Fast Control Plane Analysis Using an Abstract Representation

Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, Ratul Mahajan

UNIVERSITY OF WISCONSIN–MADISON

Microsoft Research

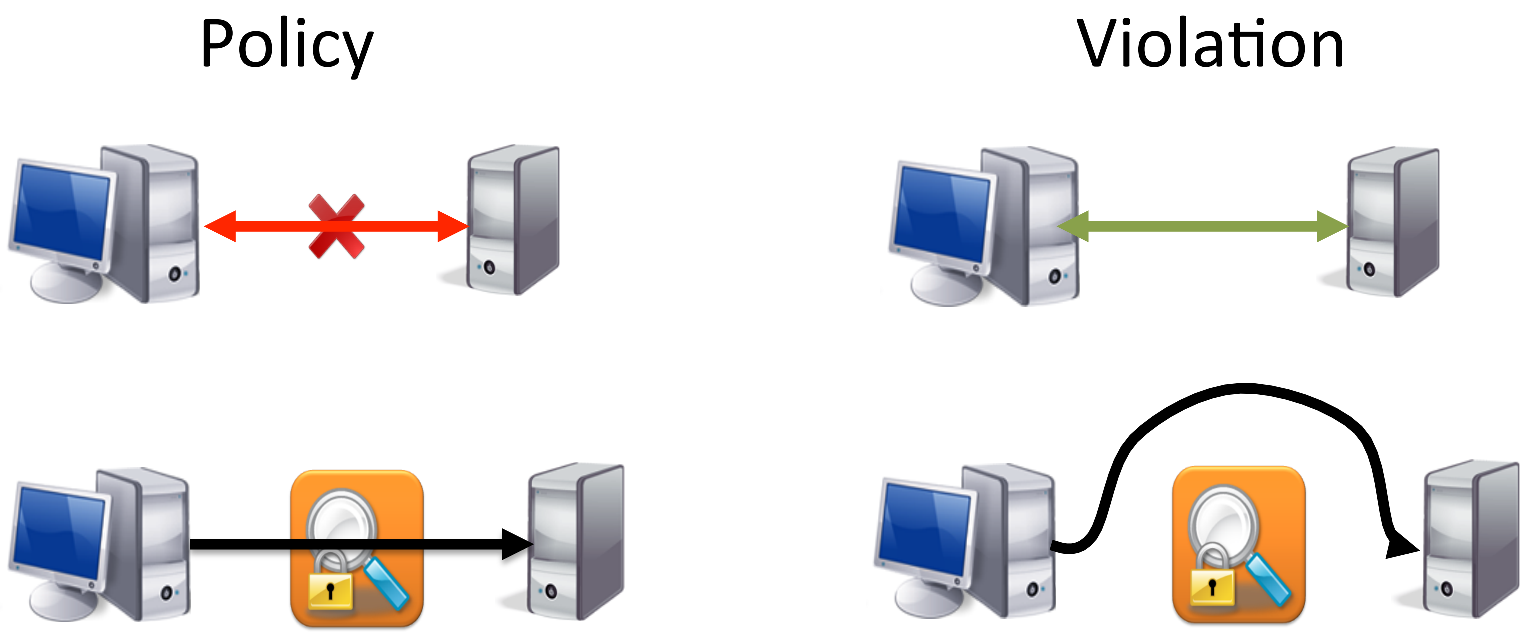
Configuration errors are common

```
route-map tag-filter deny 10
match tag 777
!
router ospf 10
 network 1.1.1.1 0.0.0.255 area 0
 distribute list route-map tag-filter in
 redistribute rip 10
!
router bgp 100
 no synchronization
 bgp router-id 2.2.2.2
 distance 190 3.3.3.3 0.0.0.0
!
router rip
 networ
 redist
!
```

- Multiple routing protocols
- Routing process priorities
- Route exchange
- Traffic Selectivity
  - Route Filters
  - ACLs

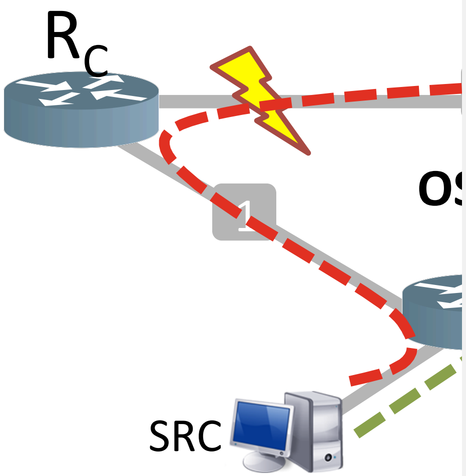
Human errors are unavoidable

Errors lead to policy violations



Network verification is important

Some violations or



DatacenterDynamics
The Business of Data Centers.

Microsoft: misconfigured network device led to Azure outage

30 July 2012 | By Yevgeniy Sverdlik

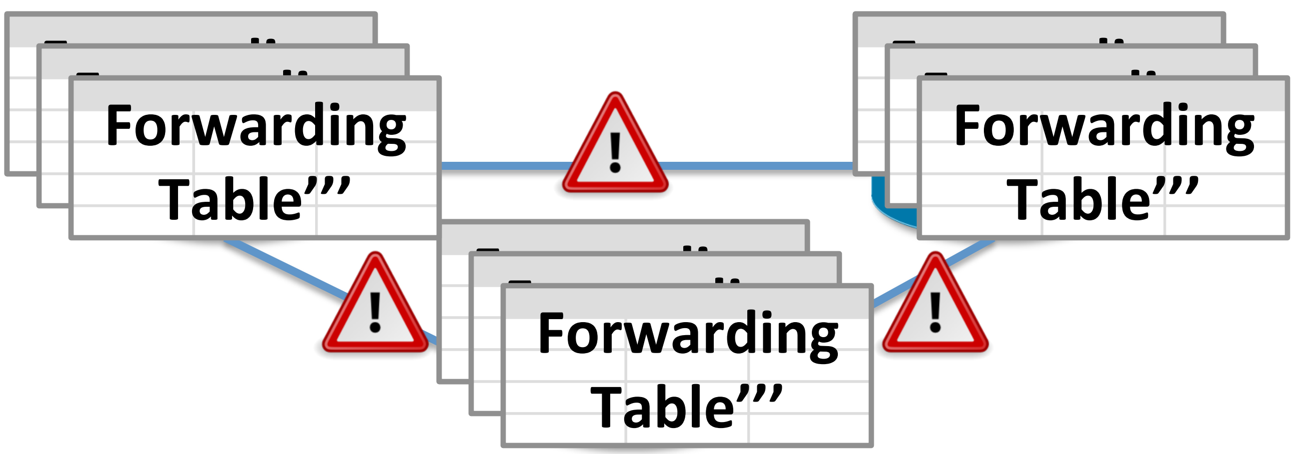
"The service interruption was triggered by a misconfigured network device that disrupted traffic to one cluster in our West Europe sub-region," Mike Neil, general manager for Windows Azure, wrote in a

further complicated network management and recovery."

Network verification under **arbitrary failures** is required

State-of-the-art verification with failures

- Analyze current data plane [HSA NSDI'13, VeriFlow NSDI'13]
  - Cannot verify policies across failures
- Simulate low level protocol messages [Batfish NSDI'15]
- Generate data planes for each failure case
  - Time consuming

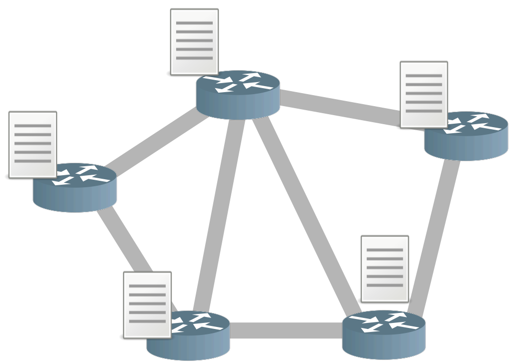


How do we speedup network verification under failures ?

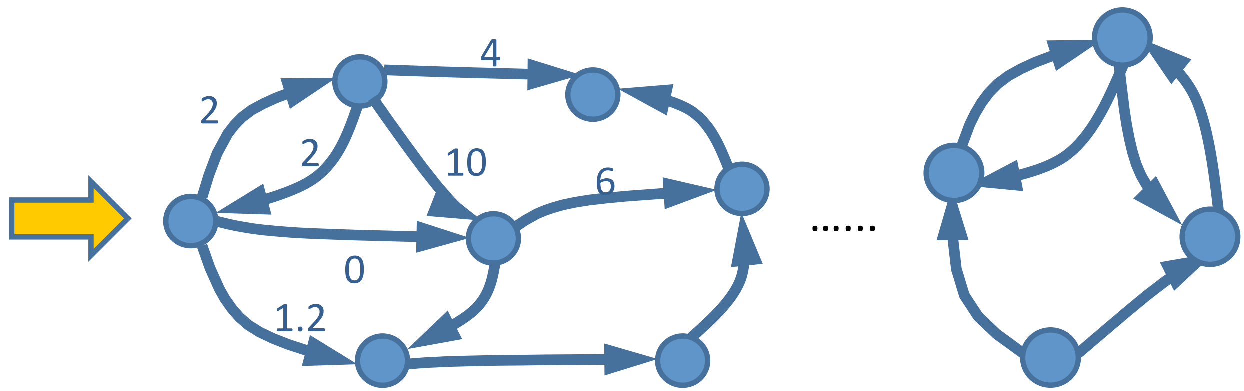
Network verification under failures

Graph Analysis

Network verification under failures using graph algorithms



Network configurations



Collection of weighted digraphs

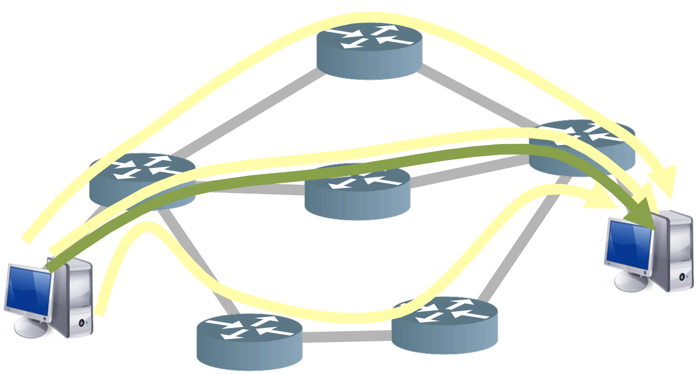
- Graphs encode the network's forwarding behavior under **all possible** failure scenarios
- Verification reduces to checking simple graph-level properties → **polynomial time**
- Collection of digraphs → **ARC**: Abstract Representation for Control planes

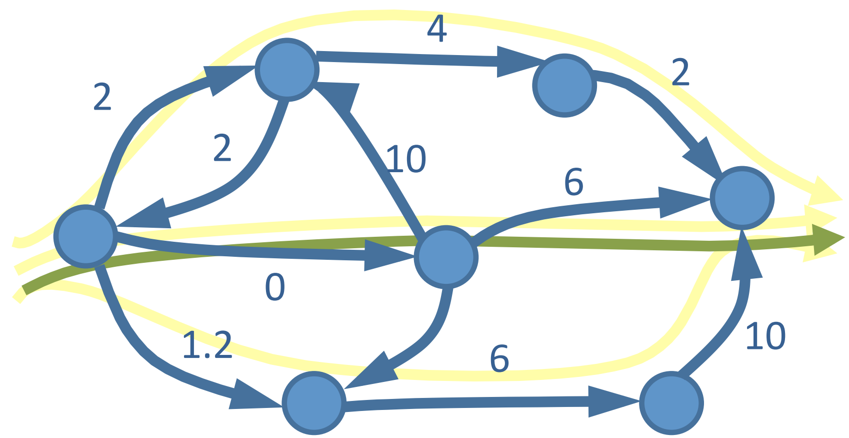
Outline

- Motivation
- Requirements & Challenges for ARC creation
- Our approach for constructing ARCs
- Network verification using ARCs
- Evaluation

Requirement: Encoding forwarding behaviors under all failures

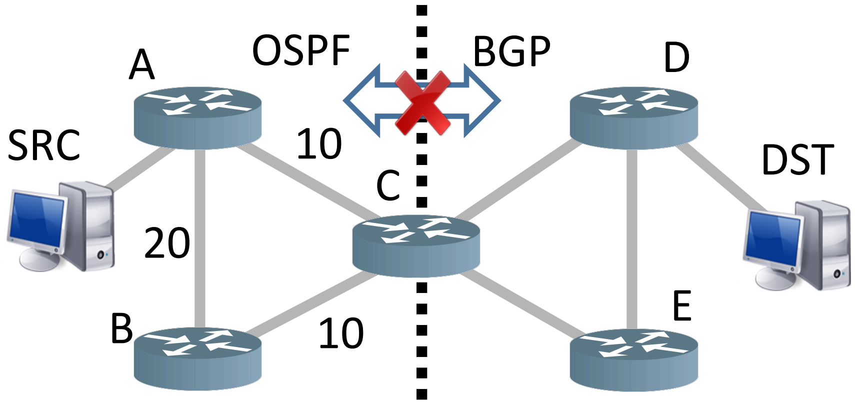
- Graph contains all possible paths in the actual network

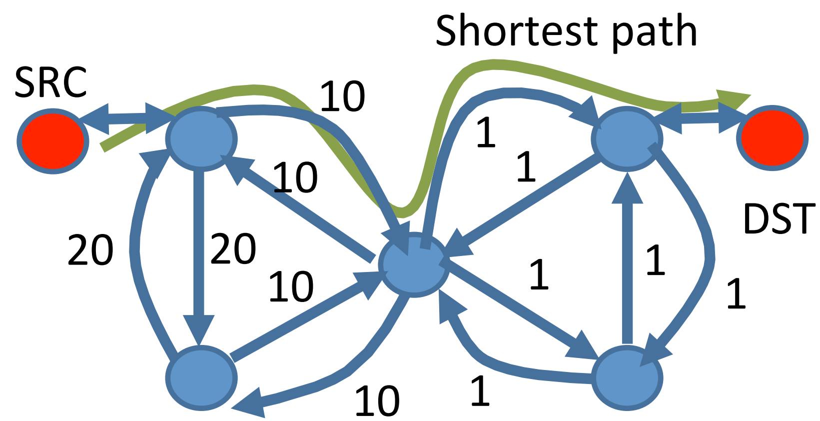




- Actual path under particular failure scenario is obtainable through graph traversal

ARC construction: First steps





Opportunities

- Network topology is essentially a graph

Challenges

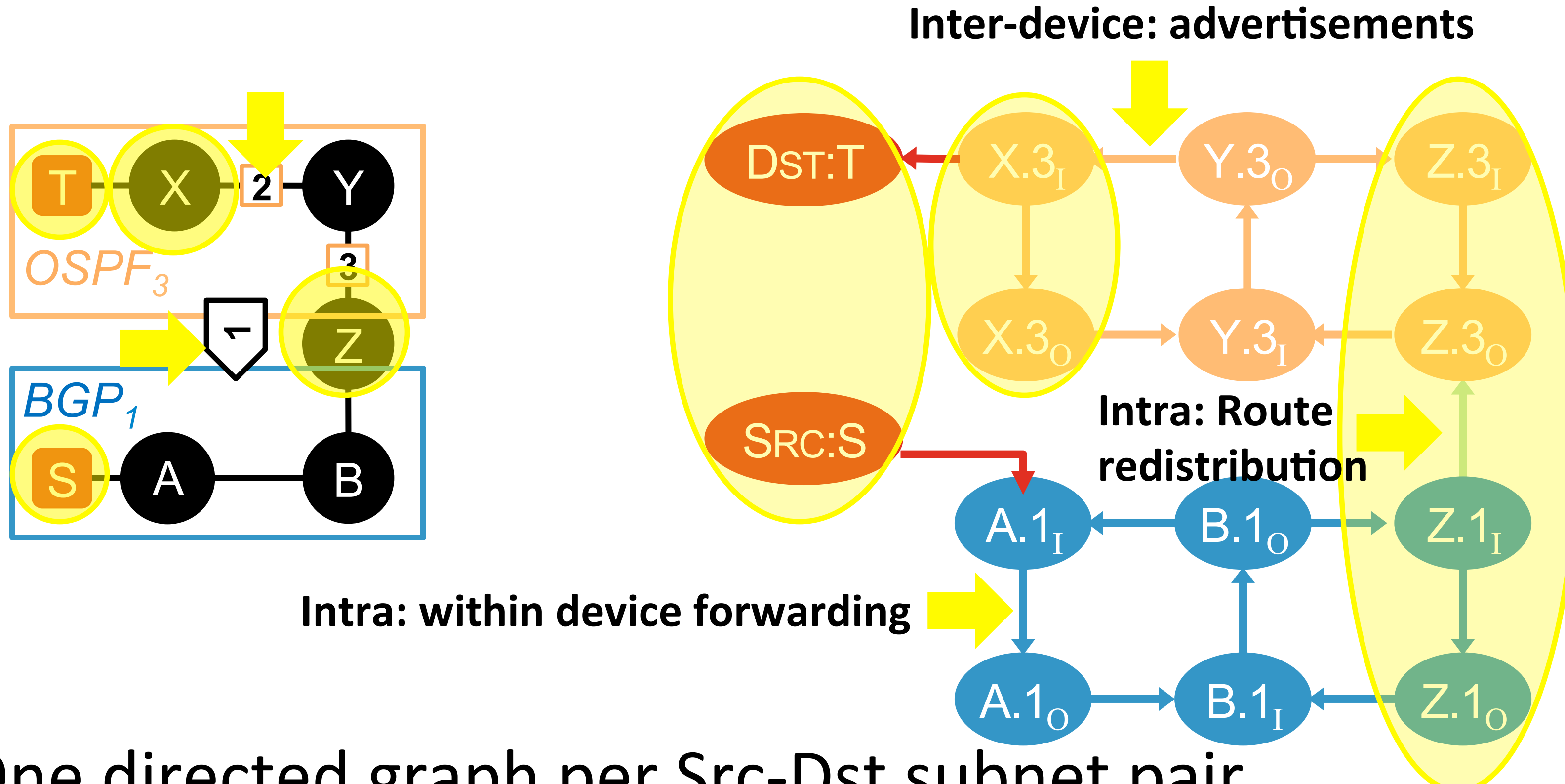
- Route redistribution
- Routing cost varies / protocol

Need sophisticated approaches to determine graph structure and edge weights

– BGP: Min AS hops

- ...

ARC Construction: Graph Structure



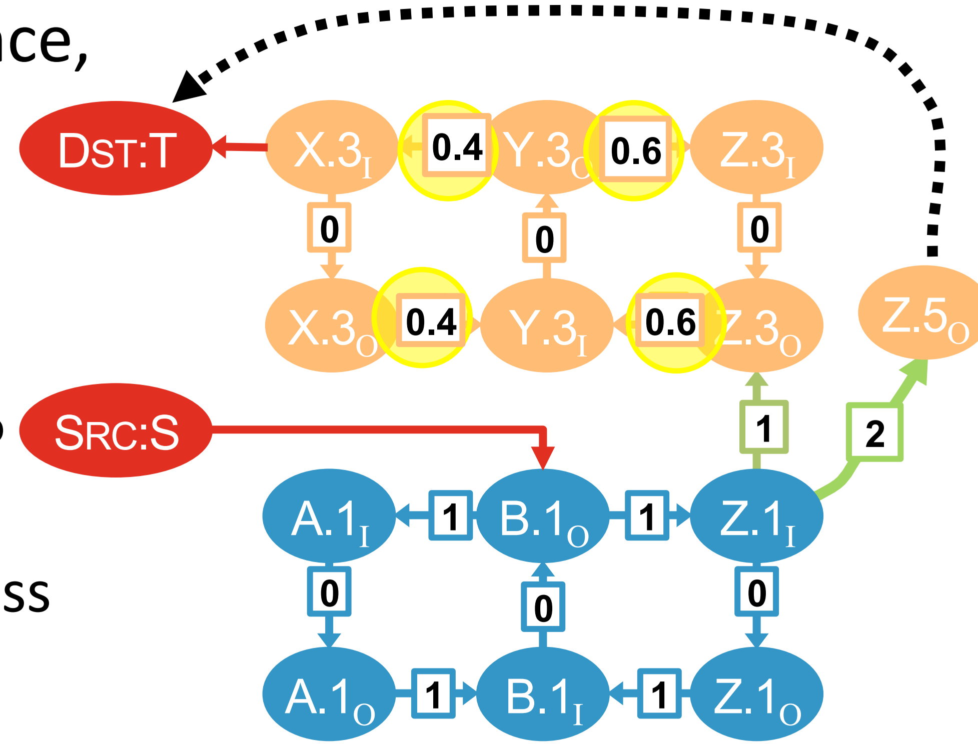
- One directed graph per Src-Dst subnet pair
- **Vertices**: hosts, routing processes
- **Edges**: flow of data enabled by exchange of routing information

ARC construction: Edge weights

- For single routing instance, use:
  - OSFP link weights
  - BGP hop counts

- Multiple processes: AD? Redistribution?
  - **Normalize** weights across instances

- Novel algorithm for scaling weights



Shortest path in ARC == actual path

Policy verification using ARCs

Is a policy violated in the network?

::

Does the graph satisfy some property ?

What graph algorithms to use ?

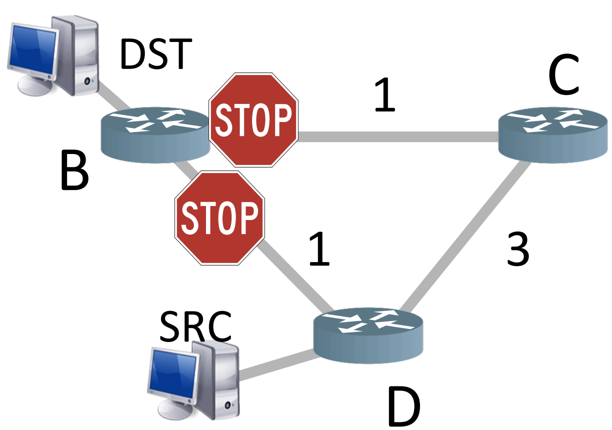
Verify always blocked policy

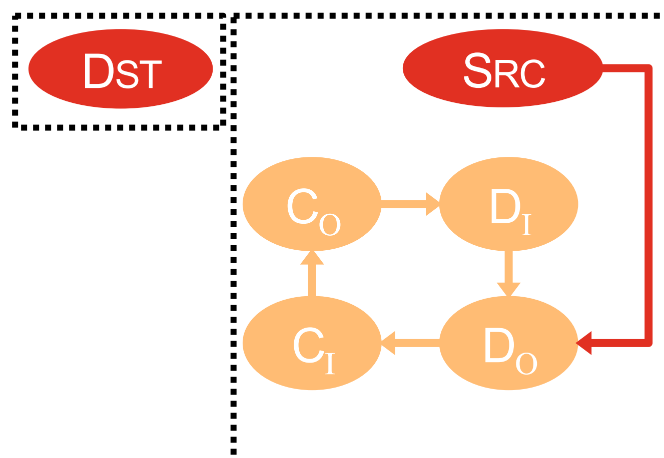
Is communication between SRC and DST not allowed under any failure scenario?

::

Does there exist a path from SRC to DST in the corresponding ARC?

Connected components





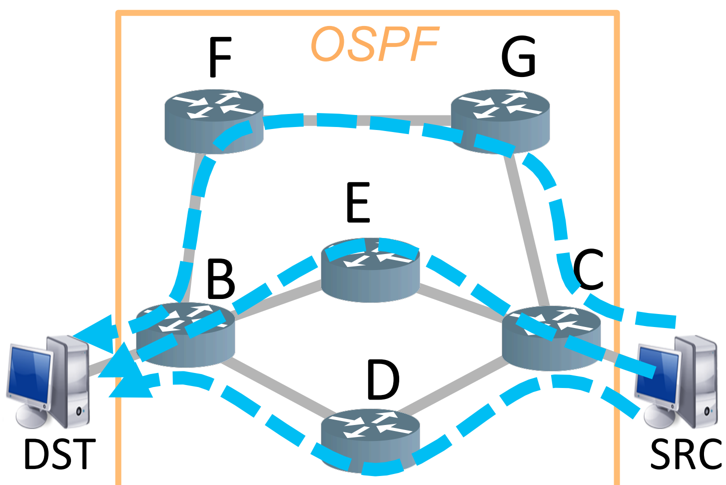
Verify ‘k-’reachability policy

Is DST always reachable from SRC with ‘< k’ failures ?

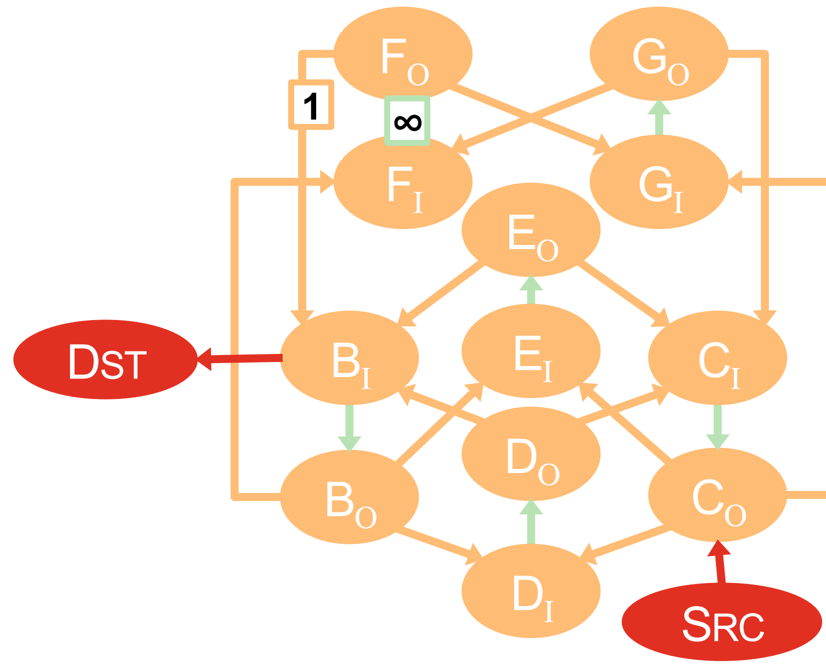
::

Are there ’k’ edge-disjoint paths from SRC to DST ?

Max-flow algorithm on ARC



3 edge-disjoint paths



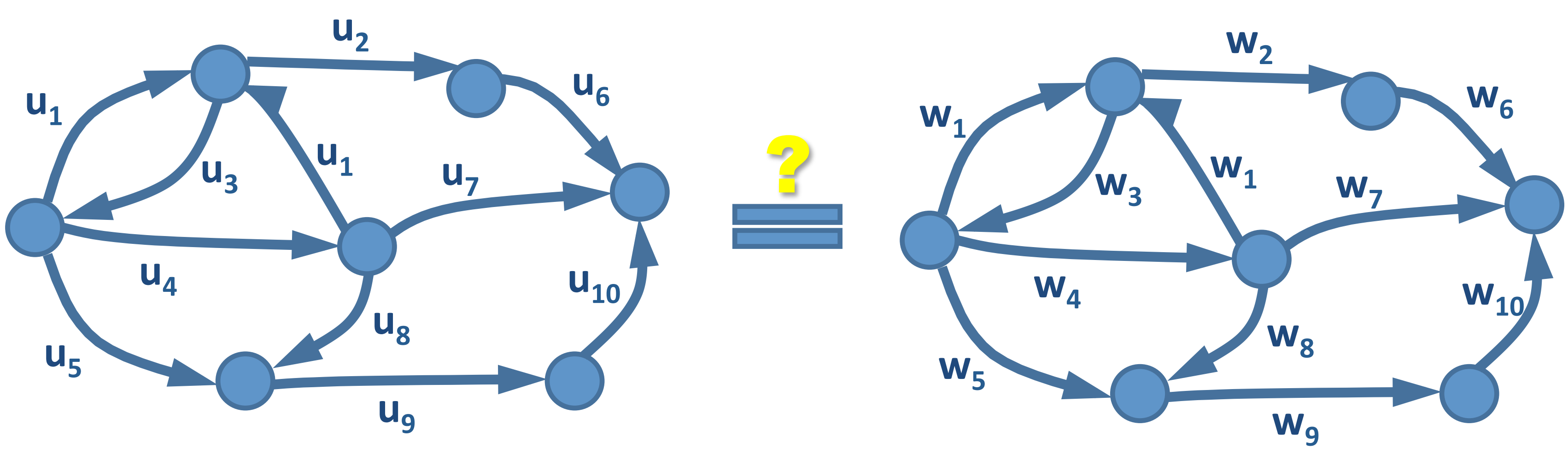
Max-flow = 3

Verify path equivalency

Is a traffic class forwarded in the same manner, before and after a configuration change?

::

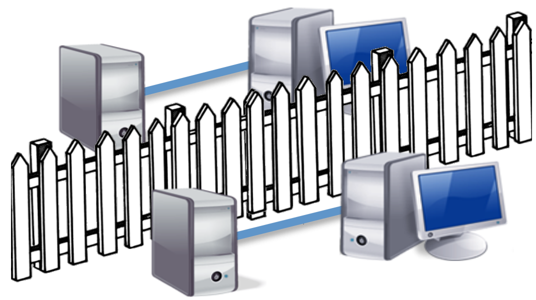
Are ARCs the same ?



- Re-scaling algorithms can result in different weights
- Reduce weights to canonical form and compare

Additional properties we can verify

- Always isolated: Traffic of different tenants are always isolated



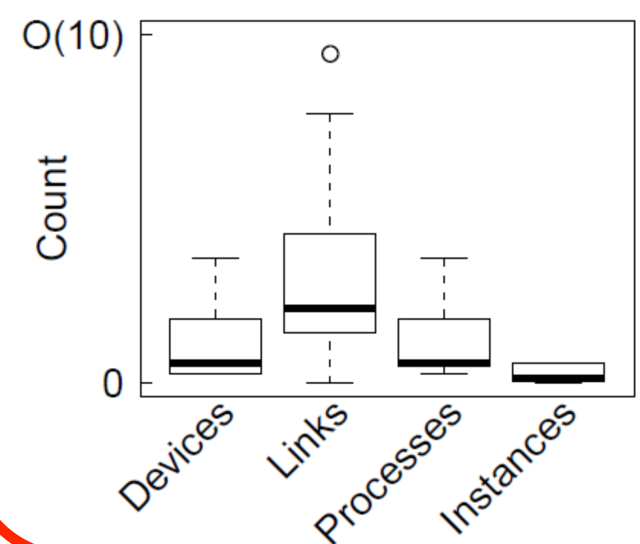
- Always traverse waypoints: Traffic between hosts always traverse waypoints

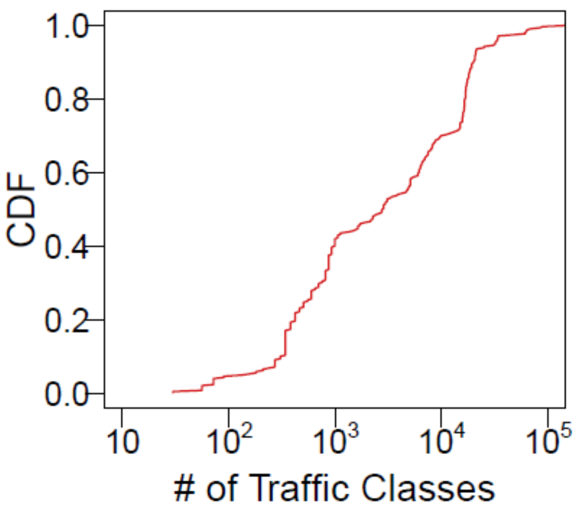
Evaluation

- ARC construction performance
- ARC verification performance
- ARC fidelity

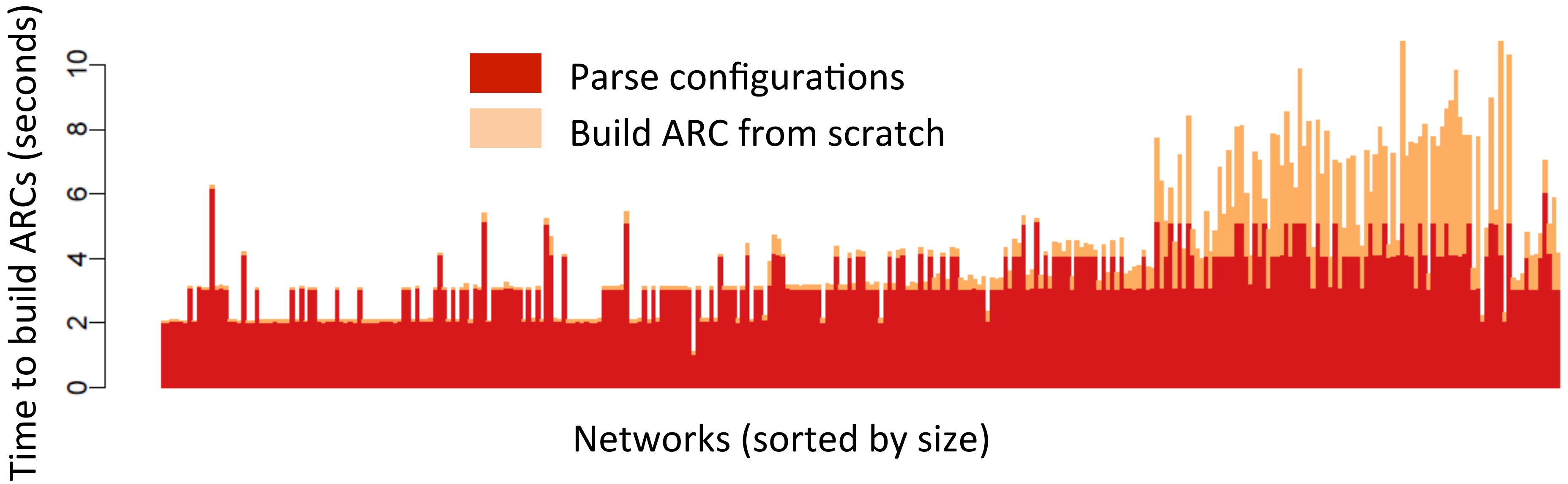
Network configurations

- Configurations from 314 data center networks operated by a large online service provider



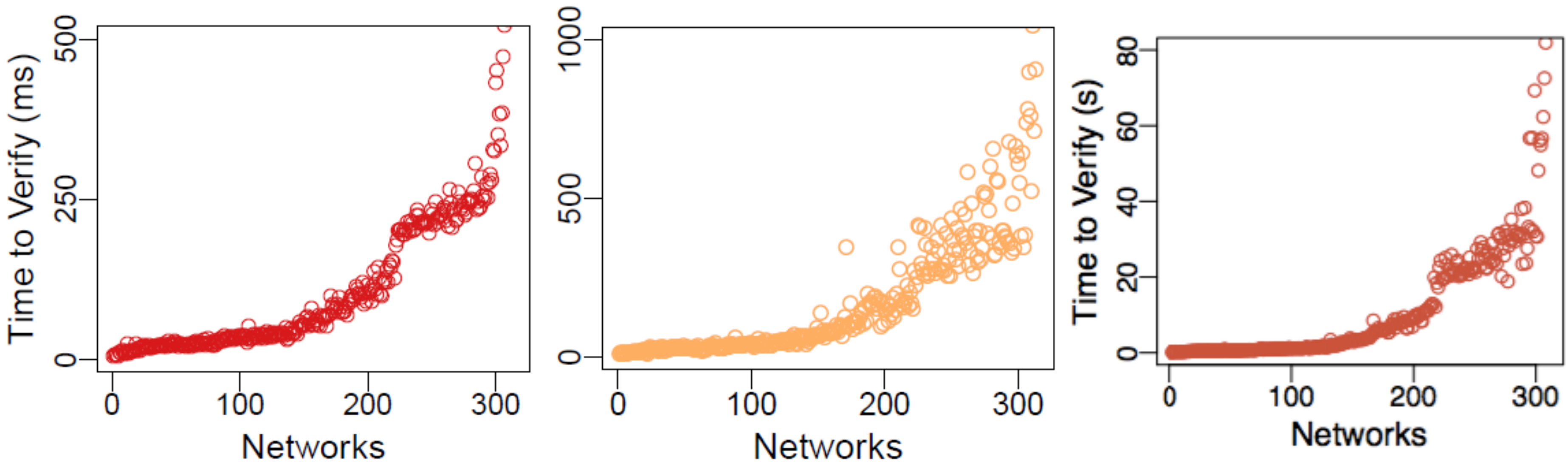


Time to generate ARC



Fast (<10 seconds) even for large networks

Time to verify ARC



Always blocked *(connected components)*

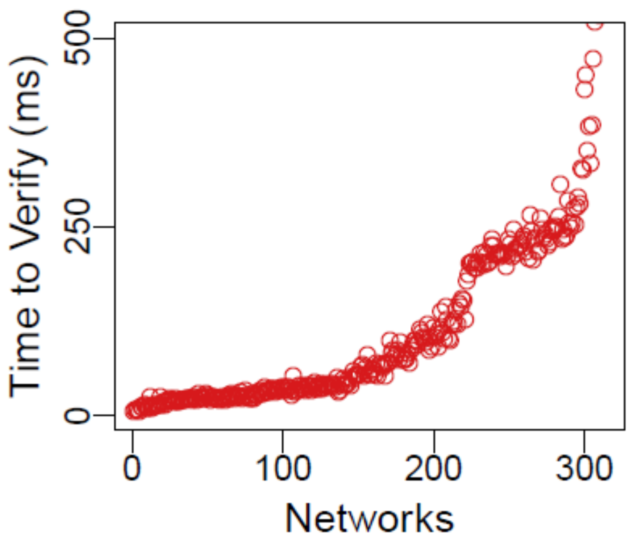
Always reachable with < k failures *(max flow)*

Equivalent paths *(convert to canonical weights and compare)*

< 500 ms **< 1 sec** **Up to 100 s**

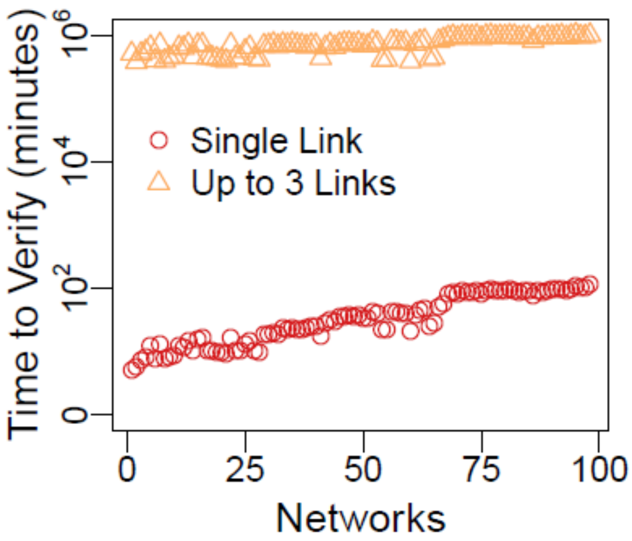
- Verification per traffic class is parallelizable

Comparison with Batfish



Always blocked using ARC

< 500 ms



Always blocked using Batfish

Up to 694 days!

3 - 5 orders of magnitude speedup

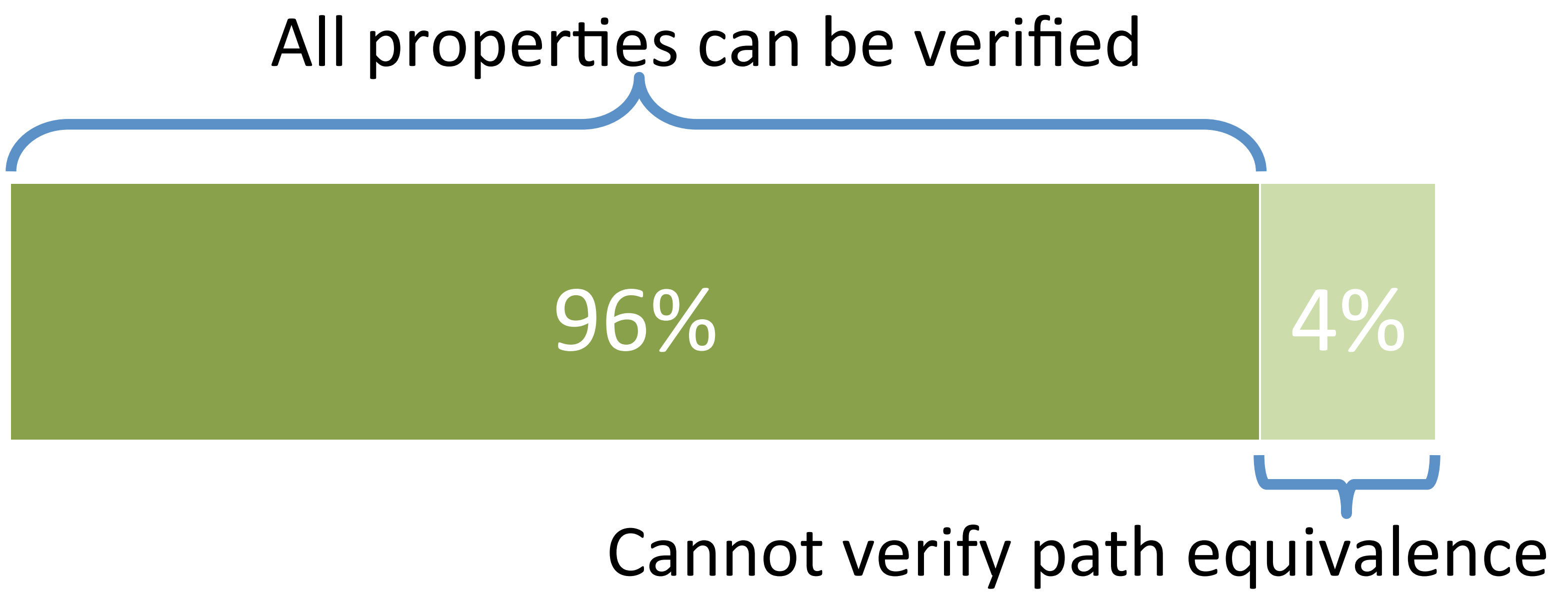
ARC fidelity

- For any given failure scenario, is ARC shortest path == actual network path?
- Formally prove ARC fidelity for networks with:
  - Routing protocols : OSPF, RIP, BGP
  - Route redistribution is acyclic
  - Route selection preference follow a global order

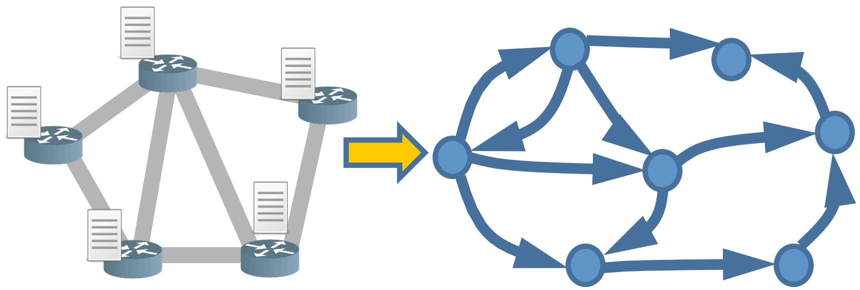
96% of networks satisfy these properties

ARC fidelity

- For remaining networks
  - We can still generate the graph structure
  - Cannot generate edge weights
  - Verify “always blocked”, “k-reachability”



Summary



- Network verification under failures can be formulated as graph analysis
- Presented an abstract representation, **ARC**
- Can construct high fidelity ARCs for **96%** of networks
- **$O(10^3)$-$O(10^5)$** speedup in verification

https://bitbucket.org/uw-madison-networking-research/arc