Verifying Forwarding Plane Connectivity and Locating Link Failures using Static Rules in Software Defined Networks

[Extended Abstract]

Ulaş C. Kozat, Guanfeng Liang and Koray Kökten
DOCOMO Innovations, Inc., Palo Alto, CA 94304
Email: {kozat,gliang,kkokten}@docomoinnovations.com

ABSTRACT

We present efficient solutions that install static rules on the forwarding elements such that network controllers use these rules to verify topology connectivity and locate link failures. For a forwarding plane with $|E|$ links, we verify topology connectivity using $\leq 2|E|$ static rules and one control message. We guarantee locating at least one link failure using $\leq 6|E|$ static rules and $\Theta(\log(|E|))$ control messages. We can also detect multiple link failures in a probabilistic sense.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Operations—*Network monitoring*

Keywords

SDN, OpenFlow, Verification, Failure Detection

1. INTRODUCTION

In SDNs, presumably, controllers maintain a global view of the forwarding plane and data flows to take the correct actions against the network events. This might not always be possible due to control plane issues such as failures, overload, slow-path, or malicious behavior.

As a back up, network controllers can directly use the forwarding plane by allocating a fraction of hardware forwarding rules on each switch to check and verify forwarding plane properties at forwarding plane speeds. This idea in general might be infeasible or too costly for arbitrary properties. This paper presents that for at least two critical cases, i.e., verifying topology connectivity and locating link failures, it is feasible and efficient to use hardware rules. Our results are strong in the sense that: (i) As long as an arbitrary controller can reach an arbitary switch, it can verify topology connectivity and locate an arbitrary link failure. (ii) The number of forwarding rules to install and the number of control messages to inject are optimal or near-optimal. For a forwarding plane with $|E|$ links, we verify topology connectivity using $\leq 2|E|$ static rules and one control message. We guarantee locating an arbitrary link failure using $\leq 6|E|$ static rules and $\Theta(\log(|E|))$ control messages. (iii) All controllers use the same static forwarding rules.

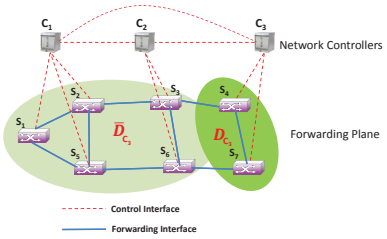

Figure 1: System Model

System Model and Assumptions: At any given time, each controller C_j has a control domain $\mathbf{D}_{C_j}$ that it can directly monitor and program. Each link is assumed to be bidirectional. We assume that an OpenFlow-like protocol is run between the controller and the switches in $\mathbf{D}$. Switches check only the health of their local interfaces. Thus, a controller does not receive any failure notifications or topology updates about the interfaces between the switches in its $\overline{\mathbf{D}}$. E.g., in Fig. 1, forwarding plane has seven switches (s_1 through s_7) and nine links between them. C_3 has active control connections with $\mathbf{D}_{C_3} = \{s_4, s_7\}$ whereas has no active control interfaces with $\overline{\mathbf{D}}_{C_3} = \{s_1, s_2, s_3, s_5, s_6\}$. A link failure between s_2 and s_3 is not reported to C_3 by s_4 or s_7. Controllers together compute and install a set of static rules onto the switches that they use later for topology verification and failure detection as described next.

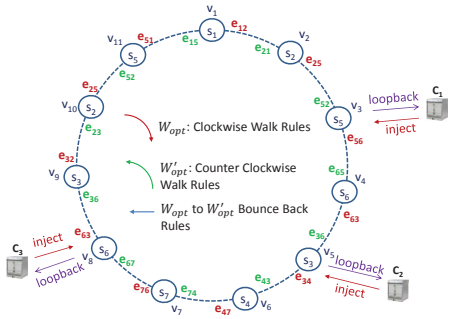

Figure 2: Bidirectional Logical Ring Topology

HotSDN'13, August 16, 2013, Hong Kong, China.
ACM 978-1-4503-2178-5/13/08.

2. SOLUTION

The first step of the solution is to compute a minimum length cycle $\mathbf{W_{opt}}$ with length $|W_{opt}|$ that visits every link in the forwarding plane at least once. If both directions of a link must be inspected, the forwarding plane is represented as a directed graph with $2|E|$ arcs. Otherwise it is represented as an undirected graph with $|E|$ edges. In directed graph case, for each link we have two arcs which guarantees that Euler cycles of length $2|E|$ exist (i.e., each direction of each link is visited exactly once) and they can be computed in $\Theta(|2E|)$ time steps. For general graphs, finding a minimum length cycle that visits every edge at least once is known as *Chinese Postman Problem* and it has a polynomial time solution for undirected graphs [1]. Since the optimum walk cannot be shorter than $|E|$ and longer than the walk for the directed graph case, $|E| \leq |W_{opt}| \leq 2|E|$. As the forwarding plane consists of bidirectional links, the reverse walk $\mathbf{W'_{opt}}$ also exists. $\mathbf{W_{opt}}$ and $\mathbf{W'_{opt}}$ together define a bidirectional logical ring topology G_R (see G_R in Fig. 2 for the topology in Fig. 1).

For topology verification, it suffices to install forwarding rules for $\mathbf{W_{opt}}$ (i.e., clockwise direction on G_R). Let e_{ij} denote the arc from s_i to s_j and l_{ij} denote the virtual arc from v_i to v_j. One rule at each v_j in the form *if* $\{label{=}{=}l_{ij} \wedge$ *packet ID*$=={=}y\}$ *then* $\{$*label* $:= l_{jk} \wedge$ *forward to switch* $v_k\}$ realizes $\mathbf{W_{opt}}$. For l_{ij}'s with 1-1 mapping to any port, instead of label, forwarding rule can use switch port ID. On G_R, if an arc appears multiple times in clockwise direction, they can share the same forwarding rule with proper label push/pop/swap actions. Thus, physically, we install $(|W_{opt}| - \kappa)$ static forwarding rules overall, where κ is the number of duplicate arcs on G_R. To verify topology, a controller attaches to any switch in its $\mathbf{D}$ and injects a control packet with the proper header fields. Controller also installs a *dynamic* rule at the point of injection to loopback the packet. The loopback rule must be specific to the controller to allow distinct controllers use the ring at the same time. If controller receives its packet back, topology is verified. Otherwise, there is at least one interface at fault or missing. For directed graphs, in number of control packets and forwarding rules, this becomes optimum. For undirected graphs, we have order optimality in number of static rules because $|E| \leq (|W_{opt}| - \kappa) \leq 2|E|$. Also, the latency of topology verification is $|W_{opt}| \cdot \tau$, where τ is per hop switching delay.

For locating a single link failure, we install static rules for $\mathbf{W_{opt}}$, $\mathbf{W'_{opt}}$ and bounce back rules from $\mathbf{W_{opt}}$ to $\mathbf{W'_{opt}}$. Static rules for $\mathbf{W'_{opt}}$ are computed the same as $\mathbf{W_{opt}}$ by labeling arcs in the counter clockwise direction on G_R. We need one bounce back rule for each arc on $\mathbf{W_{opt}}$ to revert the walk. Hence, total number of static rules are $6|E|$ and $(3 \cdot |W_{opt}| - 2\kappa)$ for directed and undirected forwarding topology graph, respectively. We also need a dynamic rule to loop back the packet to the controller at the injection point. Locating link failure then becomes a simple binary search over G_R. At each iteration, a control packet is sent to a different bounce back point as specified in the packet header such that half of the candidate failure locations on G_R are eliminated from further consideration. Overall, $\log_2(|W_{opt}|)$ messages are injected sequentially. Thus, the latency of locating link failure becomes less than or equal to $|W_{opt}| \cdot \log_2(|W_{opt}|) \cdot \tau$.

Performance Evaluations: We use topologies available in Internet Topology Zoo to evaluate $|W_{opt}|/|E|$ as the measure of performance. Fig. 3 plots $|W_{opt}|/|E|$ only for undirected graphs as $|W_{opt}|/|E|$ is always two for directed graph case. For around 11% of the topologies (e.g., for star and tree topologies), each link must be visited in both directions matching this upper bound. Less than 6% of the topologies has an Euler cycle and no links has to be visited multiple times. To put things into context, suppose the network uses fully connected 48-port switches. Performing both topology verification and failure localization would cost less than 3×96 static rules per switch. For a switch with 10K hardware forwarding rules, this corresponds to $\leq 3\%$ overhead.

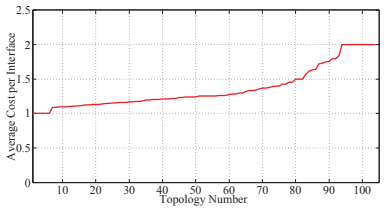

Figure 3: Performance in terms of $|W_{opt}|/|E|$

3. DISCUSSION AND FUTURE WORK

Maybe the closest work to ours is the concept of using preconfigured optical paths for fault diagnosis (i.e., monitoring trails or m-trails) in all optical networks [2, 6]. In our work: (i) we do not know assume a priori knowledge on available monitoring points (e.g., any switch is a candidate monitoring point); (ii) different controllers use the same static rules; and (iii) costs are different. In SDN domain, several works on network debugging, fault diagnosis and detection, policy verification, dynamic and static state analysis exist [3, 4, 5]. Installing static forwarding rules to be used for later forwarding plane diagnosis, optimizing the associated costs, and not requiring any specific attachment points are our unique contributions.

The logical ring can be used to locate more link failures. E.g., controller can inspect the ring from different attachment points in its $\mathbf{D}$ and bounce back rules that map $\mathbf{W'_{opt}}$ to $\mathbf{W_{opt}}$ can be installed to examine the ring in both directions. Though guaranteeing the detection of multiple link failures is not a solvable problem in general, probabilistic guarantees are possible. As another direction, we look into finding efficient solutions that trade-off control message overhead for reduced latency by sending packets in parallel.

4. REFERENCES

[1] J. Edmonds and E. L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5(1):88–124, Dec. 1973.

[2] S. Ahuja et al. SRLG failure localization in all-optical networks using monitoring cycles and paths. In *INFOCOM*, 2008.

[3] N. Handigol et al. Where is the debugger for my software-defined network? In *HotSDN*, 2012.

[4] P. Kazemian et al. Header space analysis: Static checking for networks. In *NSDI*, 2012.

[5] A. Khurshid et al. Veriflow: Verifying network-wide invariants in real time. In *NSDI*, 2013.

[6] J. Tapolcai et al. On monitoring and failure localization in mesh all-optical networks. In *Infocom*, 2009.