Abstractions for Network Update

Background

Configuration changes can cause instability.

Intermediate configurations exhibit errors.

Outages, security vulnerabilities, etc.

Existing solutions are limited to a specific protocol and/or properties.

Example Application	Policy Change	Desired Property	Practical Implications
Stateless firewall	Changing access control list	No security holes	Admitting malicious traffic
Planned maintenance [1, 2, 3]	Shut down a node/link	No loops/blackholes	Packet/bandwidth loss
Traffic engineering [1, 3]	Changing a link weight	No loops/blackholes	Packet/bandwidth loss
VM migration [4]	Move server to new location	No loops/blackholes	Packet/bandwidth loss
IGP migration [5]	Adding route summarization	No loops/blackholes	Packet/bandwidth loss
Traffic monitoring	Changing traffic partitions	Consistent counts	Inaccurate measurements
Server load balancing [6, 7]	Changing load distribution	Connection affinity	Broken connections
NAT or stateful firewall	Adding/replacing equipment	Connection affinity	Outages, broken connections

Table 1: Example changes to network configuration, and the desired update properties.

Example

Three filter switches F1, F2, F3.

Load shifts, divide traffic between F1 and F2.

Switches configured one by one.

It is possible to allow untrustworthy traffic through if one does not have a verifiably correct transition plan.

Finding a correct transition plan requires in depth reasoning about intermediate steps.

Configuration I

	Type	Action
I	U, G	Forward F_1
	S	Forward F_2
	F	Forward F_3
F_1	SSH	Monitor
	$*$	Allow
F_2	$*$	Allow
F_3	$*$	Allow

→

Configuration II

	Type	Action
I	U	Forward F_1
	G	Forward F_2
	S, F	Forward F_3
F_1	SSH	Monitor
	$*$	Allow
F_2	SSH	Monitor
	$*$	Allow
F_3	$*$	Allow

Figure 1: Access control example.

Software Defined Networks

SDNs give programmers control over routing, access control and load balancing.

Allow for useful abstractions to help with the network update problem.

On their own they are insufficient due to a wide range of possible intermediate behaviors.

Per Packet and Per Flow Consistency

Abstractions that allow the programmer to change the entire network configuration.

Per Packet

Each packet in flight is processed by one consistent global configuration.

Per Flow

Guarantees packets in the same flow are handled by the same configuration.

Per Packet Abstraction

Stronger than “atomic updates”.

Without this the programmer has to reason about every possible trace between two configurations.

Each trace comes from only one configuration.

Packet equivalence relation, ~.

Two traces are equivalent if all packets are equivalent according to ~.

Per Packet Mechanisms

One touch updates

No packet can follow a path through the network that reaches an updated or non updated part of the rule space more than once.

Unobservable updates

An update does not change the set of traces on the network.

Two Phase update

Install on internal ports but only enable for the packets with the correct version number.

Property Invariance

Turning a trace property checker for static network configurations to one for the invariance of dynamic configuration trace properties.

Trace property: a path a packet is allowed to follow.

No loops: AF (port = DROP | port = WORLD)

Egress: H = 1 -> AF (port = WORLD)

Waypoint: H-1 -> AF (switch = s4)

Per Flow Consistency

More complex switches required.

Flow is a sequence of packets with similar header fields and not separated by n seconds. Corresponds to TCP connections.

Rules with timeouts: Soft timeouts for old rules

Wildcard Cloning: Exploits the *clone* feature of Openflow to roll out an update.

End-host feedback: Provide a list of active sockets to the controller, hosts force traffic to the correct endpoint until the new configuration can take effect.

Implementation

Python Implementation on top of Openflow.

per_packet_update() and per_flow_update()

Application	Toplogy	Update	2PC		Subset		
			Ops	*Max Overhead*	*Ops*	*Ops %*	*Max Overhead*
Routing	Fat Tree	Hosts	239830	92%	119003	50%	20%
		Routes	266234	100%	123929	47%	10%
		Both	239830	92%	142379	59%	20%
	Waxman	Hosts	273514	88%	136230	49%	66%
		Routes	299300	90%	116038	39%	9%
		Both	267434	91%	143503	54%	66%
	Small World	Hosts	320758	80%	158792	50%	30%
		Routes	326884	85%	134734	41%	23%
		Both	314670	90%	180121	57%	41%
Multicast	Fat Tree	Hosts	1043	100%	885	85%	100%
		Routes	1170	100%	634	54%	57%
		Both	1043	100%	949	91%	100%
	Waxman	Hosts	1037	100%	813	78%	100%
		Routes	1132	85%	421	37%	50%
		Both	1005	100%	821	82%	100%
	Small World	Hosts	1133	100%	1133	100%	100%
		Routes	1114	90%	537	48%	66%
		Both	1008	100%	1008	100%	100%

Experimental results comparing two-phase update (2PC) with our subset optimization (Subset). We add or remove hosts and change routes to trigger configuration updates. The *Ops* column measures the number of OpenFlow install operations used in each situation. The Subset portion of the table also has an additional column (Ops %) that tabulates (Subset Ops / 2PC Ops). *Overhead* measures the extra rules concurrently installed on a switch by our update mechanisms. We pessimistically present the maximum of the overheads for all switches in the network – there may be many switches in the network that never suffer that maximum overhead.

Questions?

All figures come from the paper.

Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. 2012. Abstractions for network update. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (SIGCOMM '12). ACM, New York, NY, USA, 323-334. DOI: https://doi.org/10.1145/2342356.2342427