Simplifying Datacenter Network Debugging with PathDump

Praveen Tammana, University of Edinburgh. Rachit Agarwal, Cornell University. Myungjin Lee, University of Edinburgh

Presented by Luca Campanella
Motivated by the **Telemetry** paper initial arguments

Short recap of motivation:
- complexity of datacenters
- difficulty in debugging
Goal

- Partition debugging effort between network switches and end hosts
- Different than existing tools:
  - No dataplane snapshots, no per-switch per-packet logs, no packet mirroring, no packet sampling, no dynamic rule installation
- “Our goal is **not to beat existing [in-network] tools**, but to help them focus on a smaller set of nails (debugging problems) that we need a hammer (debugging techniques) for.”
The idea

- PathDump design based on tracing packet trajectories
- Three main players:
  - Switches embed their identifier into packet header
  - Edge devices record the list of switch identifiers in the packet header on a local storage and query engine
  - A query server can slice and dice entries across multiple edge devices
Switches: tracing packet trajectories

- Problem: packet header space is a scarce resource
  - Use VLAN tags (what if already used?)
  - At in-line speed only two VLAN tags (24 bits)

- Suggested solution: CherryPick
  - Previous work, reviewed to keep the paper self contained
  - Most frequent datacenter network topology are well structured
  - No need to save all the switch IDs
  - Does not work with arbitrary topologies
End-hosts

- Upon arrival save packet info in trajectory memory
- Wait for FIN, RST or 5 seconds timeout
- Reconstruct path using topology knowledge
- Save information inside TIB
- Reply to queries looking in TIB information
End-hosts

- Problem?

- Information relevant for the query may still be inside trajectory memory and not yet in TIB

- Solution: IPC channel allows query to look inside trajectory memory
  - How efficient and complete if in trajectory memory paths are not constructed yet?
Aggregator (Controller)

- Installs flow rules on switches that append the link IDs in the packet header (once)
- Runs debugging applications / queries
  - On demand from user
  - Triggered by an Alarm(...) query installed before
- Contacts single end-hosts to reply to queries
Example applications

- Path conformance check
- Load imbalance diagnosis
- Silent random packet drop
  - Implemented using the MAX-COVERAGE algorithm
- Blackhole diagnosis
- Rooting loop debugging
  - A packet carrying more than two tags is automatically directed to the controller
- TCP performance anomaly diagnosis

(a) Recall
Performance of silent random packet drop debugging algorithm

(b) Precision
System Evaluation

- Tested on 28 physical servers, 4 docker containers per server (112 TIBs)
- Average end to end response time of top 10000 flows query: ~2 seconds with aggregated queries
- 4% throughput loss compared to vanilla DPDK vSwitch (already suffers throughput degradation)
- CPU usage at end hosts: ~25% of one core cycles
- 10 MB of RAM, 110 MB of disk
Conclusion - PathDump

- **Pros:**
  - Doesn’t use many resources on the dataplane
  - Is a simple model

- **Cons:**
  - As declared by the authors, debugs only a subset of the problems
  - It may not work in case of arbitrary topologies or long paths
  - It can be slow to answer real time queries due to path reconstruction
  - Relies on modifying packet tags
Conclusion – The paper

- **Pros**
  - Makes a good argument for pushing debugging complexity to end hosts
  - Has good references to other papers and does a good comparison to other techniques at the end

- **Cons:**
  - Doesn’t spend so much time on motivations
  - Could be better structured (also matter of taste)
  - Too high level
The end

- Questions?
Backup slides
CherryPick example

- Naïve Solution: 4 switch IDs
CherryPick example

- CherryPick for VL2 topology: 1 switch ID

Only one shortest path from Core to Dst
PathDump interface

- **Host API**
  - `getFlows(linkID, timeRange)`
  - `getPaths(flowID, linkID, timeRange)`
  - `getCount(Flow, timeRange)`
  - `getDuration(Flow, timeRange)`
  - `getPoorTCPFlows(Threshold)`
  - `Alarm(flowID, Reason, Paths)`

- **Controller API**
  - `execute(List⟨HostID⟩,Query)`
  - `install(List⟨HostID⟩,Query,Period)`
  - `uninstall(List⟨HostID⟩,Query)`
Graphs for query evaluation

Figure 11: Average end-to-end response time and traffic amount of a flow size distribution query.

Figure 12: Average end-to-end response time and traffic amount of a top-10,000 flows query.