

Anemone: Overview

An Edge-based Network Management Architecture

Richard Mortier · Rebecca Isaacs · Austin Donnelly · Paul Barham

Network management is hard!

- Large (10^5 hosts, 10^3 routers), heterogeneous
- Distributed protocols (IP, OSPF, BGP)
- Continuous failure, reconfiguration and growth

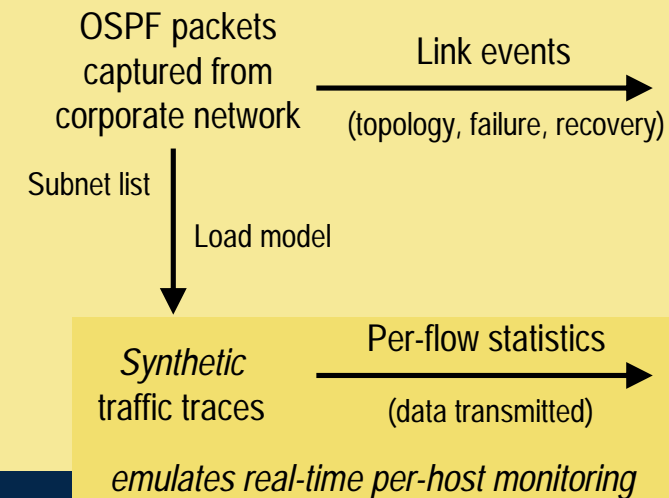
State of the art

- Get topology via recursive use of *ping* and *traceroute*
- Get traffic data from routers using *SNMP* and *NetFlow™*
- Wrap it all up in a GUI with graphs, top-10s, etc

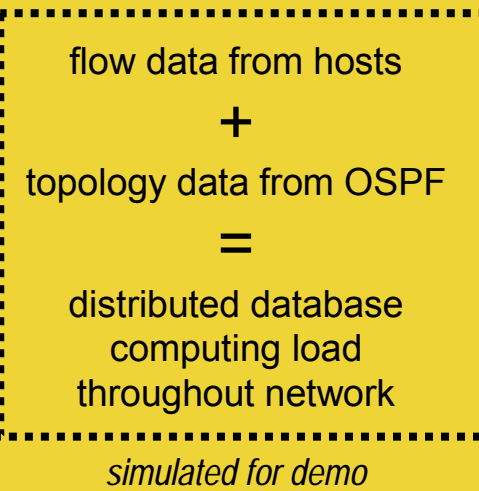
But...

- Opaque protocols (e.g. IPSec)
- Buggy SNMP data
- Resource starved routers

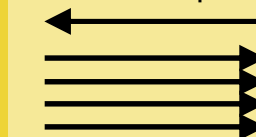
Demo overview



Anemone platform



Continuous queries



results

One-shot queries



Sample management application

Requirements

- What is happening?
- What shouldn't be happening?
- What will be happening?

Potential applications

- Visualization* of current network state
- Analysis* of flow data for intrusion detection
- Simulation* of reconfiguration/failure for planning

Benefits

- Visibility* into normally opaque protocols: IPSec, PPTP
- Independent* of poorly supported management APIs
- Resources* plentiful at hosts dealing with own traffic

Anemone: System challenges

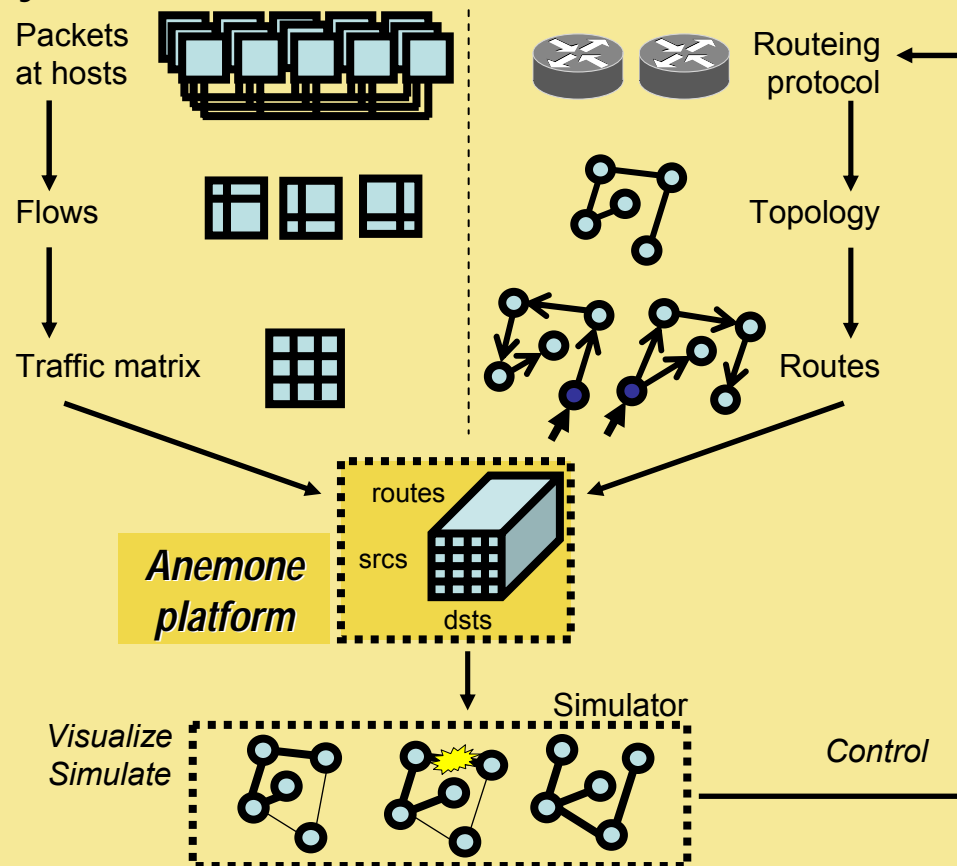
An Edge-based Network Management Architecture

Richard Mortier · Rebecca Isaacs · Austin Donnelly · Paul Barham

How routers route IP using OSPF, 101

Discover *neighbouring routers* and *advertise* throughout network
Collect adverts to build *topology*, compute *best paths* to destinations
Chose destination network via *longest prefix match*, forward packets

System outline



Anemone platform properties

Distributed database, logically containing the *traffic matrix*, each entry annotated with the current *source-destination (src-dst) route*
Redundancy: only a subset of the flow data is required
Flexibility: "flow" is IP 5-tuple, src-dst pair, src-dst-app triple, or ...?

Design questions

Can we better the accuracy/overhead trade-off of e.g. NetFlow™?
Should we distribute data or queries or both? If so, then how?
How much aggregation is appropriate? If any, then where?

Scalability: want to manage a 300,000 node network...
...minimizing overheads (CPU, bandwidth, latency)
Robustness: must work if nodes fail or network partitions...
...as well as deal with L4 routing, NAT, transparent proxies
Accuracy: will not be able to monitor 100% of traffic...
...by accident (unmanaged hosts) or design (aggregation)
Control systems: aiming for real-time network optimization...
...but at what timescales and to what benefit?

Sample queries

"Who are the top-10 traffic generators?"
Easy to aggregate, don't care about topology
"What is the load on link ℓ ?"
Can aggregate from hosts, but need to know routes
"What happens if we remove links $\{\ell...m\}$?"
Interaction between traffic matrix, topology, flow control