NetQuery: A Knowledge Plane for Reasoning about Network Properties

Alan Shieh†, Emin Gün Sirer, Fred B. Schneider

Department of Computer Science
Cornell University

†Nicira Networks
Problem

- Existing networks do not provide mechanisms for querying the properties of network participants
  - All networks look the same
  - All clients look the same
  - No differentiation between network operators
No mechanisms for querying network properties

Clients cannot differentiate between different networks

Identical WiFi base stations!

TinCan ISP
1 Mb/s
90% uptime

Proper ISP
100 Mb/s
99% uptime
No mechanisms for querying network properties

Networks cannot differentiate between clients

Identical end-points!
No mechanisms for querying network properties

Networks cannot differentiate between other networks
Other examples

- What are the instantaneous performance properties of my ISP?
- Does my route from London to Langley go through China? (or, from Taiwan to Beijing through the US?)
- Do my ISPs failover paths provide sufficient capacity?
- Is my cloud provider’s oversubscription within SLA limits?
- Are mutual backup links used only in response to failure?
Commoditization of Networks

- Networks have vastly different properties depending on their composition, configuration, management
  - Yet network operators cannot communicate these effectively

- Much past work aims to discover these properties from the data plane
  - Often requires smarts & hacks
  - Sometimes not possible
Goals

• A system to check if a network possesses a property of interest

• Goals:
  – Trustworthy
  – Privacy-preserving
  – Scalable
  – Federated
  – Extensible
A Knowledge Plane

- A knowledge plane provides an interface and protocol for querying the network for meta-information

- A global, federated tuple-store that contains information about network elements
  - Physical devices, e.g. routers, switches, hosts, etc.
  - Virtual entities, e.g. flows, principals, ASes, etc.

- Information about each network element is stored in an associated **tuple**
Tuplespace Example

\[ \text{H1: Type = Host} \]
\[ \text{H1: OS = ...} \]

\[ \text{R1: Type = Router} \]
\[ \text{R1: FwdTable = ...} \]
\[ \text{R1: PhyLinks = ...} \]

\[ \text{R2: Type = Router} \]
\[ \text{R2: FwdTable = ...} \]
\[ \text{R2: PhyLinks = ...} \]

\[ \text{R3: Type = Router} \]
\[ \text{R3: FwdTable = ...} \]
\[ \text{R3: PhyLinks = ...} \]
Tuplespace Example

- **H1**: Type = Host
  - OS = ...

- **H2**: Type = Host
  - OS = ...

- **R1**: Type = Router
  - FwdTable = ...
  - PhyLinks = ...
  - Speed = 10 Gb/s

- **R2**: Type = Router
  - FwdTable = ...
  - PhyLinks = ...
  - Speed = 10 Gb/s

- **R3**: Type = Router
  - FwdTable = ...
  - PhyLinks = ...
Tuple Abstraction

- A tuple contains **factoids**, attribute-value pairs with an attribution to a source principal

  - **Principal**
  - **Attribute name**: MachineType
  - **Attribute value**: Router

- Tuples are identified by a TupleID
  - Attribute values may reference other tuples

- Standard schemas define the base attributes for classes of network elements, e.g. routers, hosts, links, flows, etc
Tuplespace Implementation

- Tuplespace is federated

- A tuplespace server is just a server at an IP:PORT
  - Provided to devices at boot time
  - TupleIDs are simply IP:PORT:ID, a TupleID is sufficient to retrieve the tuple for that device

- Portions of the space can be delegated to other servers
  - All the way down to individual devices, if desired
Factoid Origins

- Factoids come from a variety of sources
  - NetQuery-aware devices provide their own factoids
  - An SNMP-bridge synthesizes tuples for legacy SNMP devices
  - Network administrators can manually create factoids
  - Third-parties can add factoids post-facto

- Why would anyone trust factoids?

- Why would anyone export factoids?
Trusting Factoids

- Secure coprocessors, such as the Trusted Platform Module, are cheap and ubiquitous.

- TPMs can provide an **attestation chain** to back a statement about an attribute value:
  - Generated by a secure coprocessor, with an embedded secret key.
Attestation Chains

Atmel says TPM speaks for Atmel on TPM.PlatformHash

TPM says TPM.PlatformHash = Hash(IOS)

IOS says IOS.LossRate(\textit{Link1}) = 0.0032
Trusting TPMs

- TPMs provide unforgeable attribution to a key
  - They do not make the platform any more trustworthy than it used to be
  - But they unforgeably identify the platform
  - So a consumer of an attestation chain can make an informed decision

- “Potemkin attacks” are possible
  - But there are countermeasures

- Applications can specify which principals they trust in an import policy
Collecting Factoids

- Factoids are retrieved from the tuplespace servers by **queries**

- **Triggers** enable applications to be notified of changes to an attribute
  - Query-and-Set-Trigger for atomicity

- Received factoids that pass the import policy are loaded into a logic framework
Reasoning with Factoids

• Applications are typically interested in a characteristic, a high-level property
  – E.g. do I have a good VOIP path?

• There are many ways to satisfy a desired characteristic using factoids from the network
  – E.g. “Trusted router reports low loss” or “I have an SLA with this AS”

• NetQuery applications import factoids from the tuplespace into a logical framework, and construct a proof of the characteristic
Factoid Confidentiality

- ISPs do not like to reveal details of their internal operation
  - **Export policy** to restrict principals’ access to factoids
  - **Secure remote evaluation** to confidentially create and check a proof, without leaking any factoids
Factoid Confidentiality

- ISPs do not like to reveal details of their internal operation
  - **Export policy** to restrict principals’ access to factoids
  - **Secure remote evaluation** to confidentially create and check a proof, without leaking any factoids
NetQuery Prototype

- NetQuery client library
- Tuplespace server
- Logic framework and proof checker
- Devices
  - Host
  - Ethernet switch
  - BGP router
  - SNMP proxy
- Applications and proof generators
Example applications

- Network access control
- Topology quality
  - Over-subscription
  - Maximum capacity
  - Failover capacity
- BGP configuration and peering
  - Mutual backup
- AS hop count
- Wi-Fi access point quality
- Network redundancy
## NetQuery Prototype

### Libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server &amp; client</td>
<td>18,286</td>
</tr>
<tr>
<td>Logic Framework</td>
<td>2,254</td>
</tr>
</tbody>
</table>

### Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>543</td>
</tr>
<tr>
<td>Ethernet switch</td>
<td>1,853</td>
</tr>
<tr>
<td>Quagga router</td>
<td>777</td>
</tr>
<tr>
<td>SNMP proxy</td>
<td>1,578</td>
</tr>
</tbody>
</table>

### Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network access control</td>
<td>787</td>
</tr>
<tr>
<td>L2/L3 traceroute</td>
<td>483</td>
</tr>
<tr>
<td>Oversubscription</td>
<td>356</td>
</tr>
<tr>
<td>Maximum capacity</td>
<td>316</td>
</tr>
<tr>
<td>Redundancy</td>
<td>333</td>
</tr>
</tbody>
</table>
Testbed and datasets

- Compute cluster
  - 8-core 2.5 GHz Intel Xeon
  - 1 Gb/s Ethernet
- Departmental network
  - 73 L2/L3 switches (HP & Cisco)
  - >700 end hosts
- RouteViews BGP traces
- RocketFuel router-level topology
Feasibility evaluation

- Tuplespace performance
  - Is query rate suitable for applications?
  - What is the overhead for a typical device?

- Case studies and end-to-end performance
  - ISP topology
  - Deployment in department network
Query microbenchmark

Tuplespace server achieves high throughput

Tuplespace server achieves high throughput
## Analysis performance and overhead: CS department network

<table>
<thead>
<tr>
<th></th>
<th>Completion time (seconds)</th>
<th>Network cost (sent/recv'd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2/L3 traceroute</td>
<td>0.16 s</td>
<td>247 KB</td>
</tr>
<tr>
<td>Oversubscription</td>
<td>(pre-processing) 7.9 s</td>
<td>17 MB</td>
</tr>
<tr>
<td>(per-switch) 0.1 s</td>
<td></td>
<td>0 KB</td>
</tr>
<tr>
<td>Best-case capacity</td>
<td>0.16 s</td>
<td>247 KB</td>
</tr>
<tr>
<td>Redundancy</td>
<td>12.65 s</td>
<td>24 MB</td>
</tr>
</tbody>
</table>

Analyses are suitable for service selection, slow changing topology
ISP topology

Play back RouteViews BGP update traces against BGP router

- Initialization: Load full BGP table (270K prefixes)
- Steady state: Average completion time of update

<table>
<thead>
<tr>
<th></th>
<th>Initialization</th>
<th>Steady state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>5.7 s</td>
<td>62.2 ms</td>
</tr>
<tr>
<td>With NetQuery</td>
<td>13.5 s</td>
<td>63.4 ms</td>
</tr>
</tbody>
</table>

Tuplespace servers can scale to typical POP size

Minimal impact on BGP convergence time.
Summary

- A federated, distributed knowledge plane that:
  - disseminates network properties through a uniform interface
  - incorporates attestation certificates
  - supports reasoning based on certified properties

- Enables many new applications that are not possible today
Proof checking speed: Network Access Control

- Check host process list against policy before allowing access

- Proof size:
  5 from host, 3 attestations

- Check completion time:
  Dominated by verifying digital signatures on attestations

Completion time is appropriate for connect-time policy enforcement
ISP topology

Simulated IGP failures in RocketFuel topology

- Simulated whole POP on one machine
- 5% link failure rate

Global convergence time increase:

<table>
<thead>
<tr>
<th>Mean</th>
<th>0.24s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>0.14s</td>
</tr>
</tbody>
</table>

Convergence time within ISPs' operational goals (< 1s)