Automatically Correcting Networks with *NEAt*

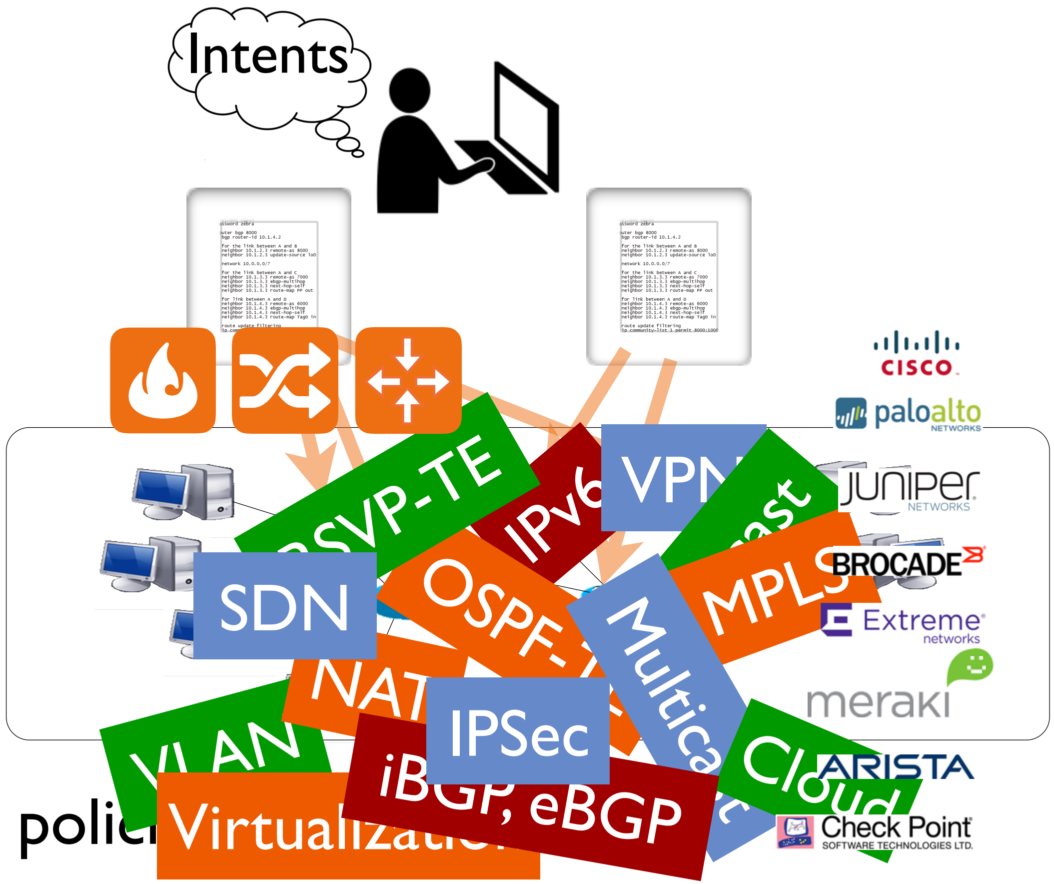
Wenxuan Zhou, Jason Croft, Bingzhe Liu,
Elaine Ang, Matthew Caesar

University of Illinois at Urbana-Champaign

A simple idea about complexity…

Networks are so complex it's hard to make sure they're doing the right thing.

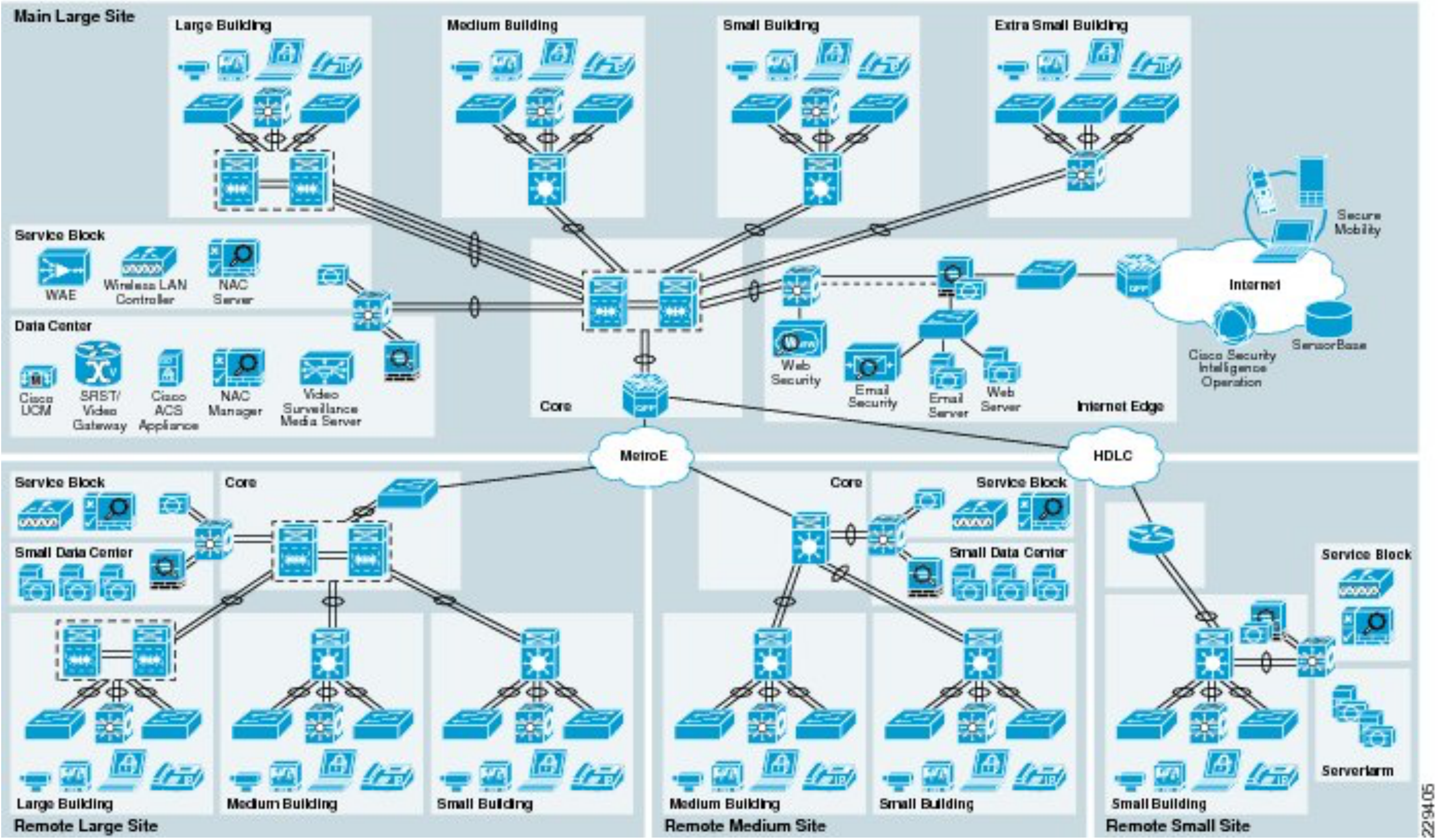
What is a network supposed to do?



They instill polic

- no untrusted traffic entering a secure zone
- the preference of one path over another
- loop & black hole avoidance
- …

A Typical Enterprise Network



Source: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Medium_Enterprise_Design_Profile/MEDP/chap5.html

Network errors are common

Lots of problems arise today

89% of operators are <u>not</u> certain their configuration changes are safe. *[Kinetic NSDI'15]*

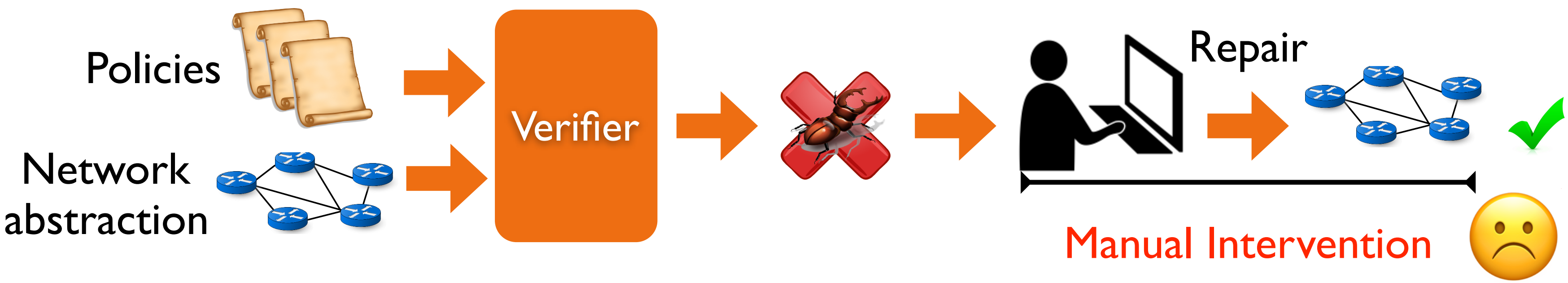
Networks are so complex it's hard to make sure they're doing the right thing.

Let's automate.

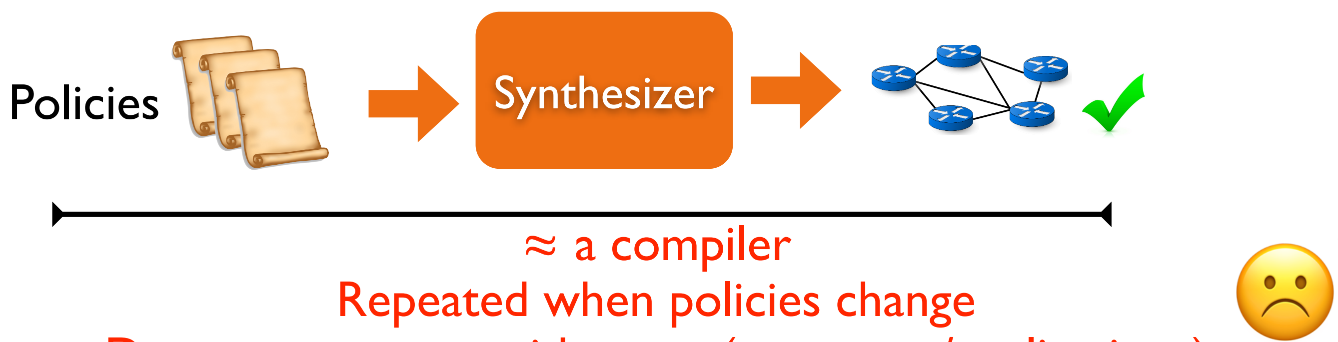
What to automate?

Many have tried.

Automatically identifying errors in networks (Verification)

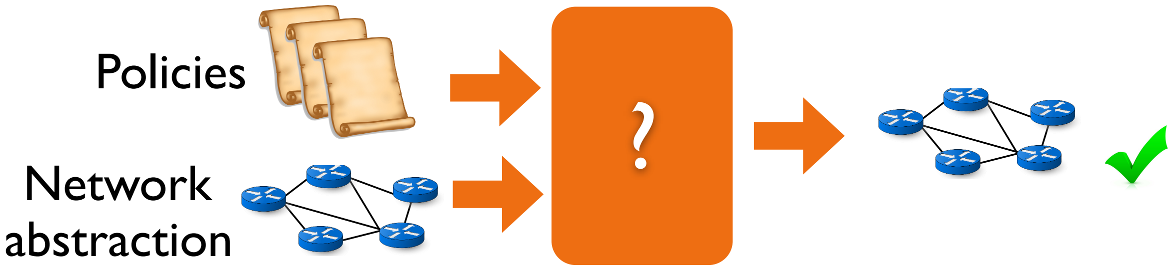


Automatically synthesize correct networks

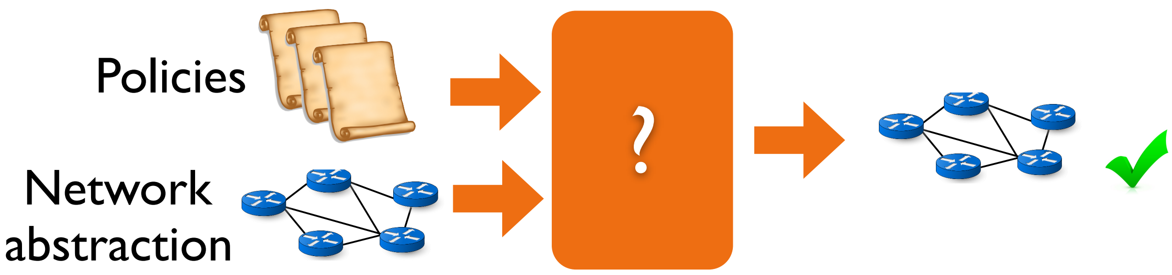


≈ a compiler
Repeated when policies change
Do not cooperate with users (operators/applications)

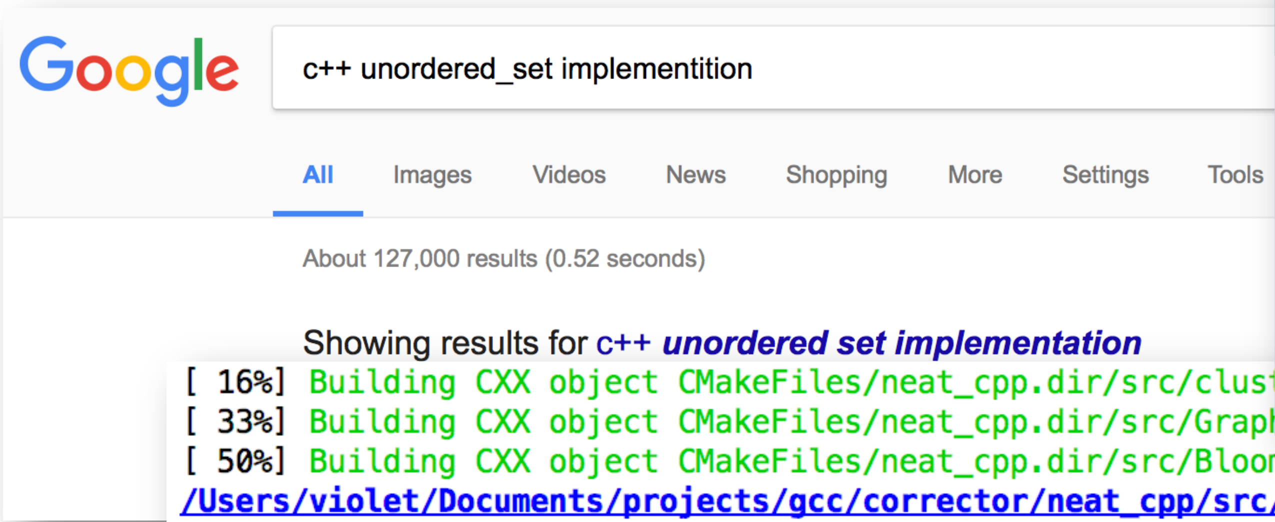
What if we can automatically correct networks on the fly?

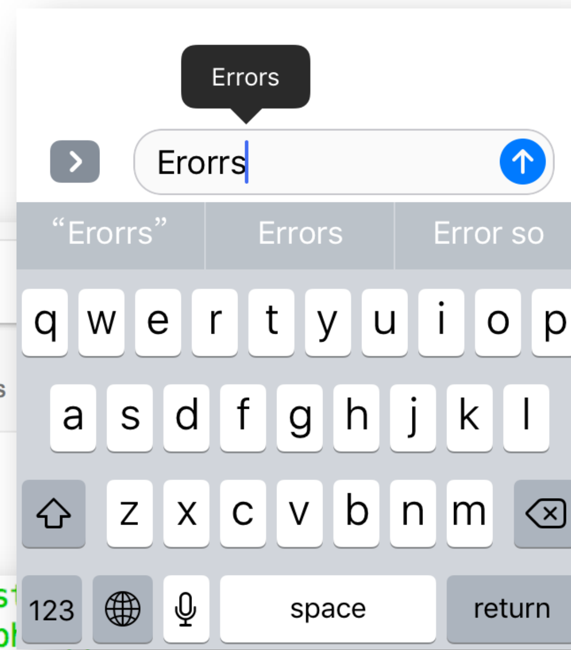


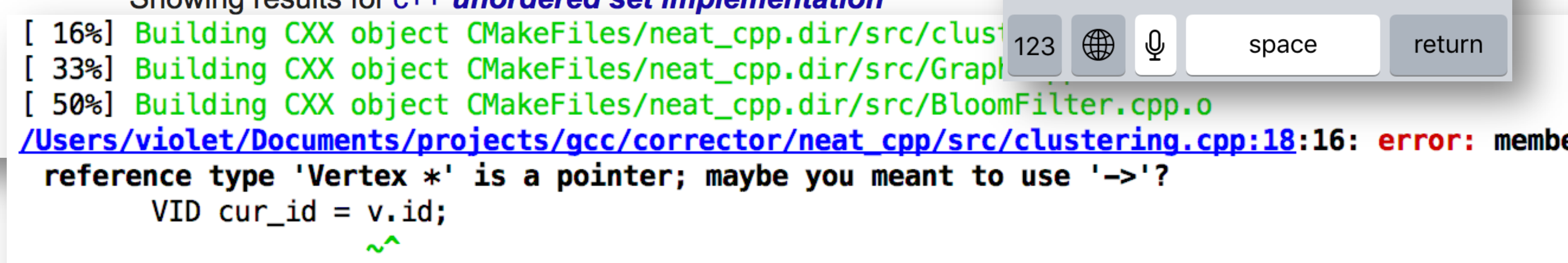
What if we can automatically correct networks on the fly?



Auto-correct is not a new idea.

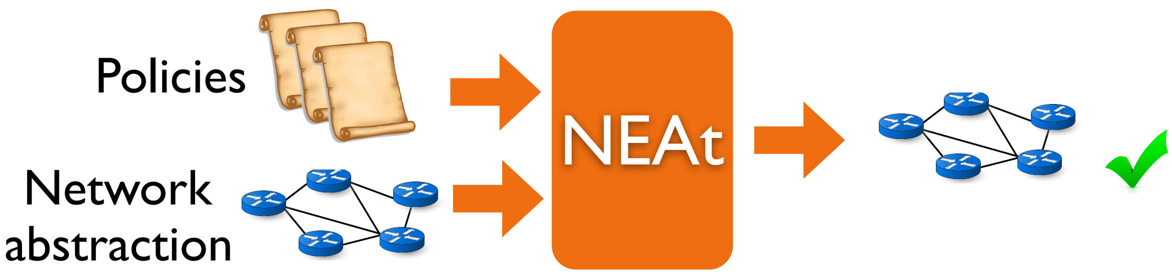




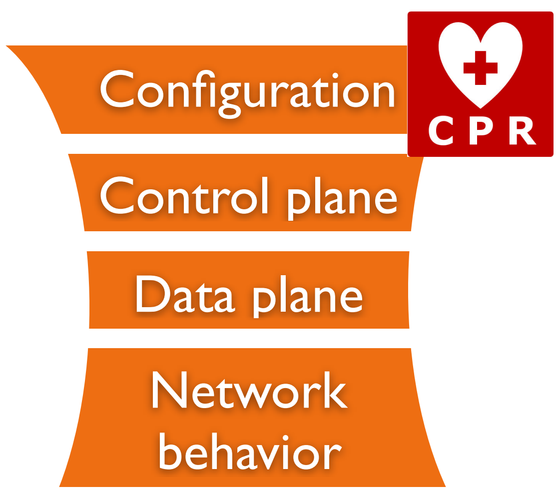


Network Error Auto-Correct

What if we can automatically correct networks on the fly?



Network Abstraction



Operating on the data plane simplifies our work

- *Diagnose problems as close as possible to actual network behavior*
- *Data plane is a "narrower waist" than configuration*

Challenges

Goal: Improve upon a manual effort with transparency in both performance and architecture.

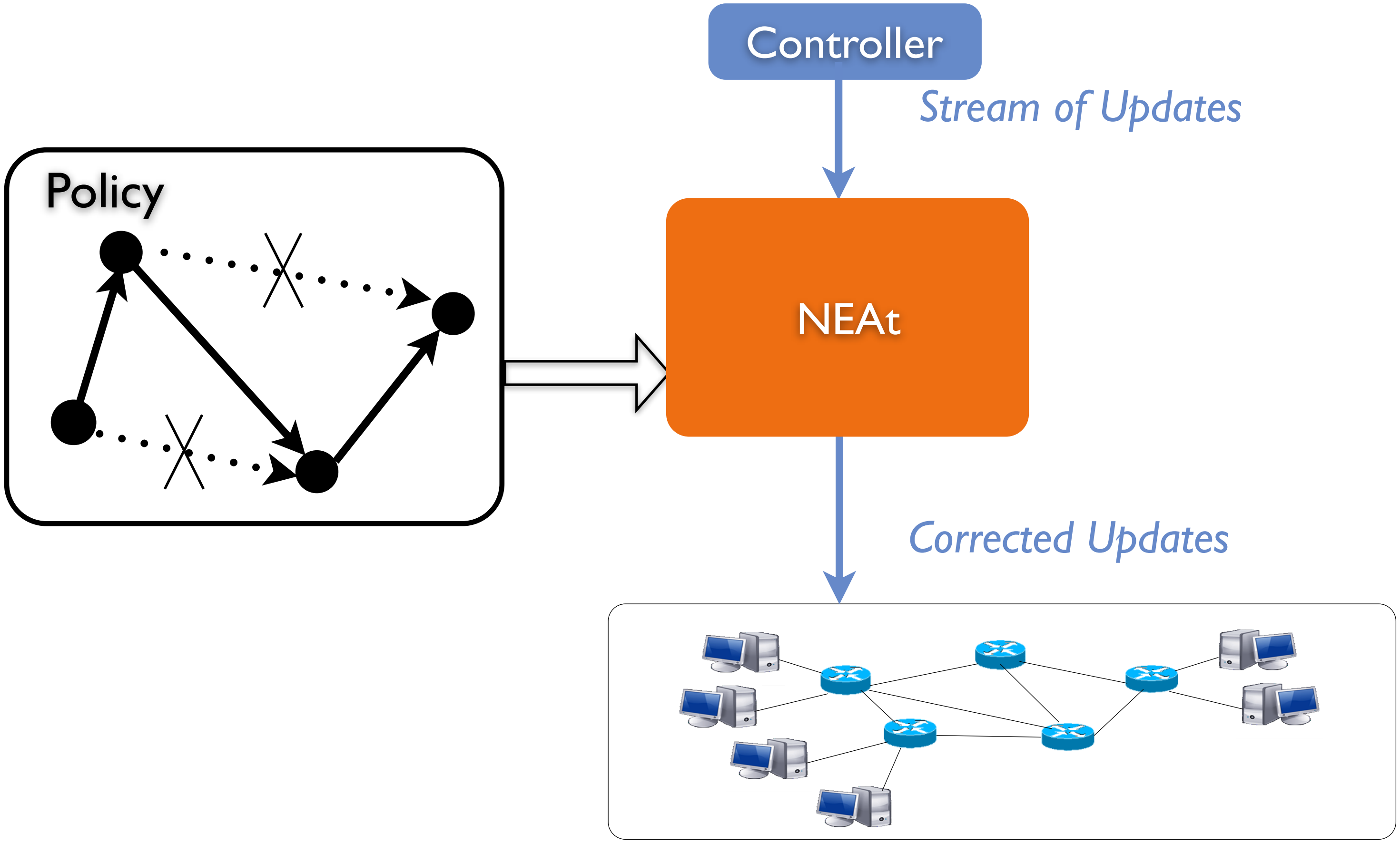
Challenge 1: Repair speed

- Based on real-time verification technique
- Derive fixes via linear optimization, with min. changes
- Topology limitation & graph compression

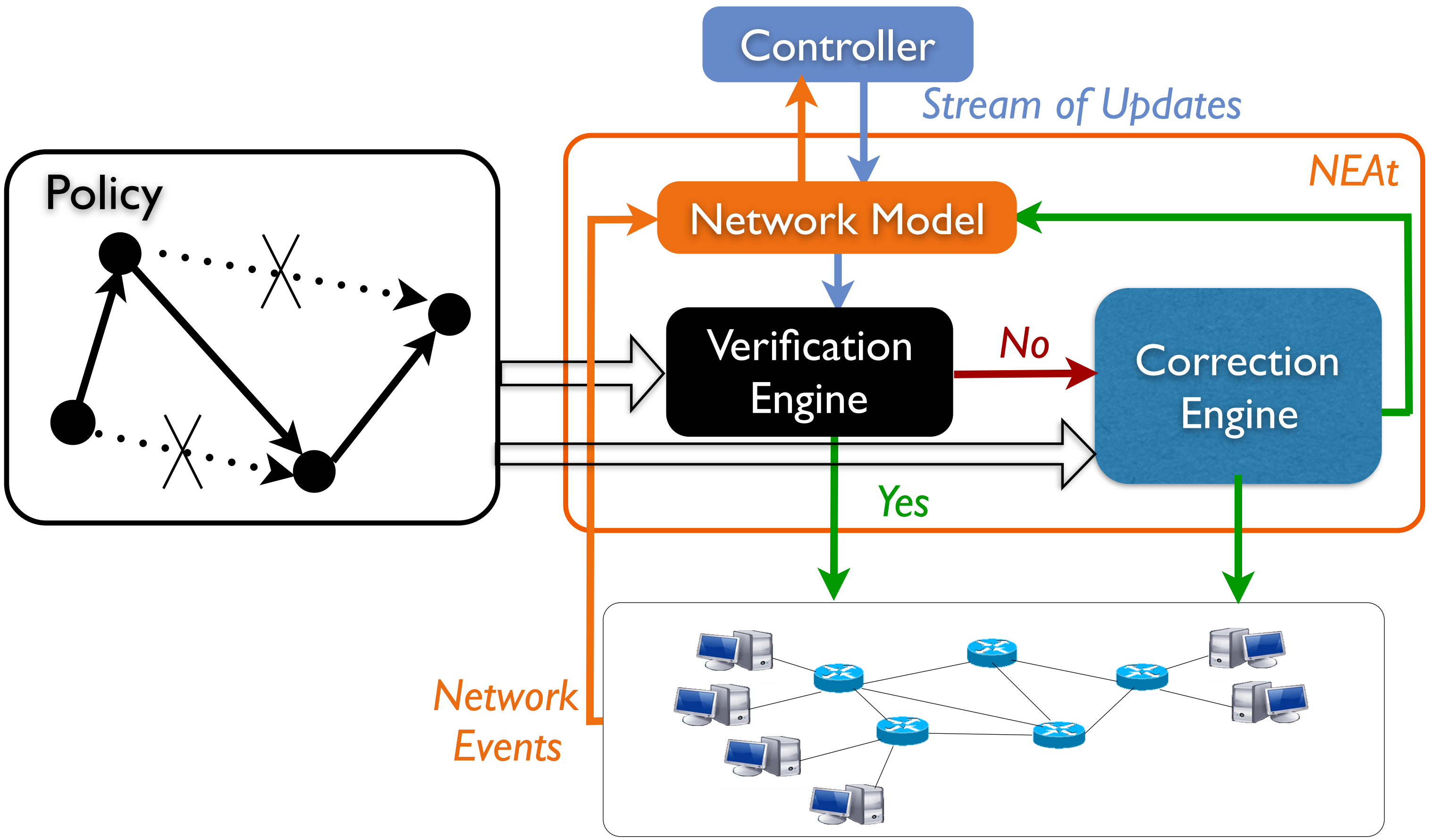
Challenge 2: Zero/minimal architecture/application changes

- Minimal changes
- Pass-through mode
- Interactive mode

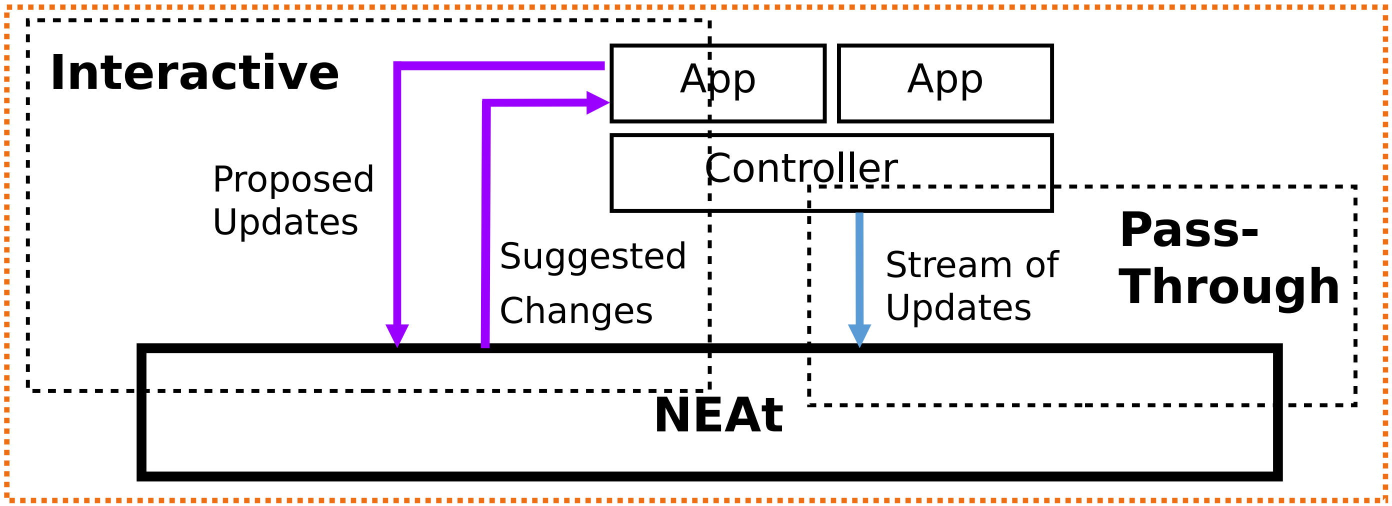
Design of NEAt



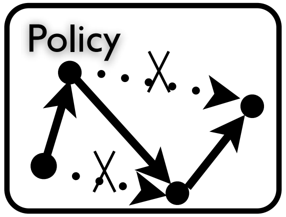
Design of NEAt



Application Mode



Policy as Graphs

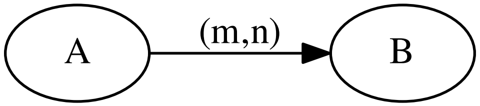


Graphs are ***neat***

- Network state synthesis → viewing the network as a whole.
- Graphs → richer set of policies.

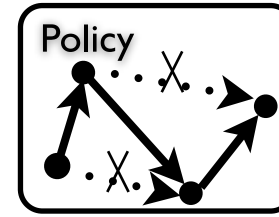
A *policy graph* is defined on a packet header pattern

- ip dst 10.0.1.0/24, port 443.

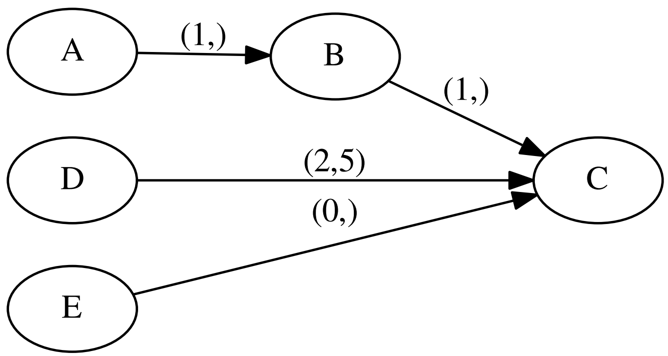


Reachability	m = 1
Bounded path length (shortest path)	m =1 n = path_length
Multipath/Resillience	m = k (k > 1)
Isolation	m = 0

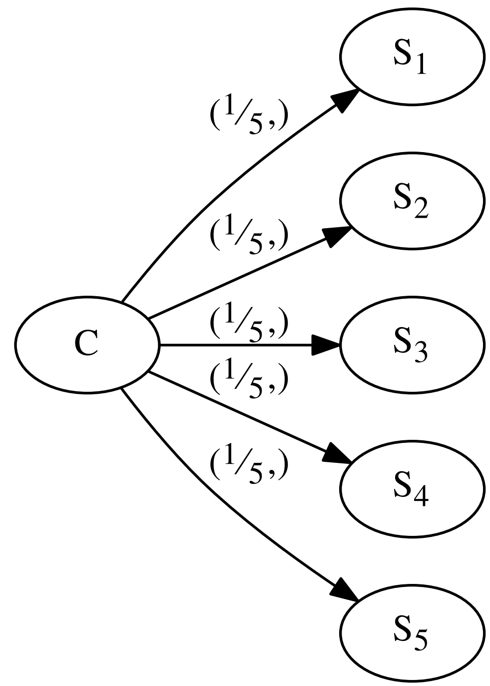
Policy as Graphs (Cont'd)



Service Chaining



Load balancing



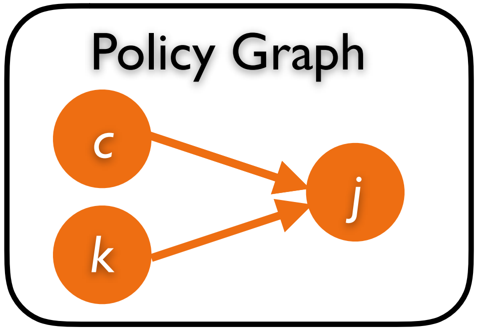
Use *policy graphs* to express both <u>qualitative</u> and <u>quantitative</u> reachability constraints

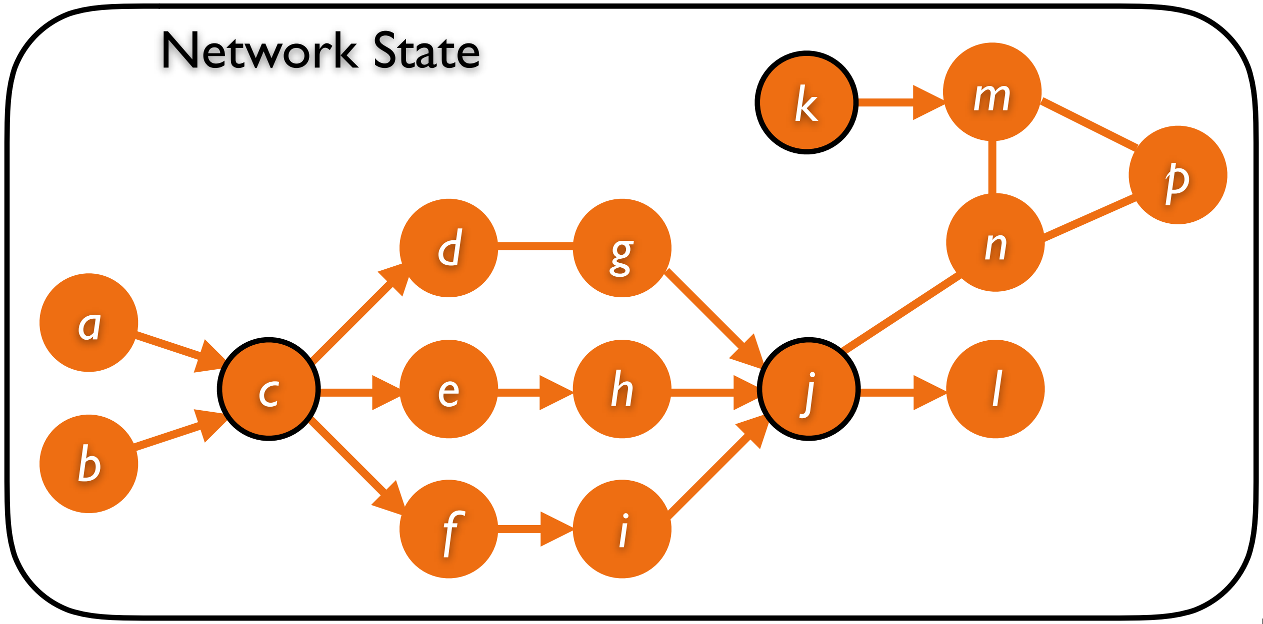
Repair Algorithm

Correction Engine

Cast the problem as an optimization problem:

- Map forwarding graph to policy graph
- Minimize # of changes



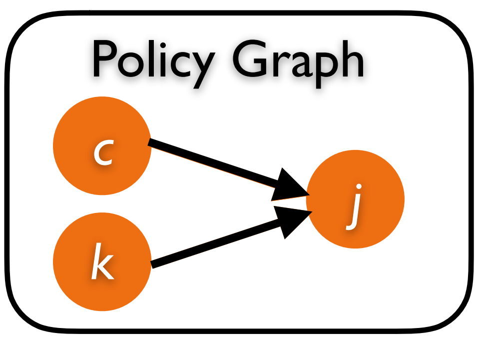


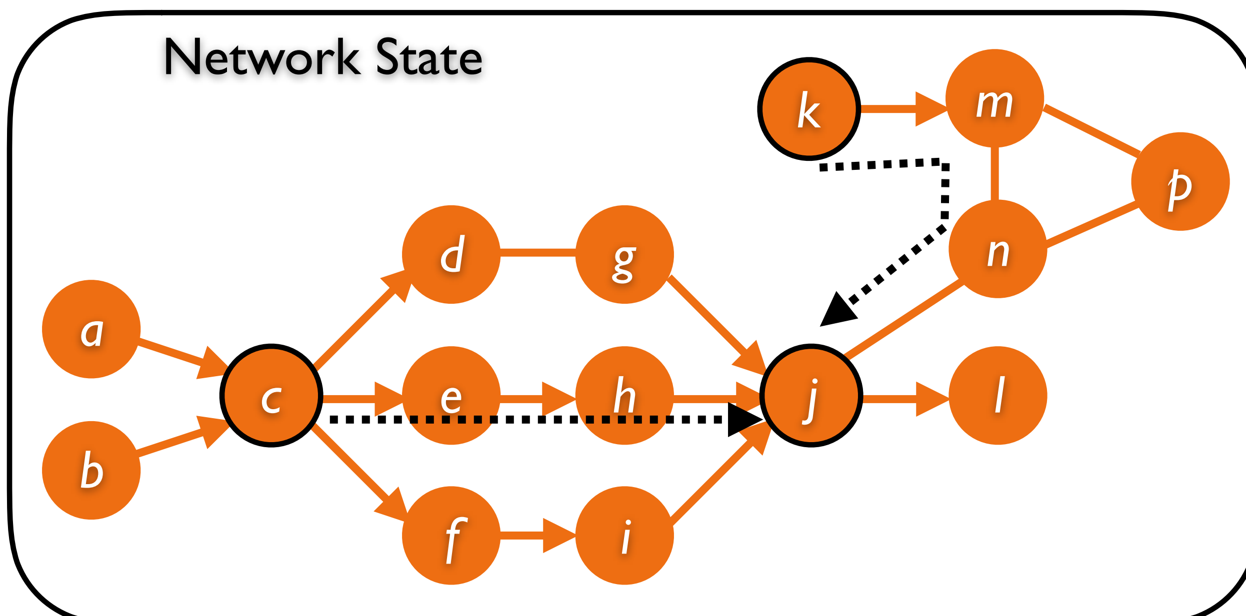
Repair - Basic Reachability

Correction Engine

Cast the problem as an optimization problem:

- Map forwarding graph to policy graph
- Minimize # of changes
- boolean variable $x_{i,j,p,q}$:
  - *topology edge (i, j)* → *policy edge (p, q)*
- *s.t.*, policy level reachability *(p, q)*





Repair - Generalized Reachability

Basic Reachability

$$\forall(i,j) \qquad x_{i,j} \ge \sum_{(p,q)\in E_{\wp c}} \frac{x_{i,j,p,q}}{N(E_\wp)}$$

No tight loops

$$\forall(j,i) \qquad x_{j,i} \ge \sum_{(p,q)\in E_{\wp c}} \frac{x_{j,i,p,q}}{N(E_\wp)}$$

$$\forall(j,i) \qquad x_{i,j} + x_{j,i} \le 1$$

$\forall(p,q), \forall i \in T:$

$$\begin{cases} \sum_{j\in NB_T(i)} x_{i,j,p,q} = 1 \\ \sum_{j\in NB_T(i)} x_{j,i,p,q} = 0 \end{cases} \qquad \text{if } i = p$$

Flow conservation

$$\begin{cases} \dots \\ \sum_{j\in NB_T(i)} x_{j,i,p,q} = 1 \end{cases} \qquad \text{if } i = q$$

$$\left\{ \sum_{j\in NB_T(i)} (x_{i,j,p,q} - x_{j,i,p,q}) = 0 \right. \qquad \text{otherwise}$$

- Isolation

$$\begin{cases} \sum_{j\in NB_T(i)} x_{i,j,p,q} = 0 \\ \sum_{j\in NB_T(i)} x_{j,i,p,q} = 1 \end{cases} \qquad \text{if } i = DROP$$

Flow sinks at DROP node

$$\left\{ \sum_{j\in NB_T(i)} x_{i,j,p,q} = 0 \right. \qquad \text{if } i = q$$

- Service Chaining

$$\begin{cases} \sum_{j\in NB_T(i)} x_{i,j,p,q} = 0 \\ \sum_{j\in NB_T(i)} x_{j,i,p,q} = 0 \end{cases} \qquad \text{if } i \in \wp_c \text{ and } \dots p_{q,i})$$

Correct waypoint order

- Bounded or Equal Path Length

$$\sum_{(i,j)\in E_T} (x_{i,j,p,q} + x_{j,i,p,q}) \le n$$

path length <= n

- MultiPath (Resilience)

$$\begin{cases} \sum_{j\in NB_T(i)} x_{i,j,p,q} \ge m \\ \sum_{j\in NB_T(i)} x_{j,i,p,q} = 0 \end{cases} \qquad \text{if } i = p$$

# of paths >= m

$$\begin{cases} \sum_{j\in NB_T(i)} x_{i,j,p,q} = 0 \\ \sum_{j\in NB_T(i)} x_{j,i,p,q} \ge m \end{cases} \qquad \text{if } i = q$$

- Load Balancing

$$\prod_{x_{i,j,p,q}\ne 0} x_{i,j,p,q} = m$$

Flow distribution propagates

Missing anything?
Repairing Loop-Freedom

Correction Engine

The preceding algorithm operates on a *loop-free* graph.

First check for and remove loops before repairing other type violations.

Objective: Minimize changes

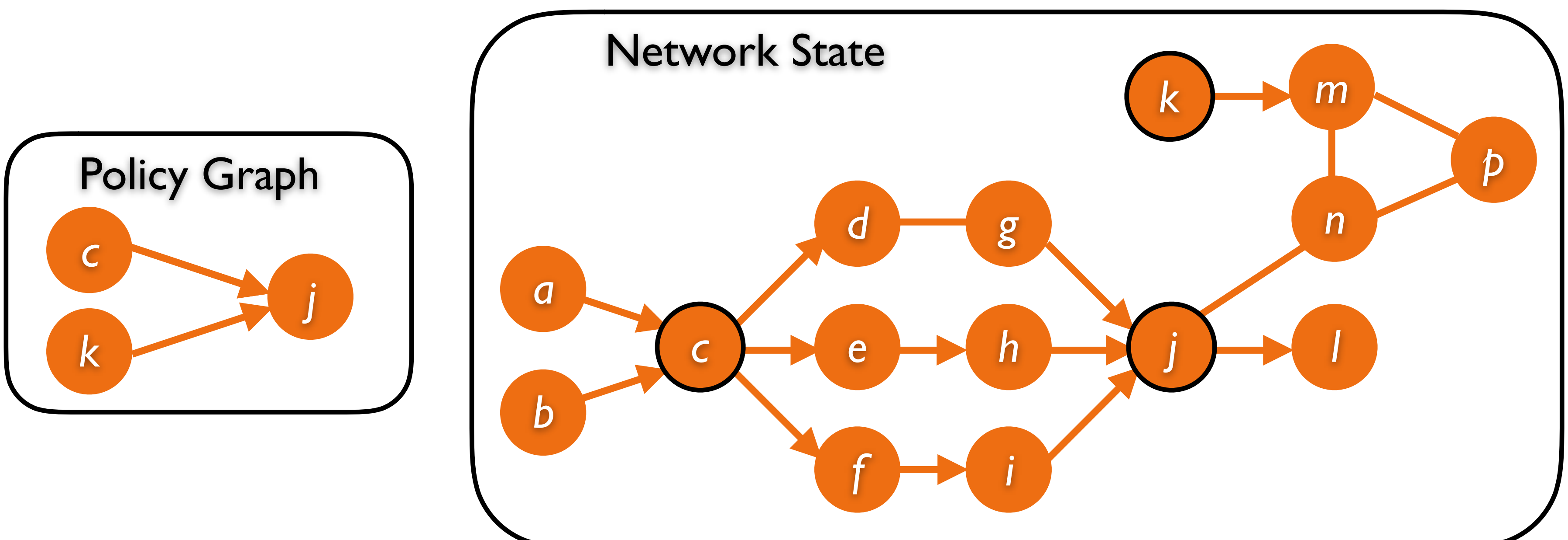
- Remove the minimal # of rules.
- Affect few packets as possible.
    - E.g. remove a rule matching 10.0.0.1/32 over one for 10.0.0.0/8.

Scalability Challenge and Solution

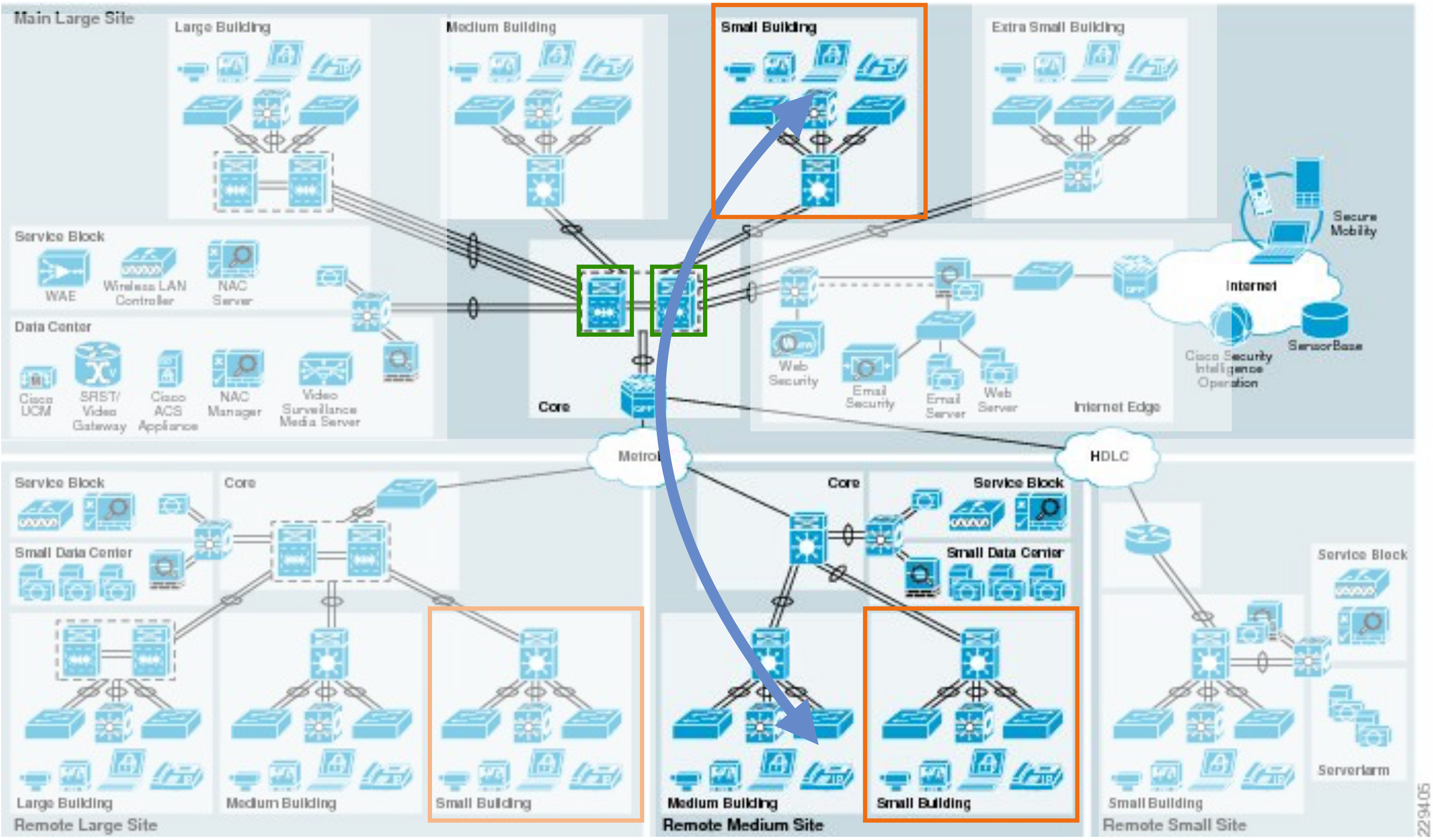
Scalability challenge

- # of variables ≈ $|E(G_{topo})| \times |E(G_{policy})|$
- Easily exceeds 100k

Solution: ?

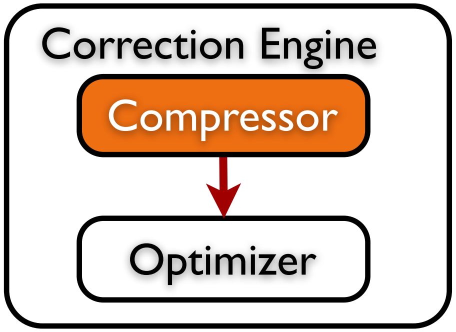


A Typical Enterprise Network



Source: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Medium_Enterprise_Design_Profile/MEDP/chap5.html

Scalability Challenge and Solution



Scalability challenge

- # of variables ≈ $|E(G_{topo})| \times |E(G_{policy})|$
- Easily exceeds 100k

Solution

- Topology Limitation
- Graph Compression *w.r.t policy*
  - Key: Compressed graph == original graph
  - Bisimulation Based Graph Compression

Implementation & Evaluation Setup

Prototype implementation in Python

- Use Gurobi within optimization engine
- Pass-through mode: proxy
- Interactive mode: XML-RPC API

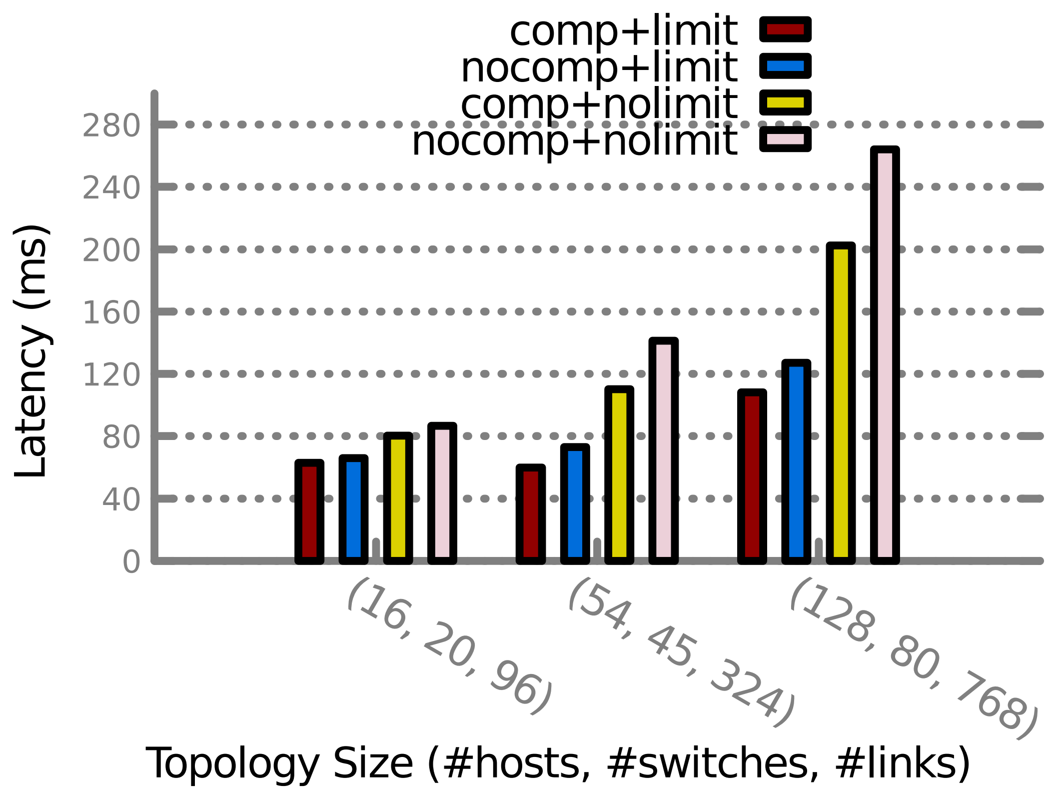
Datasets:

- Synthetic fat-tree configurations
- SDN applications
- 244-router enterprise network trace

Application End-to-End Delay

Pox + Mininet

- Learning switch app (pass-through)
- Load balancing app (interactive)



Enterprise Network Trace Study

244 routers, one million forwarding rules

Policy: loop freedom & reachability

Issues found and repaired:

- Loops caused by default route
- Load balancing shouldn't be turned on

Repair vs. Synthesis

Synthesizer (NetGen) as repair tool

#TopoLinks	NEAt	NetGen	NetGen-C
96	5.9ms	743.2ms	513.2ms
324	7.2ms	4404.0ms	1160.8ms
768	9.0ms	16337.7m	2056.3ms

NEAt as synthesizer

#TopoLinks	NEAt	NetGen
96	921.7ms	7.1min
324	16.3s	381.7min
768	2.9min	173.2hrs

Isn't that NEAt?

NEAt, a system analogous to a smartphone's autocorrect.

- Casting the problem as an optimization problem
- Millisecond to second repair speed
- Generic policy support

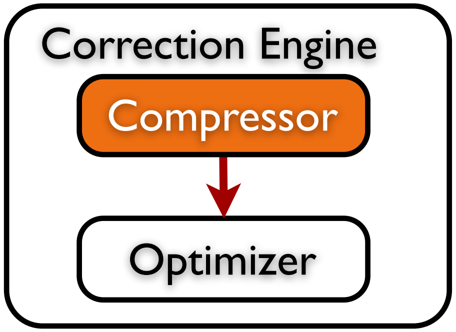
Future work:

- Evolving & richer policies
- Different optimization goals
- Repair relevance study

Graph Compression

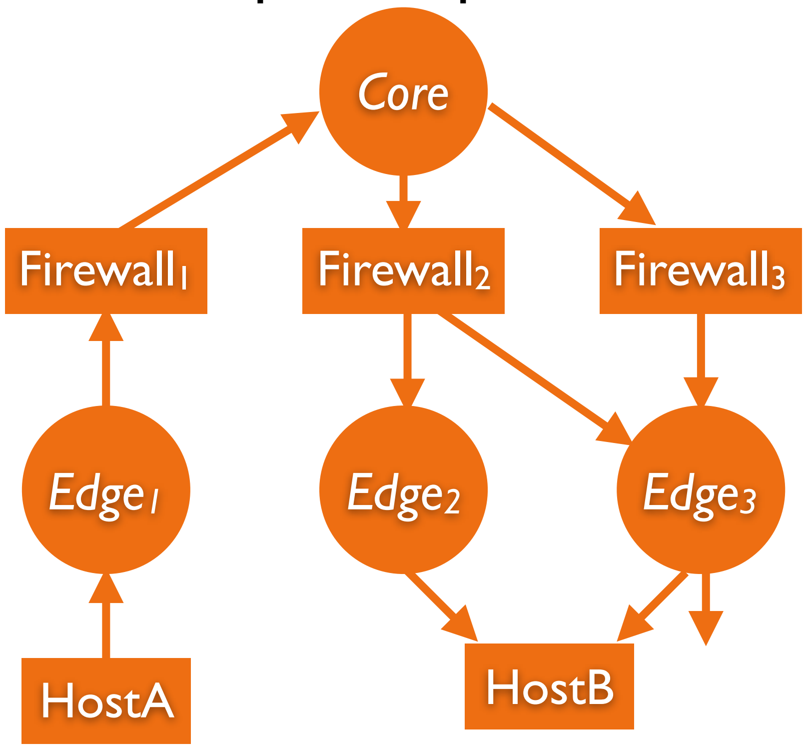
w.r.t policy

Key: Compressed graph == original graph



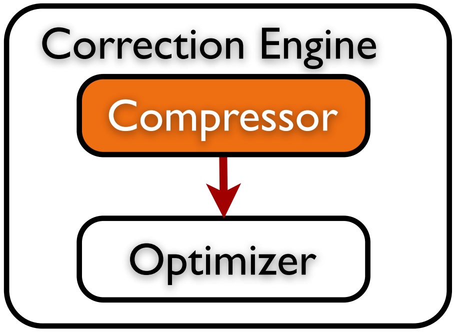
Major building block:

- Bisimulation Based Graph Compression*



*Query preserving graph compression. *SIGMOD 2012. W. Fan et al*

Graph Compression (Cont'd)



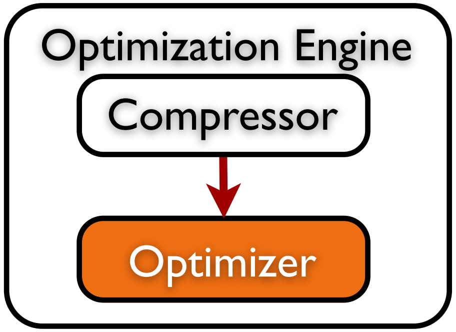
- Customized policy-preserving compression

Topology	Compression Ratio
Fattree (6750 hosts, 1125 switches)	99.38%
Enterprise (236 routers)	88.98%

- Incremental Compression

- Repair compressed graphs

  Modified mutipath constraint



- Mapping back

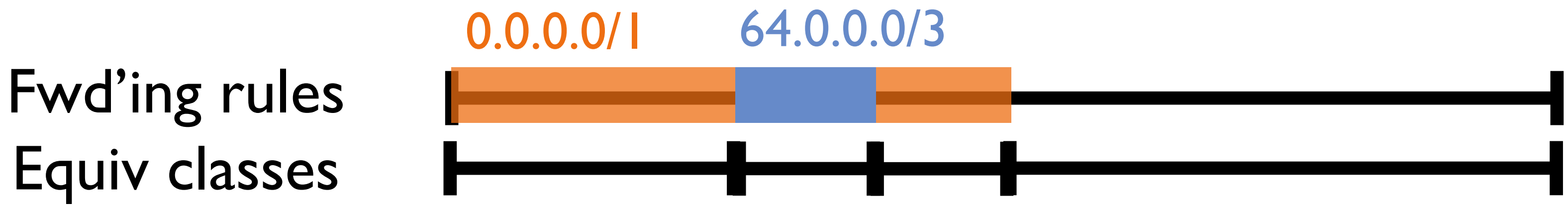
- Proved Policy Perseverance

Backup: Network Model

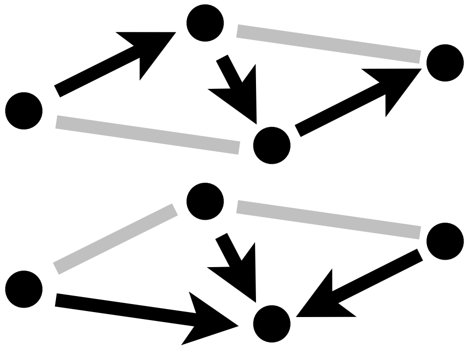
Correction Engine

Model packet space as a set of Equivalence Classes

Equivalence class (EC): Packets experiencing the same forwarding actions throughout the network.



Model forwarding behavior of each EC as a directed graph

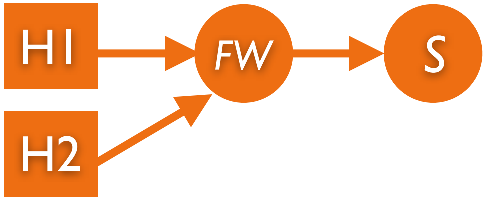


Insights Behind NEAt

Preventing errors at run-time

- Allow arbitrary SDN applications to run on top
- Not restricted to any programming language
- Influence updates, rather than synthesize from scratch

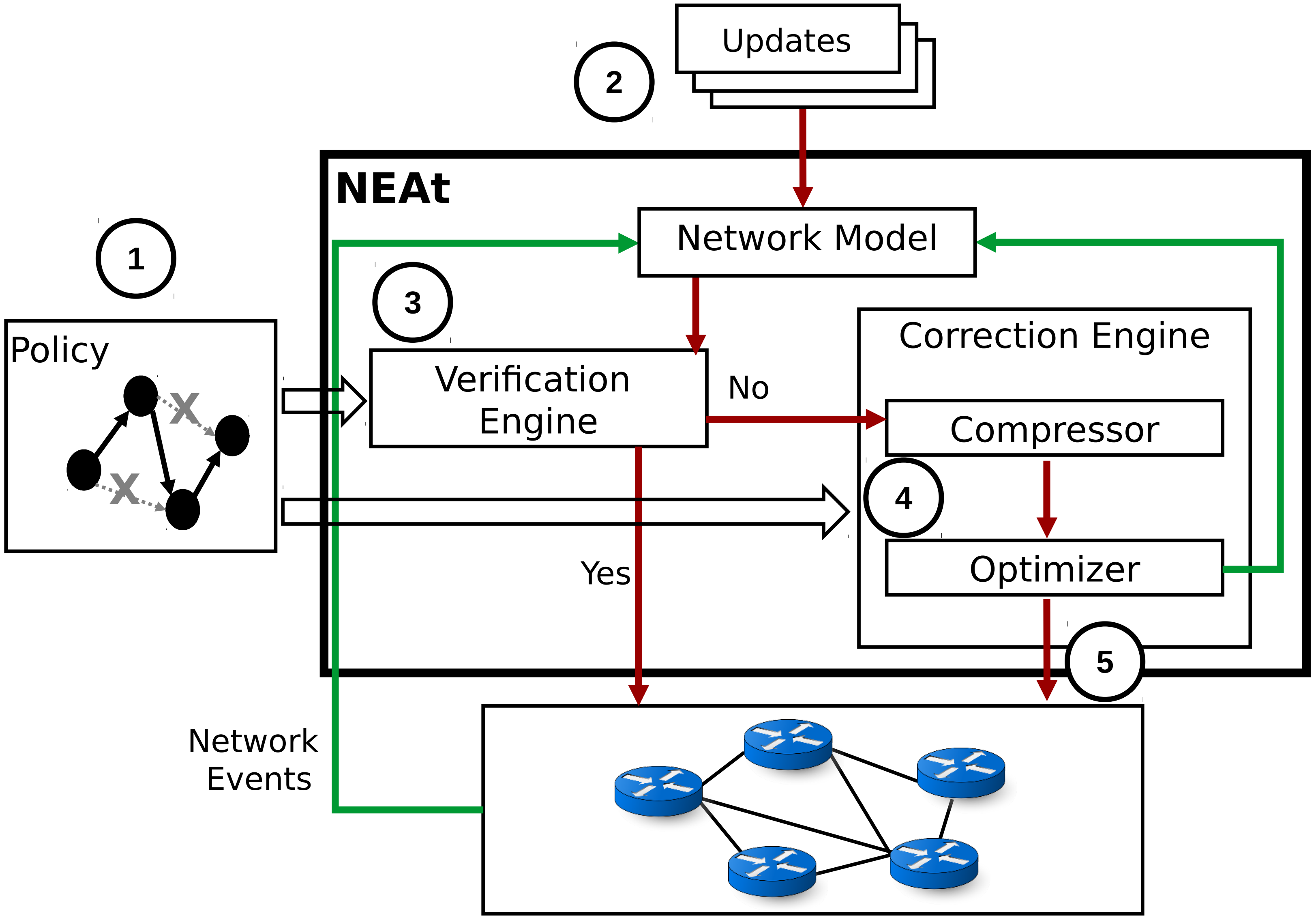
Graphs are ***neat***



- Networks are graphs
- Model network forwarding behaviors as directed graphs
- Represent operator intents as a *policy graph*

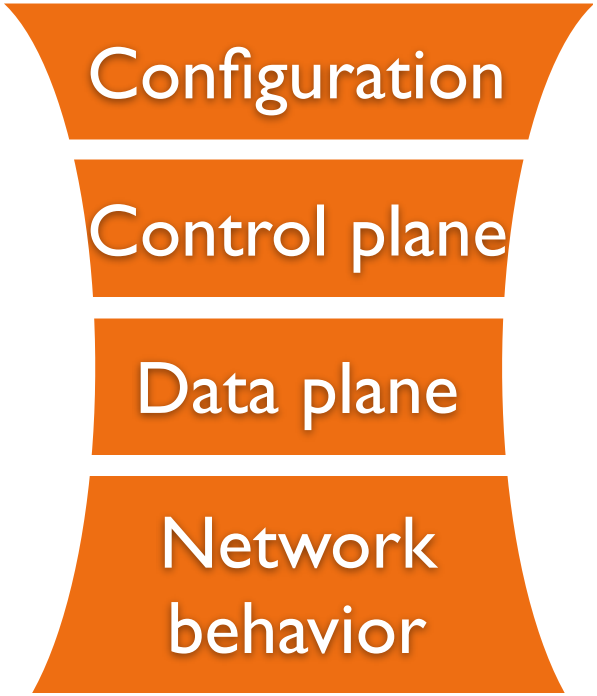
Discovering repairs

- Equivalent to modifying network state graph so that there exists a mapping between it and the policy graph



Updates
2
NEAt
Network Model
1
3
Correction Engine
Policy
Verification Engine
No
Compressor
4
Optimizer
Yes
5
Network Events

Approach: Data plane verification & repair



Configuration	Data-plane
Prediction is difficult: • Various configuration languages • Dynamic distributed protocols	Closer to actual network behavior
	Unified analysis for multiple control-plane protocols
Misses control-plane bugs	Can catch control-plane bugs
Test prior to deployment	Only detects bugs that are present in the data plane

Operating on the data plane simplifies our work

- *Diagnose problems as close as possible to actual network behavior*
- *Data plane is a "narrower waist" than configuration*