Correspondence Checking for Computer Networks

CS 263 Project, Spring 2012

Colin Scott*

1. BACKGROUND

Many network invariant checks [2, 3] are specified in terms of generally undesirable states such as loops or blackholes. In this project we discuss how to check finer-grained, more application-specific invariants without requiring human involvement. Our approach is to extract the network policies already reflected in the network controller's data structures, and compare this to the actual state of the physical network. We refer to this technique as correspondence checking. Correspondence checking builds on the virtual packet algebra pioneered in headerspace analysis [2].

1.1 Model

Computer networks can be represented as directed graphs:

$$G = (V, E)$$

Forwarding elements are represented as internal vertices:

$$V_{fwd} = \{v \in V | \; degree(v) > 1\}$$

End-hosts are represented as edge vertices:

$$V_{host} = \{v \in V | \; degree(v) = 1\}$$

End-hosts send and receive packets. A packet's source and destination, among other information, is encoded in the packet's header:

$$h \in \{0, 1\}^L = H$$

where L is the maximum number of bits in the header.

Upon receiving a packet, forwarding elements apply a transformation function[1]:

$$T : (H \times E) \to (H \times E_\emptyset)$$

Note that forwarding elements may rewrite fields of a packet header before passing it along. Forwarding elements may also drop packets, in which case $T(.) = (., \{\})$.

We use 'Ψ' to denote the collection of all transfer functions present in the network at a particular point in time.

In this model, network traversal is simply a composition of transformation functions. For example, if a header h enters the network through edge e, its state after k hops will be:

$$\Phi^k(h, e) = \Psi(\Psi(\ldots \Psi(h, e) \ldots))$$

Only end-hosts can insert new packets into the network[2]. The access links of end-hosts are therefore the only entry and exit points for packets:

$$E_{access} = \{(v_1, v_2) \in E | \; v_1 \in V_{host} \vee v_2 \in V_{host}\}$$

The externally visible behavior of the network is expressed as a relation between packets originating from end-hosts and the packets' final locations:

$$\Omega : (H \times E_{access}) \to (H \times (E \cup \{\emptyset, LOOP\}))$$
$$\Omega(h, e) = \Phi^\infty(h, e)$$

We use the special value $LOOP$ to distinguish a packet dropped by a network device from a packet entering an infinite loop (both of which never leave the network).

1.2 Software-defined Networks & Network Virtualization

Software-defined Networking (SDN) is a paradigm for controlling network behavior. SDN control software maintains a data-structure representing the global state of the network. We refer to this data-structure as the 'view':

$$G^{view} = (V^{view}, E^{view})$$

Control programs running on top of the SDN controllers configure the network by defining Ψ^{view}. The role of the platform is then to map Ψ^{view} to routing entries in the underlying forwarding elements of the physical network. The physical network is also a graph:

$$G^{physical} = (V^{physical}, E^{physical})$$

Note that the mapping between G^{view} and $G^{physical}$ is not necessarily one-to-one; the SDN control software may virtualize the view such that a single member of V^{view}_{fwd} maps to multiple members of $V^{physical}_{fwd}$. A common pattern is to map a single forwarding element $v \in V^{view}_{fwd}$ to an entire physical network $V^{physical}_{fwd}$. Figure 1 depicts an example. In the example, $V^{view}_{fwd} = \{Switch\}$, and $Switch$ maps onto:

$$\{vswitch_1, vswitch_2, vswitch_3, vswitch_4,$$
$$TOR_1 TOR_2, Gateway\} = V^{physical}_{fwd}$$

As we will see later on, it is important to note that there is still a one-to-one correspondence between V^{view}_{host} and $V^{physical}_{host}$, as well as between E^{view}_{access} and $E^{physical}_{access}$.

2. PROBLEM STATEMENT

It should always be the case that $\Omega^{view} \sim \Omega^{physical}$. Formally:

$$\forall_h \forall_{e \in E^{view}_{access}} \Omega^{view}(h, e) = \Omega^{physical}(encap(h), \hat{e})$$

where $encap$ is a function from packets in G^{view} to packets in $G^{physical}$, and $e \sim \hat{e} \in E^{physical}_{access}$.

*In collaboration with Andi Wundsam and Scott Shenker

[1] We assume unicast forwarding for the purposes of this proposal

[2] We ignore control and diagnostic traffic for the purposes of this proposal

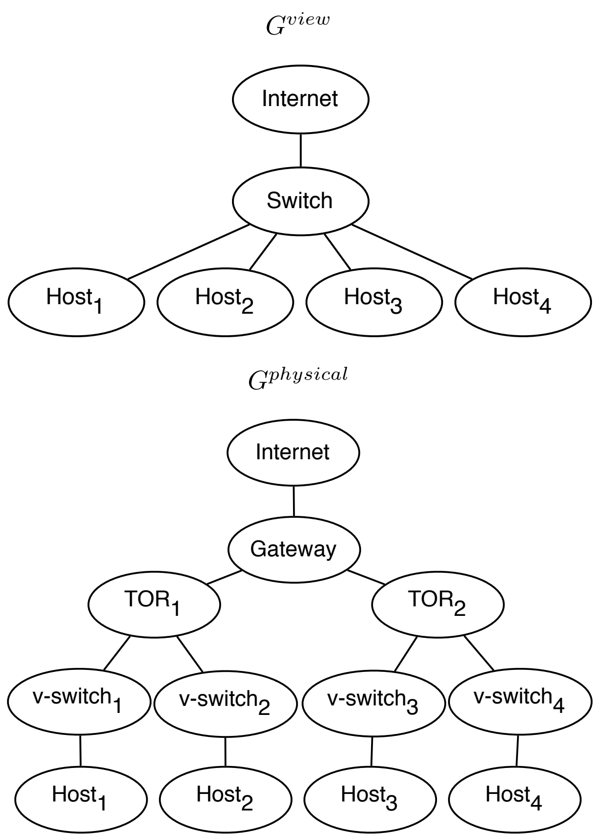

Figure 1: Example correspondence between G^{view} and $G^{physical}$. In the view a single switch connects four hosts and the rest of the Internet (modeled as a single vertex). This single switch is mapped onto four v-switches, two top-of-rack switches and one gateway router in the physical network.

In practice however, there are often miscorrespondence between the two. Some of these miscorrespondence are innocuous; networks are distributed systems, and there are inevitable delays between updates in G^{view} and the corresponding changes in $G^{physical}$. Other miscorrespondence are due to critical bugs in SDN control software. In general, controller software is a highly complex piece of software; it must deal with hardware-faults, communication delays, and complex mapping functions between G^{view} and $G^{physical}$. Moreover, production networks may contain $10's$ of thousands of forwarding devices. In such an environment, the control program must be replicated across multiple servers to handle the load and ensure fault-tolerance. The SDN control software must ensure that G^{view} remains consistent between all replicas despite server failures, communication delays, and message re-orderings.

3. ALGORITHM

Correspondence checking verifies correspondence between Ω^{view} and $\Omega^{physical}$. It works as follows:

i. Take a distributed snapshot of $G^{physical}$ [1]

ii. Translate the routing tables of $V^{physical}_{fwd}$ into transformation functions $\Psi^{physical}$

iii. Feed a symbolic packet h_s through each access link $e \in E_{access}$

iv. Track the progress of h_s through the network, thereby computing Ω^{view}. If a packet enters a loop before exiting the network, we mark the value as $LOOP$. Otherwise, the leaves of the propagation graph define the final outcomes of the input packets injected at that access link.

v. Do the same for G^{view}

vi. Because network policies are defined by configuring the logical view, any mismatch between Ω^{view} and $\Omega^{physical}$ represents an instance of a correctness violation. Note that mismatches are unambiguous; that is, two slightly different correctness violation from different runs of the system are always distinguishable. We therefore seek to identify counterexamples:

$$\{(h,e)|\ h \in H, e \in E^{view}_{access} \sim E^{physical}_{access}, \\ \Omega^{view}(h,e) \neq \Omega^{physical}(encap(h),e)\}$$

4. LIMITATIONS

The price of correspondence checking's generality is that it represents a somewhat weak notion of correctness. Correspondence checking only captures external behavior and loops; it does not capture internal behavior such as load-balancing over links. It also assumes that the policies as expressed by the configuration of the logical view are correct. Finally, correspondence checking can not verify time-dependent policies such as "No link should be congested more than 1% of the time".

[1] K. M. Chandy. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems*, 3:63–75, 1985.
[2] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking For Networks. In *NSDI*, 2012.
[3] H. Mai et al. Debugging the Data Plane with Anteater. In *SIGCOMM*, 2011.