Also Diverse

Sporadic

Random flow suffers at different locations

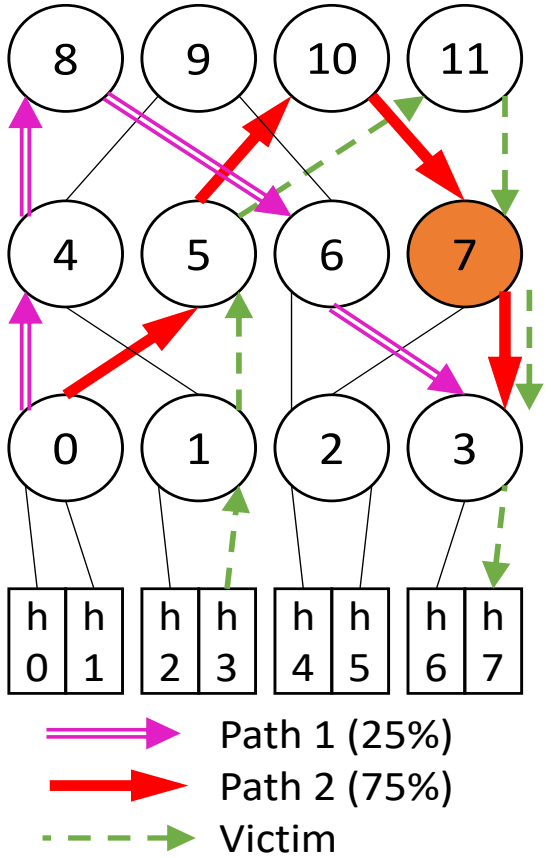
(Multiple congestions)



Network-wide

Problem may involve multiple switches

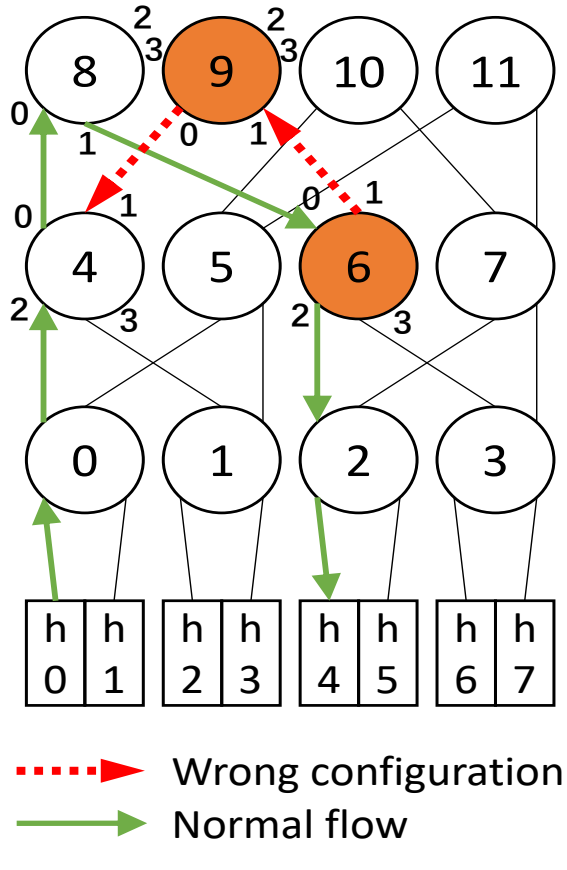
(ECMP load imbalance)



Transient

Last for a short time and disappear quickly

(Transient loop)



Existing Solutions Fall Short in Visibility / Overhead / Coverage

Host-based

Monitor and diagnose purely on the hosts **(Trumpet, Dapper, …)**

Lack of in-network information

In-Network

Always-on

Monitor the whole network at all time **(NetSight, PINT, …)**

High overhead for collecting and storing non-related information

Query-based

Add telemetry on some switches based on query **(Sonata, Marple, …)**

Low coverage due to pre-defined monitoring scope

Another Common Shortage: An Efficient Close-loop Diagnosis

Diagnosis procedure in the previous monitoring works:

- Problem-specific
- Collect data in a high-recall low-precision way
- High computational complexity ---- e. g., nested loops

An ideal diagnosis procedure:

- **Extensible and generalizable** for many different problems / root causes
- Collect data in a **high-recall high-precision** way
- **Low computational complexity**

Outlines

- SpiderMon Design
  1. Overview
  2. Problem detection
  3. Relevant telemetry data retrieval
  4. Root cause diagnosis
- Evaluation
  1. Diagnosis accuracy
  2. System overhead

SpiderMon Overview

Always-on monitoring:

- Monitor **every packet** on **every hop** in the data plane
- Maintain causality information (for spreading “spider” packet)
- Maintain historical telemetry data (reported after recognized as related info)

SpiderMon Overview

Always-on monitoring:

- Monitor **every packet** on **every hop** in the data plane
- Maintain causality information (for spreading “spider” packet)
- Maintain historical telemetry data (reported after recognized as related info)

Problem triggered

Find related telemetry data:

- Spread the “Spider” packets
- Report related telemetry info

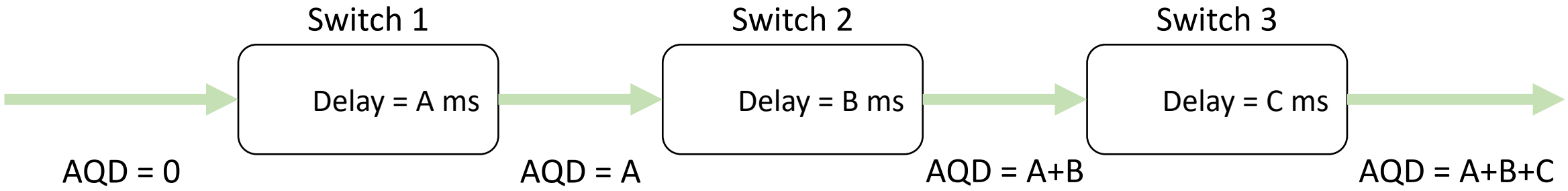
Root cause analysis:

- Analyze related telemetry info
- Diagnose the root causes

Monitoring and Problem Detection

Excessive accumulated queuing delay (AQD header):

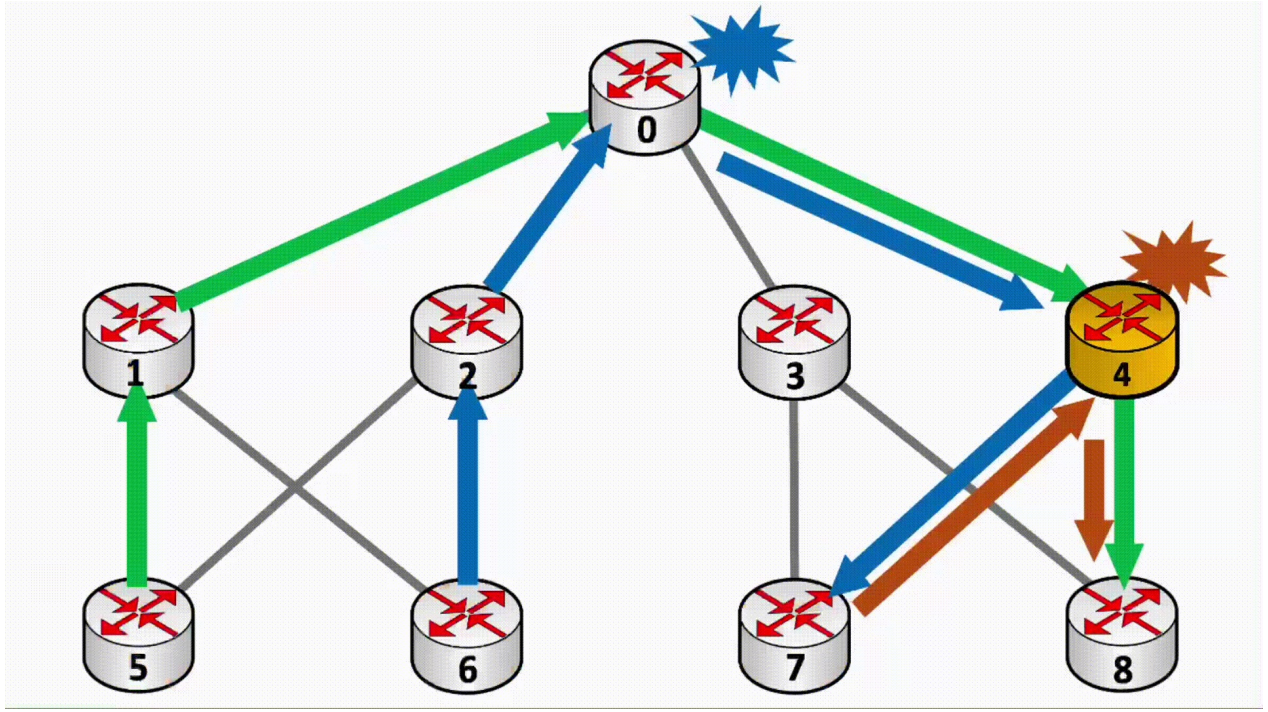
- **Immediate trigger and high coverage**
  - Monitor every packet on every hop in the data plane
- **Fixed low network overhead**
  - Fixed small additional header length in despite of path length
- **Transparent to hosts**
  - Added to each packet at the source ToR, remove at the destination ToR
- **Tunable thresholds**
  - Assign different thresholds for different QoS

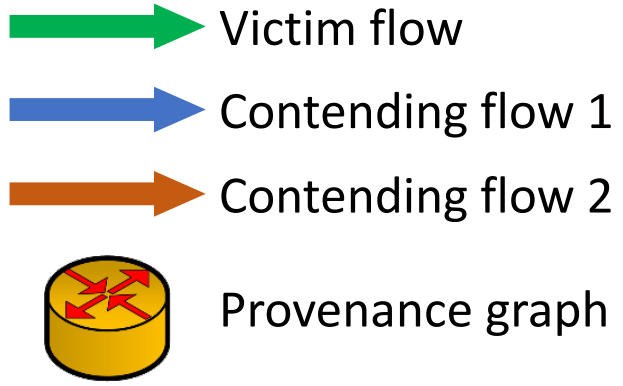


Telemetry Collection ---- “Spider” Attack!

Determine the **provenance graph** for the detected problem:

- Include every switches that may **“cause”** the problem
  - Switches on the victim’s historical path
  - Switches that sent excessive data and contended with the victim
- Minimal overhead: only related telemetry data are collected



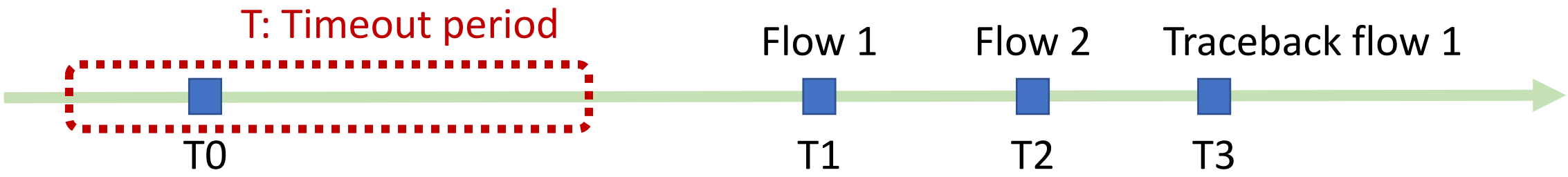


Causality Data Structures For Spreading ”Spider” Packets

Trace-back victim’s path:

- **Timeout bloom filter**
  - Record timestamp in each bucket
    - If (curr_ts – bucket_ts < T) {return true;}
  - All the items before **curr_ts-T** will be removed automatically

- Normal bloom filter
  - Turn 0 into 1 in each matched bucket
    - If (bucket == 1) {return true;}
  - Cannot remove items → low precision

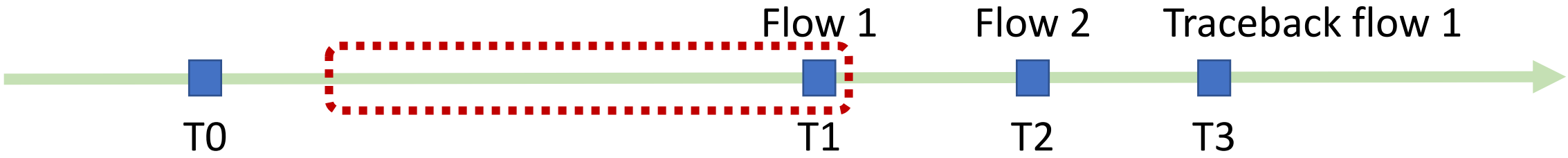


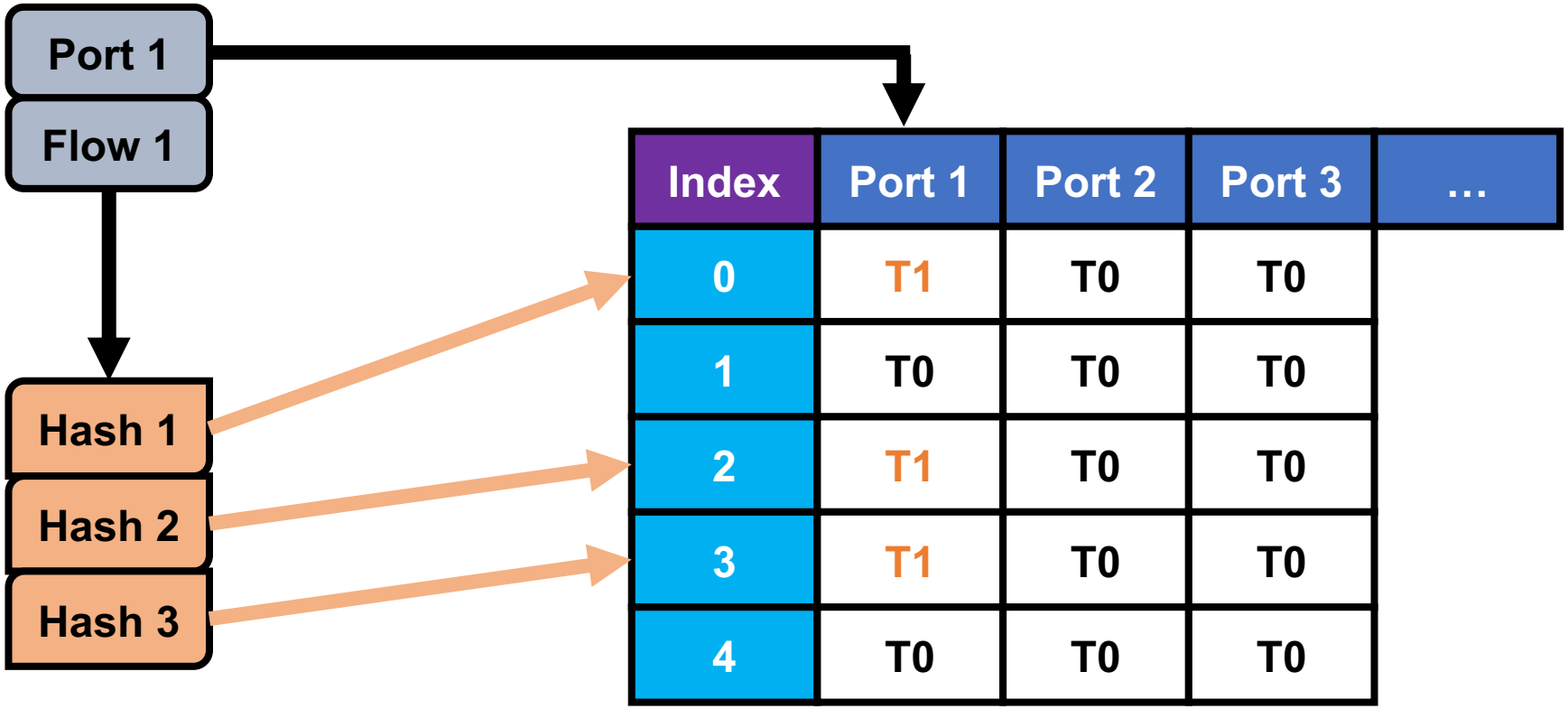
Hash 1

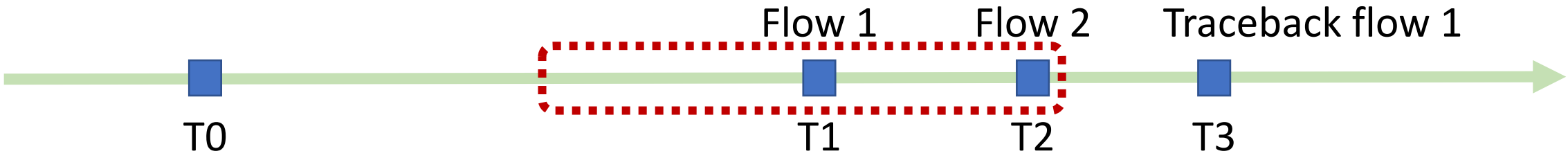
Hash 2

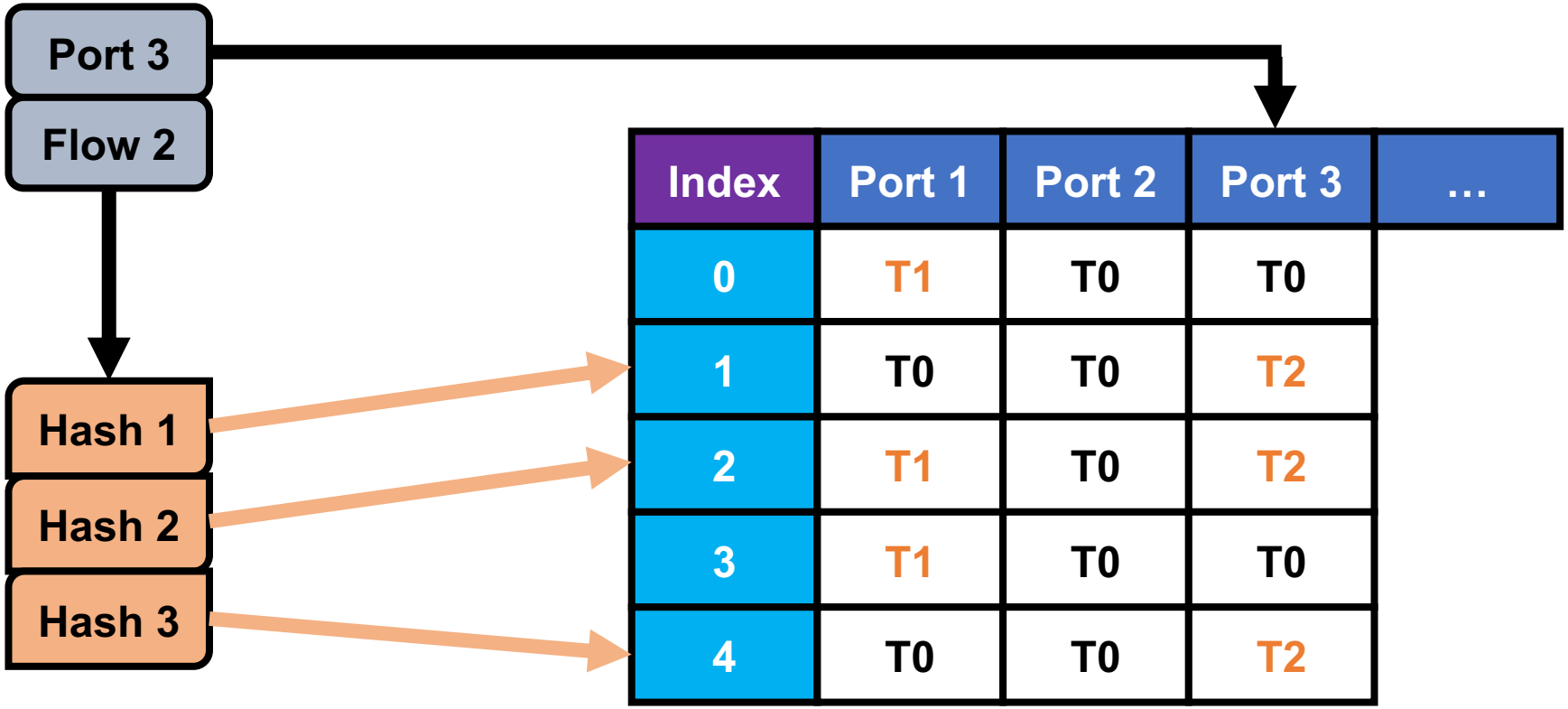
Hash 3

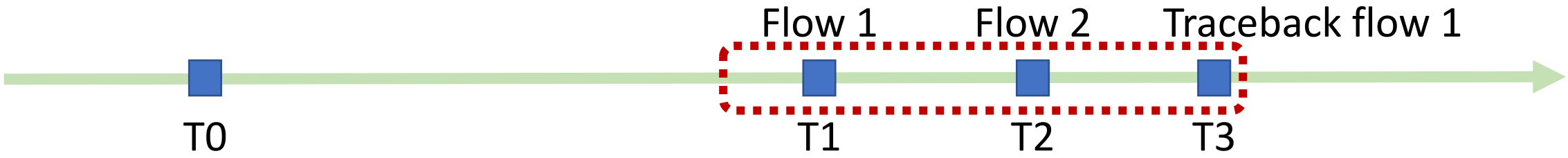
Index	Port 1	Port 2	Port 3	…
0	T0	T0	T0	
1	T0	T0	T0	
2	T0	T0	T0	
3	T0	T0	T0	
4	T0	T0	T0	


Flow 1
Flow 2
Traceback flow 1
T0
T1
T2
T3

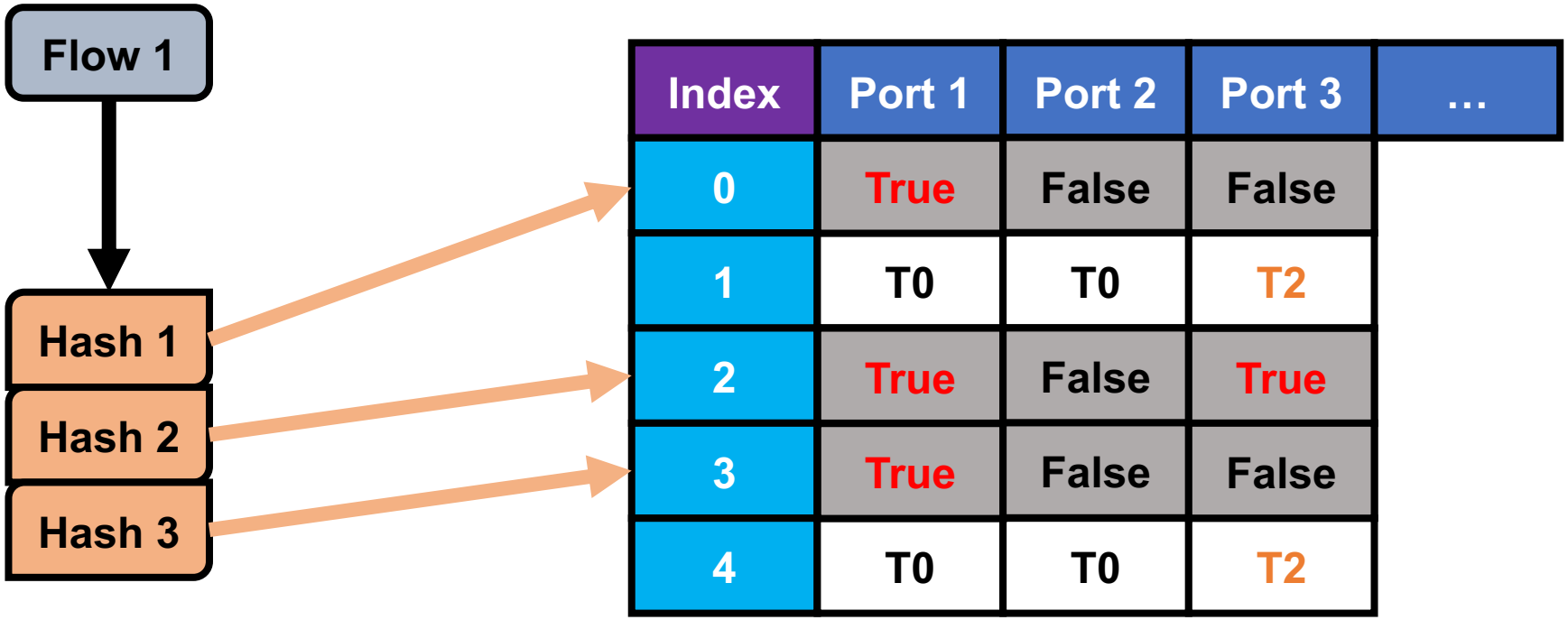

Port 1
Flow 1
Hash 1
Hash 2
Hash 3
Index
Port 1
Port 2
Port 3
…
0
T1
T0
T0
1
T0
T0
T0
2
T1
T0
T0
3
T1
T0
T0
4
T0
T0
T0


Flow 1
Flow 2
Traceback flow 1
T0
T1
T2
T3


Port 3
Flow 2
Hash 1
Hash 2
Hash 3
Index
Port 1
Port 2
Port 3
…
0
T1
T0
T0
1
T0
T0
T2
2
T1
T0
T2
3
T1
T0
T0
4
T0
T0
T2



Only port 1 return **True** for all buckets



Index	Port 1	Port 2	Port 3	…
0	True	False	False	
1	T0	T0	T2	
2	True	False	True	
3	True	False	False	
4	T0	T0	T2	

Causality Data Structures For Spreading ”Spider” Packets

Search related branches

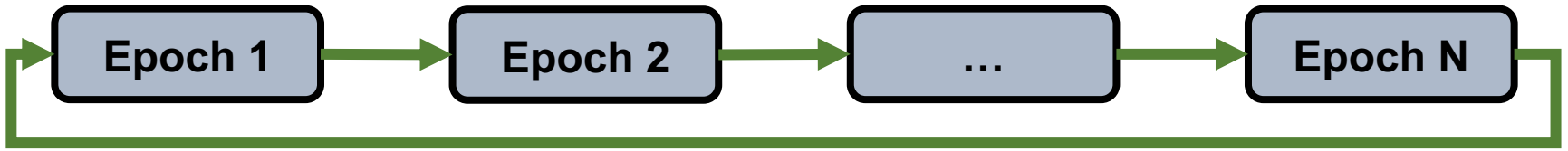
- Per-port traffic meter (more details in the paper)

Telemetry Data Collection

From every switch covered by “Spider” packets, we collect:

Per-epoch per-flow data in a circular buffer

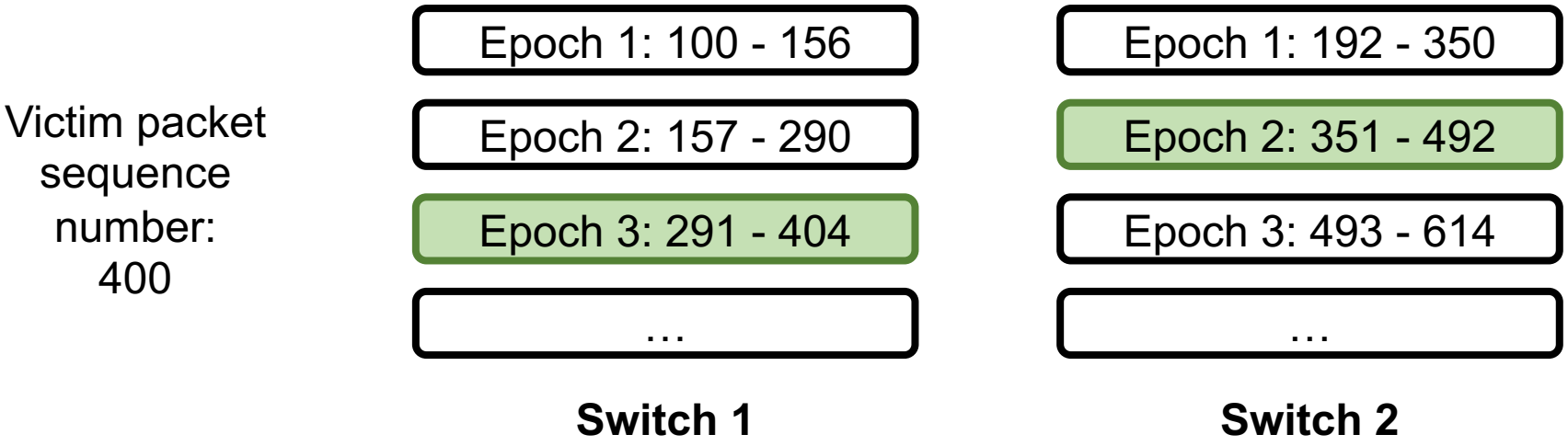
1. 5-tuple
2. Sequence number range
3. Traffic volume / packet counts
4. Queue depth
5. Flow priority
6. In-coming / outgoing port
7. ……



Telemetry Data Alignment

From every switch, we only use data from **one** relevant epoch to improve accuracy:

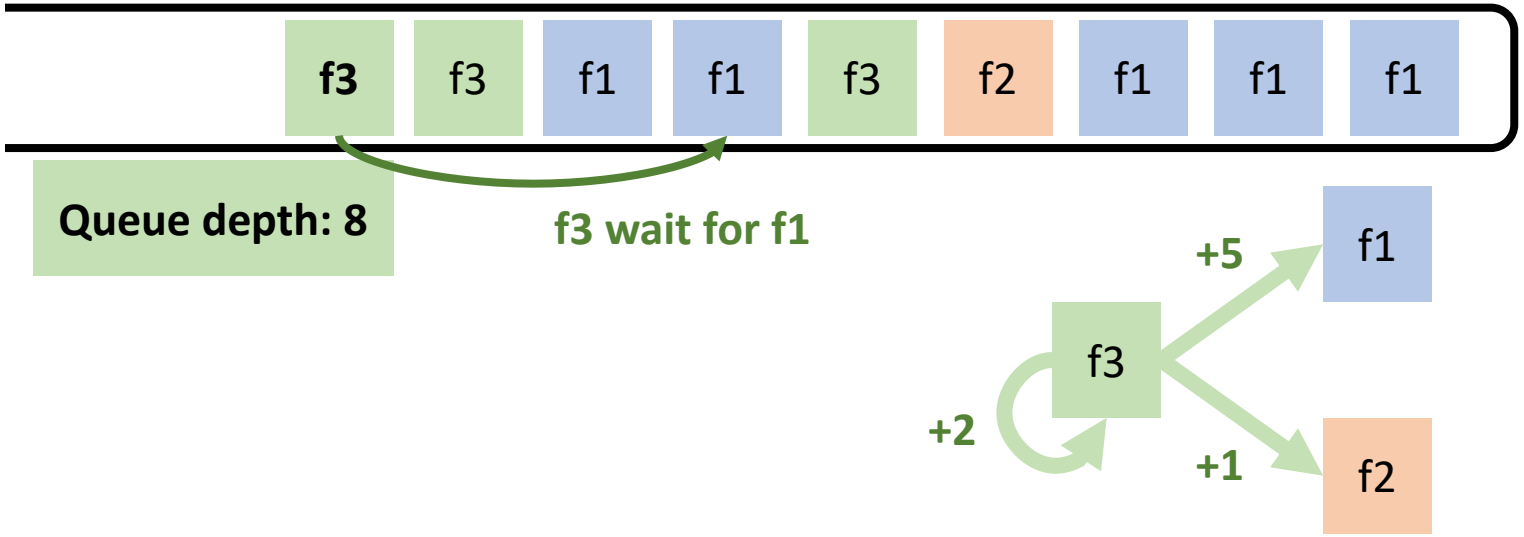
- Problems:
  - Switches' clocks are not synchronized
  - Switches process the same packet at different time
- Solution:
  - Use **sequence number** to align the telemetry data



Identify Root Causes with Wait-for Graph

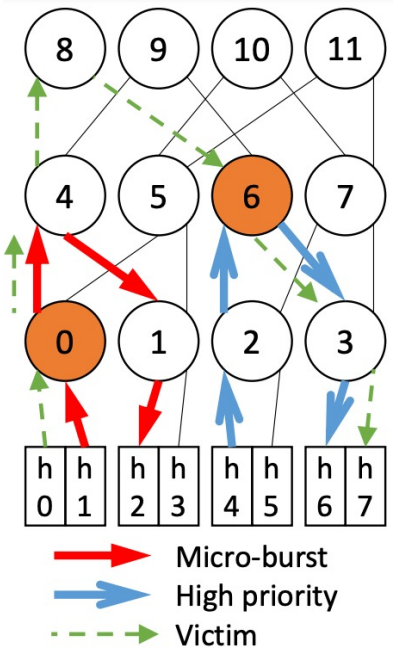
1. For every switch, replay the queue approximately based on the telemetry info:

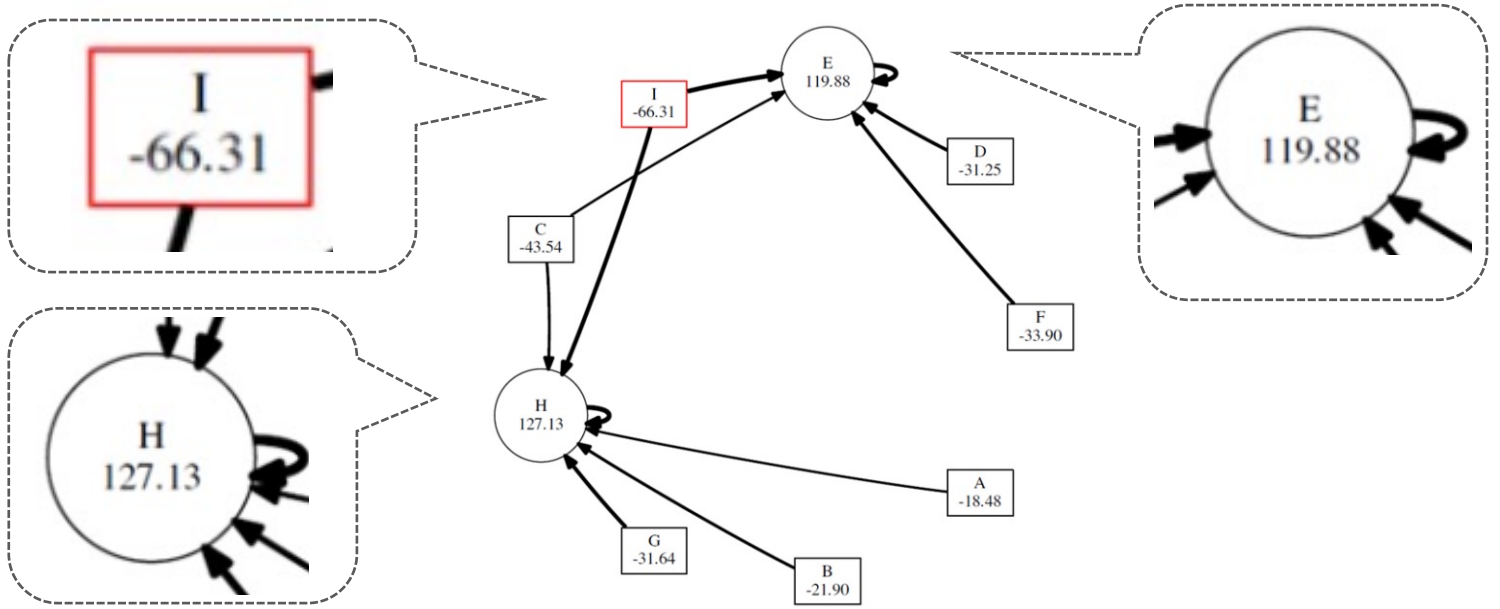
2. Parse the wait-for relation:



Identify Root Causes with the Wait-for Graph

3. Find the potential root causes with positive degree:
(degree = incoming edge weights – outgoing edge weights)





4. Determine the root causes by leveraging additional info as appropriate:
Flow priority; other flow's throughput on the same path; more details in the paper.

Evaluation Setup

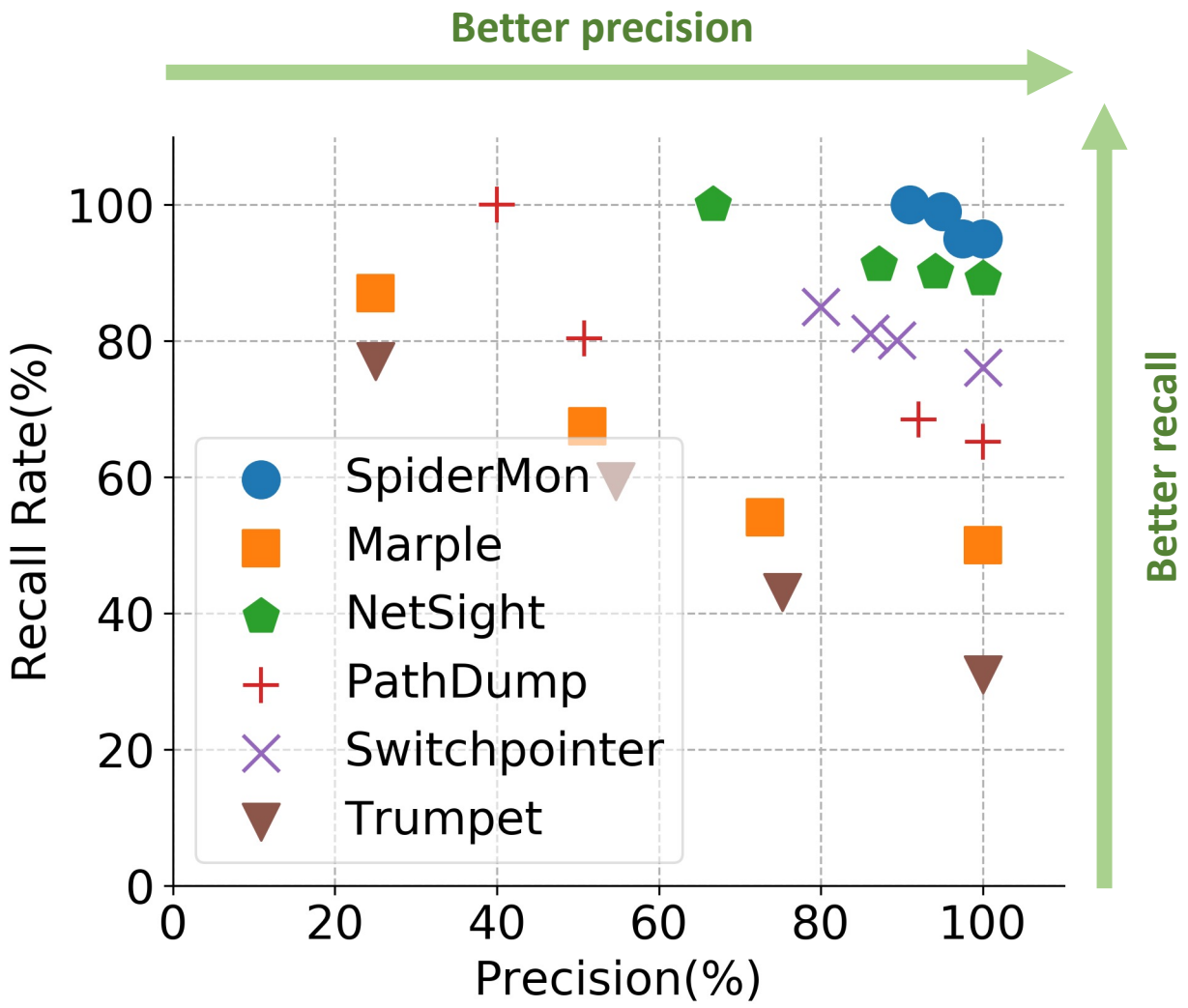
- **Simulator**
  - BMv2 Software Switch + NS3 simulator
  - 945 lines of P4 code
  - 52 instances from CloudLab
    - 8- core 2.0 Ghz CPU and 32 GB RAM
  - Fat-tree topology with 20 switches and 32 servers
    - 8 ToR switches, 8 aggregation switches, 4 core switches, 4 server per ToR
- Hardware Switch prototype
  - Barefoot Tofino switches
    - 32 ports, 2 pipelines
  - 1147 lines of P4-Tofino code

Diagnosis Effectiveness: Compare with Existing Solutions

SpiderMon has the best precision & recall:

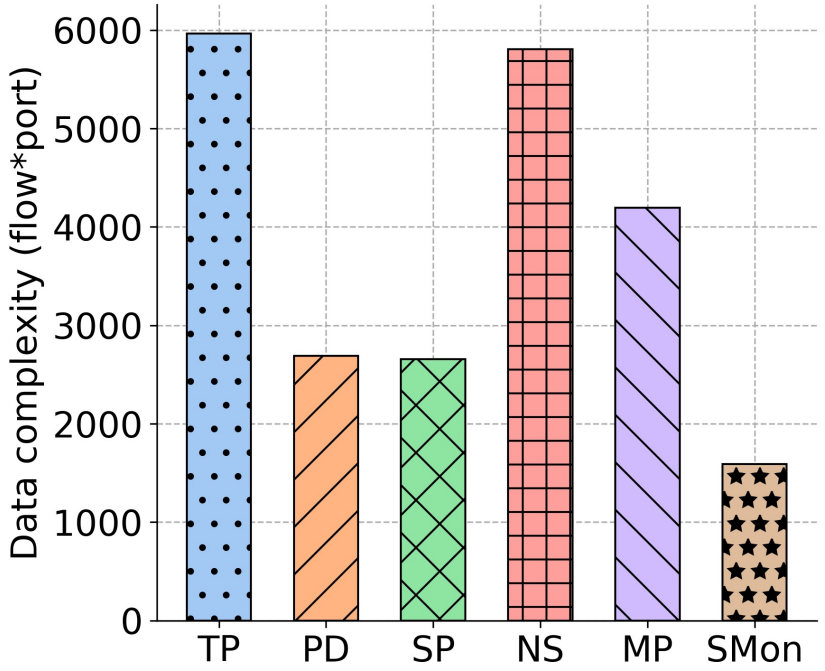
1. Collect packet-level information;
2. Only within a short time interval;

- **Trumpet** can only infer in-network condition based on the host-side info;
- **Marple** has delay to enable queries on the related switches;
- **Pathdump** and **SwitchPointer** lack of queueing information;
- **NetSight** infer the packet “postcard” order by topology information

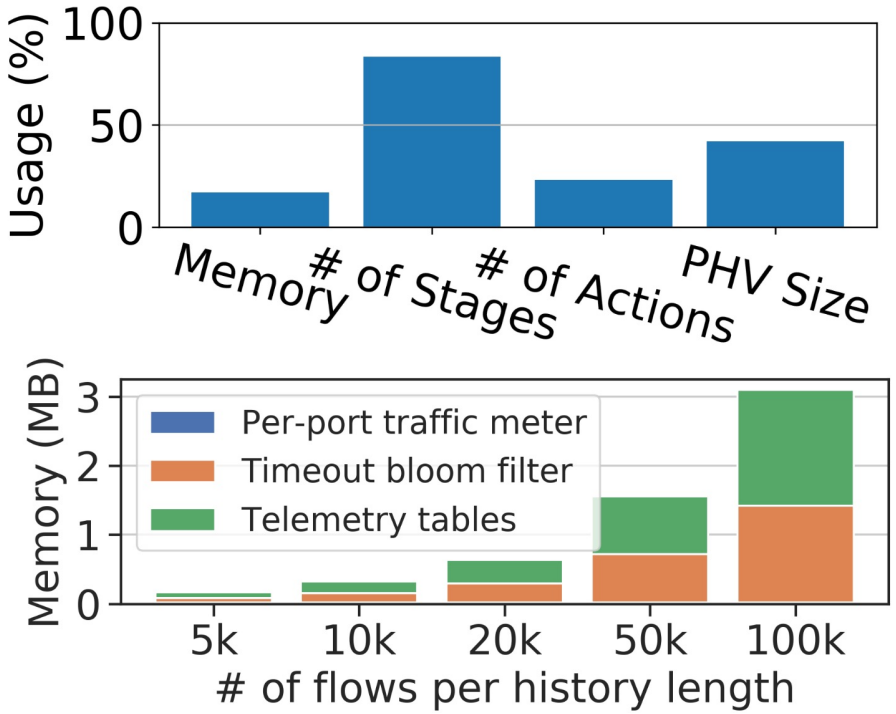


Multiple Micro-burst Scenario

Data Complexity Overhead and Switch Memory Overhead



SpiderMon collects and processes the least amount of data because it selects the related switches only.



SpiderMon's memory usage is affordable even when the flow concurrency is very high

Summary

SpiderMon is a low-overhead high-accuracy closed-loop monitoring and diagnosis system

- Network-wide always-on monitoring
- Trigger-based selective information collection
- Efficient and generalizable diagnosis
- Achieve more accurate root causes analysis with less overhead.

Summary

SpiderMon is a low-overhead high-accuracy closed-loop monitoring and diagnosis system

- Network-wide always-on monitoring
- Trigger-based selective information collection
- Efficient and generalizable diagnosis
- Achieve more accurate root causes analysis with less overhead.

Thank you!

(wtwang@rice.edu)