Detecting Routing Loops in the Data Plane

cesnet

CoNEXT 2020

Conference on emerging Networking EXperiments and Technologies

Jan Kučera
jan.kucera@cesnet.cz
CESNET, a.l.e.

In collaboration with:

Ran Ben Basat *(Harvard University)*,
Mário Kuka *(CESNET)*, **Gianni Antichi** *(Queen Mary University of London)*,
Minlan Yu *(Harvard University)* and **Michael Mitzenmacher** *(Harvard University)*

Routing loops

- **harm network** operation, **lead to losses**, which
  - increase the **tail latency** [1]
  - are **interpreted as a congestion** (TCP), cause **throughput reduction** [2]
- significantly **increase the overall traffic** [3]
- **affect other traffic** on shared links in terms of **delay and jitter** [1]

- **real-time detection is essential** for the network performance

[1] ***Detection and Analysis of Routing Loops in Packet Traces.*** *Urs Hengartner, Sue Moon, Richard Mortier, Christophe Diot. In* ***IMW 2002***.
[2] ***Packet Loss Impact on TCP Throughput in ESnet.*** *http://fasterdata.es.net/network-tuning/tcp-issues-explained/packet-loss/*
[3] ***Packet-Level Telemetry in Large Datacenter Networks.*** *Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, et al. In* ***SIGCOMM 2015.***

Routing **loops detection**

- **3 categories, classification of the past approaches:**
  (based on handling the information needed for detecting loops)
  1) keep **flow state at switches**
  2) **mirror information** at switches
  3) store **information on packets**

- **3 perspectives to evaluate them:**
  1) **switch overhead,**
  2) **network overhead,**
  3) **real-time detection**

Routing loops **detection approaches** (1)

1) On-switch state

- **aggregate flow information at switches**
- **periodically export them to a collector**
- **keeping state, e.g., up to 100K active flows**
- e.g., **FlowRadar** [1], **Hash IP Traceback** [2]
- **switch overhead: high**
- **network overhead: low**
- **real-time detection:** ✗

*[1] **FlowRadar: A Better NetFlow for Data Centers.** Yuliang Li, Rui Miao, Changhoon Kim, Minlan Yu. In **NSDI 2016.***
*[2] **Hash-based IP Traceback.** Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, et al. In **SIGCOMM 2001**.*

Routing loops **detection approaches** (2)

2) Header Mirroring

- duplicate the traffic headers
- sending it to an analyzer
- e.g., **NetSight** [1], **Everflow** [2], **Trajectory Sampling** [3]
- switch overhead: low
- network overhead: high
- real-time detection: ✗

[1] ***FlowRadar: A Better NetFlow for Data Centers.*** *Yuliang Li, Rui Miao, Changhoon Kim, Minlan Yu. In* ***NSDI 2016.***
[2] ***Packet-Level Telemetry in Large Datacenter Networks.*** *Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, et al. In* ***SIGCOMM 2015.***
[3] ***Trajectory Sampling for Direct Traffic Observation.*** *N. G. Duffield and M. Grossglauser. In* ***Transactions on Networking 2001****, Vol: 9, Issue: 3.*

Routing loops **detection approaches** (3)

3) Full Path Encoding on Packets

- **each switch records its ID in the incoming packet**
- **if its ID is already stored, a loop is detected**
- **per packet overhead cost grows linearly**
- e.g., **INT** [1], **PathDump** [2], **Tiny Program Packets** [3]

- **switch overhead: low**
- **network overhead: high**
- **real-time detection: ✓**

---

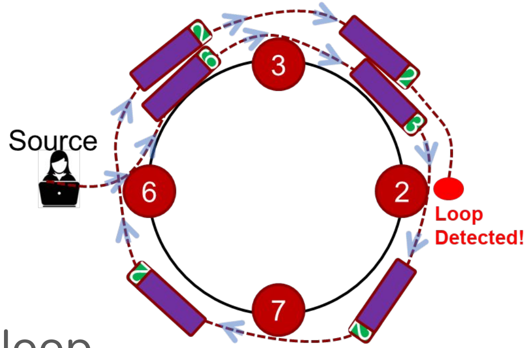
[1] ***In-band Network Telemetry (INT) Dataplane Specification.*** https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report.pdf
[2] ***Simplifying Datacenter Network Debugging with Pathdump.*** Praveen Tammana, Rachit Agarwal, and Myungjin Lee. In ***OSDI 2016.***
[3] ***Millions of Little Minions: Using Packets for Low Latency Network Programming and Visibility.*** Vimalkumar Jeyakumar, et al. In ***SIGCOMM 2014.***

Design space

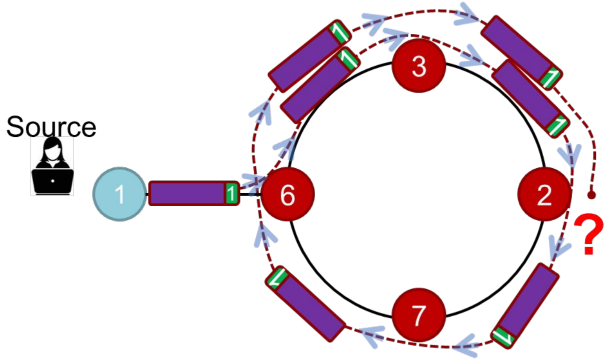
- All the **existing solutions**
  - are either **unable to detect loops in real time** or
  - have a **packet overhead** that is **linear in the number of hops**
- Can we design **an algorithm that detects routing loops**
  - in the **data plane,**
  - at **real time,**
  - while keeping **low switch and network overheads?**
- **INT** stores **all switches,** storing **Bloom Filter** saves the bandwidth
  - **reduces the overhead**, but introduces **false positives**
  - but still encodes IDs of **all the visited switches**, is it necessary?

Designing Unroller

- **No need to store all switches on the loop !**
- **Unroller**
  - stores **only some switch** on the loop
  - the **minimum switch ID** that it has seen
  - **reports the loop** when we see **repeated switch ID**
  - guaranteed detection **after two iterations** thought the loop



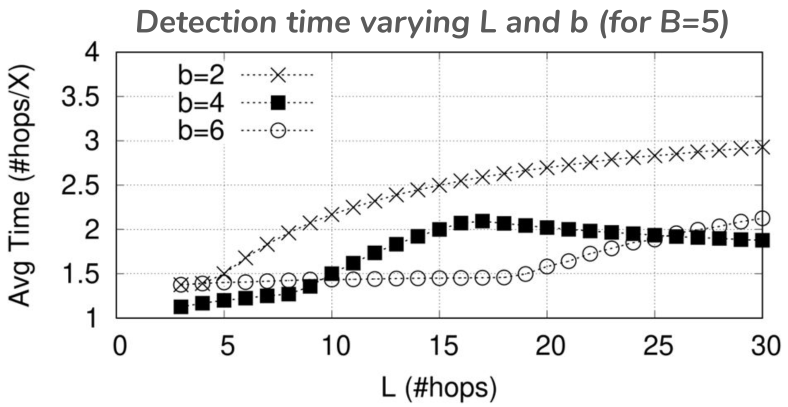
- **A path of switches before reaching the loop !**
- We occasionally **reset the stored ID**
  - **gradually increasing the resetting interval**
  - reset after each **phase** -- **b^i hops** for i=1,2,3,...
  - e.g., for b=2 phases consist of 2, 4, 8, 16, ... hops



Unroller | Detection time analysis

- **B** … the number of **hops before the loop**
- **L** … the number of **switches in the loop**
- **at least X=B+L hops** required for any algorithm

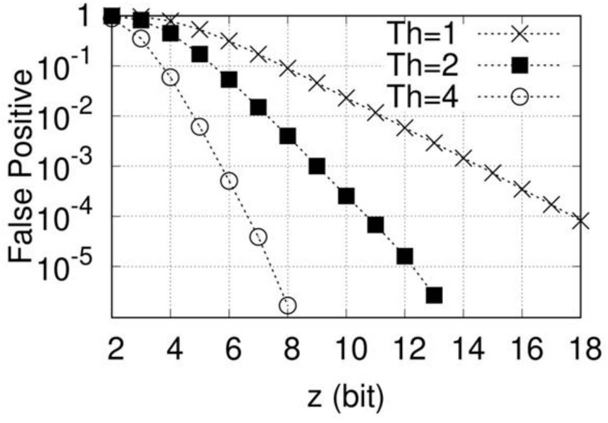
- **We showed that** (the proof presented in the paper)
  - after **no more than 4.67X hops** the packet reaches a switch that reports the loop
  - **lower bound ≈ 3.73X** (the minimal number of hops required by **any algorithm that stores a single ID**)
  - **not far from optimal** for deterministic algorithms



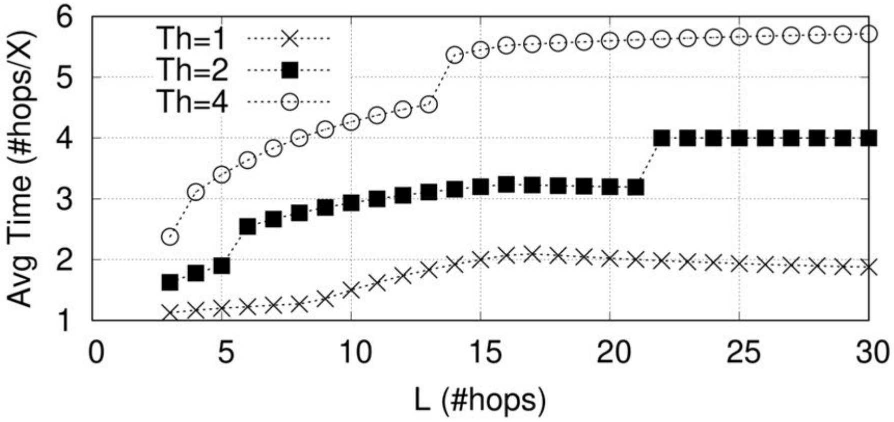
Unroller | Reducing per-packet overhead

- **hashing switch ID into z bits** and storing only hash instead
  - that introduces also **false positives** (FPs)
- **counting technique** to **reduce FPs**
  - small counter to **track the number of times the switch matches** the stored ID
  - once the **counter reaches the threshold** (Th), we **report the loop**

FPs for compressed IDs *(b=4, B=20, L=0)*



Detection time using counting technique *(b=4, B=5)*



Implementation

- Unroller implemented **using P4** and **compiled into BMv2**
  - **number of visited hops, minimum switch ID seen,**
  - value of **Th counter, encoded on packets**
  - other **b, z, Th values preset**
- **HW resources** quantified
  - compiled for **three FPGA-base NICs**
  - **two Xilinx FPGAs**, and **one Intel FPGA**

Lightweight Unroller implementation, requiring less than 8% of chip resources

Platform	LUTs	REGs	BRAM	Frequency
Virtex 7 (XCVH580T)	26 234 **(7.23%)**	29 944 **(4.13%)**	396 kb **(1.17%)**	224 MHz
Virtex US+ (XCVU7P)	26 221 **(7.23%)**	30 520 **(4.21%)**	684 kb **(2.02%)**	225 MHz
Stratix 10 (1SG280HU)	21 917 **(1.17%)**	45 907 **(1.22%)**	301 kb **(0.12%)**	189 MHz

- created **Python simulator** for evaluation **on real topologies**

Open sourced and available on GitHub: https://github.com/kucejan/unroller

Evaluation

- **Sensitivity analysis**
  how different parameters (b, B, z, ...) affect Unroller performance (presented above)
- **Comparing to state-of-the-art solutions**
  - Comparison of **false positives between Unroller and Bloom filter**
  - Comparison of **per-packet overhead on real topologies**

Comparison of Unroller and other real-time detections on real topologies

Topology	# of Nodes	Dia-meter	Bloom filter Overhead	Unroller	
				Avg Time	Overhead
Stanford	16	2	171b	1.74X	25b
BellSouth	51	7	189b	1.56X	25b
GEANT	40	8	608b	2.13X	27b
ATT-NA	25	5	608b	2.13X	27b
UsCarrier	158	35	2466b	2.47X	28b
FatTree4	20	4	414b	1.73X	28b

** over 3M runs so that there are no FPs*

6x-100x

Conclusion

- *Detecting Routing Loops in the Data Plane*
- **Unroller** = a lightweight **loop detection solution**
  - easily **deployable on programmable switches**
  - encodes **only a small subset of the switches** along the path
  - using a **minimal bit-overhead** on packets
  - **does not store state** on switches
  - identifies loops **in real time**
  - **without** a remote **analysis node**
  - detection in a **bounded number of hops**

Questions ?

jan.kucera@cesnet.cz

Unroller | Trading bandwidth for convergence

- **storing multiple identifiers on packets** (c·H in total)
- **H ∊ ℕ, the number of hash functions** (parallel runs of the algorithm)
  - multiple switches can have "minimum IDs" with respect to some hash function
- **c ∊ ℕ, the number of phase chunks** (partitioning each phase its chunks)
  - each of the c identifier tracks the minimum only on a 1/c fraction of the phase

Detection time for different c (number of chunks) and H (number of hashes) configurations *(b=4, B=5, L=20, z=32)*