Query Your Network Like a Database

Andreas Voellmy, PhD
Forward Networks
Andreasvoellmy@forwardnetworks.com
@AndreasVoellmy
Once Upon a Time, a Service Provider Came To Us ...

Simple. What could possibly go wrong?

Common Parent Org

- Upstream Network
- Downstream Network

BGP advertisements

Propagate to rest of network and Internet
BGP Import Policy Filters

Import policy:
+ filter if community X present
+ filter if AS path matches some pattern
+ routes introduced by other protocols

Policy change
+ starts using community X
+ starts prepending and triggers pattern
+ starts announcing new route
Policy Problem May Be Latent

Problem only becomes apparent when failover occurs

SLAs mean this is an expensive outage
Plan: Proactively Detect The Problem

Upstream

- Advertised BGP routes
- IPv4 FIB

Downstream

- Advertised routes not installed at U
Simple, Important, but Hard to Answer

+ Hard to answer these questions on a large (10K+ devices), heterogeneous (100s of vendor/OS combinations) network.

+ NetConf and other APIs are not widely supported on today’s networks.

+ The only workable solution for network operators today:
  + SSH and grab text
  + Parse poorly-documented, unstructured outputs
  + Organize the data set ...
Example: Get interface status on two devices

Cisco NX-OS: two commands needed

interface Ethernet1/3
  shutdown
  switchport mode private-vlan host
  switchport access vlan 50
  speed 1000
  switchport private-vlan host-association 50 2000

interface Ethernet1/4
  switchport mode private-vlan trunk secondary
  speed 1000
  no shutdown
  switchport private-vlan trunk native vlan 2000
  switchport private-vlan trunk allowed vlan 1000,2000
  switchport private-vlan association trunk 50 1000

A10: one command

Ethernet 10 is up, line protocol is up
Hardware is 10Gig, Address is 001f.a011.8dde
Member of L2 Vlan 601, Port is Tagged
Flow Control is disabled, IP MTU is 9216 bytes
Member of trunk group 1

<table>
<thead>
<tr>
<th>Trunk ID</th>
<th>Member Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trunk Name</th>
<th>Trunk Status</th>
<th>Trunk Type</th>
<th>Admin Key</th>
<th>Members</th>
<th>Cfg Status</th>
<th>Oper Status</th>
<th>Ports-Threshold</th>
<th>Working Lead</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Up</td>
<td>Dynamic (LACP)</td>
<td>1001</td>
<td>9-10</td>
<td>Enb Enb</td>
<td>Up</td>
<td>None</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trunk ID</th>
<th>Member Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trunk Name</th>
<th>Trunk Status</th>
<th>Trunk Type</th>
<th>Admin Key</th>
<th>Members</th>
<th>Cfg Status</th>
<th>Oper Status</th>
<th>Ports-Threshold</th>
<th>Working Lead</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Up</td>
<td>Dynamic (LACP)</td>
<td>1016</td>
<td>1-2</td>
<td>Enb Enb</td>
<td>Up</td>
<td>None</td>
<td>2</td>
</tr>
</tbody>
</table>

Different commands to run and formats to parse, even for the most basic data.
Simple, Important, but Hard to Answer

+ “interface status, BGP session, ... we could deploy a person for 6 months to do this.... 80% of the effort is collecting and parsing...”

+ The work would be duplicative

+ ... but in fact mostly does not get done: operators are mostly not programmers and are otherwise busy fixing stuff.
Let’s Rethink This

What if we had a database of network information, and we could just query it?
“Are there any BGP routes advertised by my downstream BGP routers that are not installed in their upstream router’s FIB?”
Many Other Examples

+ Ad-hoc questions:
  + Where have we defined VLAN 100?

+ Desired invariants:
  + Do all my connected interfaces use the same MTU?

+ Bad states:
  + Are any of my expected BGP sessions in a bad state?
Forward Network Query Engine (NQE)

Forward provides access to structured, normalized data about the network, so that users can query their network like a database.
NQE: Query Your Network Like a Database!

Vendor-specific configuration files and state information

Structured, normalized schema based on Forward data model

Possible query of network schema and results

Sample query (concept):

"Which devices have interfaces with different operational and admin states?"

Formatted results:

<table>
<thead>
<tr>
<th>Device</th>
<th>Interface</th>
<th>Admin Status</th>
<th>Oper Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>at5-lb01</td>
<td>1.3</td>
<td>UP</td>
<td>DOWN</td>
</tr>
<tr>
<td>at5-lb01</td>
<td>1.4</td>
<td>UP</td>
<td>DOWN</td>
</tr>
<tr>
<td>sjc-ce01</td>
<td>ge-0/0/1</td>
<td>UP</td>
<td>DOWN</td>
</tr>
<tr>
<td>sjc-ce02</td>
<td>ge-0/0/5</td>
<td>UP</td>
<td>DOWN</td>
</tr>
</tbody>
</table>
NQE: Single query works on all supported devices

<table>
<thead>
<tr>
<th>Vendors</th>
<th>Vendors</th>
<th>Vendors</th>
<th>Vendors</th>
<th>Vendors</th>
<th>Vendors</th>
<th>Vendors</th>
<th>Vendors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A10</td>
<td>ARISTA</td>
<td>AVI</td>
<td>AWS</td>
<td>Symantec</td>
<td>BLUE COAT</td>
<td>Check Point</td>
<td>CISCO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Networks</td>
<td></td>
<td></td>
<td></td>
<td>SOFTWARE TECHNOLOGIES CO.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUMULUS</td>
<td>f5</td>
<td>FORTINET</td>
<td></td>
<td>Hewlett Packard Enterprise</td>
<td>JUNIPER</td>
<td>palonitto</td>
<td>PICA8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>vmware</td>
</tr>
</tbody>
</table>
NQE Walkthrough: SP Use Case

“Are there any BGP routes advertised by my downstream BGP routers that are not installed in their upstream router’s FIB?”
NQE Walkthrough: Query Editor
NQE Walkthrough: BGP advertised routes
NQE Walkthrough: Get IPv4 Routes

The abstract forwarding tables (AFTs) that are associated with the network instance. An AFT is instantiated per-protocol running within the network-instance - such that one exists for IPv4 Unicast, IPv6 Unicast, MPLS, L2 forwarding entries, etc. A forwarding entry within the FIB has a set of next-hops, which may be a reference to an entry within another table - e.g., where a Layer 3 next-hop has an associated Layer 2 forwarding entry.

**FIELDS**

**ipv4Unicast:** IpUnicast

The abstract forwarding table for IPv4 unicast. Entries within this table are uniquely keyed on the IPv4 unicast destination prefix which is matched by ingress packets. The data set represented by the IPv4 Unicast AFT is the set of entries from the IPv4 unicast FIB that have been selected for installation into the FIB of the device.

**ipv6Unicast:** IpUnicast

The abstract forwarding table for IPv6 unicast. Entries within this table are uniquely keyed on the IPv6 unicast destination prefix which is matched by ingress packets. The data set represented by the IPv6 Unicast AFT is the set of entries from the IPv6 unicast FIB that have been selected for installation into the FIB of the device.
NQE Walkthrough: Query Script

+ Simple, small script:
  + Runs both queries, compares routes, prints violations.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>AS Path</th>
<th>Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.121.121.0/24</td>
<td>[1, 1, 1, 1, 6500, 3356, 42708, 38093]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>31.168.168.0/20</td>
<td>[6500, 3356, 8551, 8551, 8551]</td>
<td>Upstream selects different route: STATIC DROP</td>
</tr>
<tr>
<td>204.235.115.0/24</td>
<td>[1, 1, 1, 1, 6500, 3356, 4323, 3456, 3456, 3456, 3456, 3456]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>216.206.127.0/24</td>
<td>[1, 1, 1, 1, 6500, 3356, 54114]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>37.46.268.0/21</td>
<td>[6500, 3356, 39326]</td>
<td>Upstream selects different route: STATIC DROP</td>
</tr>
<tr>
<td>202.46.240.0/22</td>
<td>[1, 1, 1, 1, 6500, 3356, 2914, 58463, 18059, 3583]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>23.200.36.0/20</td>
<td>[6500, 3356, 2914]</td>
<td>Upstream selects different route: STATIC DROP</td>
</tr>
<tr>
<td>243.113.35.0/24</td>
<td>[1, 1, 1, 1, 6500, 3356, 4637, 1221, 38285, 10113]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>211.118.176.0/24</td>
<td>[1, 1, 1, 1, 6500, 3356, 3491, 9848, 18305, 18305, 18305]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>192.251.17.0/24</td>
<td>[1, 1, 1, 1, 6500, 3356, 22773]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>23.252.160.0/21</td>
<td>[6500, 3356, 4134, 36678, 26484]</td>
<td>Upstream selects different route: STATIC DROP</td>
</tr>
<tr>
<td>202.95.212.0/22</td>
<td>[1, 1, 1, 1, 6500, 3356, 2516, 10021, 10021]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>198.178.192.0/24</td>
<td>[1, 1, 1, 1, 6500, 3356, 701, 702]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>24.142.176.0/24</td>
<td>[6500, 3356, 19090, 53432]</td>
<td>Upstream selects different route: STATIC DROP</td>
</tr>
<tr>
<td>216.57.121.0/24</td>
<td>[1, 1, 1, 1, 6500, 3356, 5738, 26002]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
<tr>
<td>198.136.250.0/24</td>
<td>[1, 1, 1, 1, 6500, 3356, 7018, 2386]</td>
<td>Filtered by upstream BGP import policy</td>
</tr>
</tbody>
</table>

16 rows

Time to implement: 6 months → 1 hour
How to Implement a Normalized Network Database?

In theory, this is simple

Core parts of FN Platform

Collect → Parse

NQE-specific

Normalize → Query

In practice: challenges in every step of the process.
How to Implement a Normalized Network Database?

1. Collect
2. Parse
3. Normalize
4. Query
Operators are not professional programmers; we wanted a query API that was easy to use and required minimal learning.

While other choices may also have worked, GraphQL was a great fit.

“Query language for your API”

Describe your data

```typescript
type Project {
    name: String
    tagline: String
    contributors: [User]
}
```

Ask for what you want

```json
{
    project(name: "GraphQL") {
        tagline
    }
}
```

Get predictable results

```json
{
    "project": {
        "tagline": "A query language for APIs"
    }
}
```
Network data model is elaborate; users need clear definitions and help navigating this.

GraphQL schema language enables us to describe the model simply and clearly.

```graphql
type Ethernet {
  # MAC Address of the Ethernet interface
  macAddress: MACAddress
  # The duplex mode that has been negotiated.
  negotiatedDuplexMode: DuplexMode
  # The interface speed that has been negotiated.
  negotiatedPortSpeed: PortSpeed
  # The logical aggregate interface to which this interface belongs.
  aggregateId: String
  # MAC Address of the Ethernet interface
  switchedVlan: SwitchedVlan
}
```

Great tooling around the schema.
Queries are simple: they just follow the data organization.

Output is JSON and follows the data organization, with values filled in.

```json
{  
  devices { 
    interfaces { 
      name
      operStatus
      adminStatus
    }
  }
}

{  
  "data": { 
    "devices": [
      { 
        "name": "gi0/0/0/0",
        "operStatus": "UP",
        "adminStatus": "UP"
      },
      { 
        "name": "gi0/0/0/1",
        "operStatus": "UP",
        "adminStatus": "DOWN"
      }, ...
    ]
  }
}
```
The largest networks present large datasets:
  + 2M+ routes on a single device
  + 600K+ ACLs on a single device

To handle this, Forward implements custom storage formats and data structures.

GraphQL is agnostic to storage format; allows us to implement queries with custom logic.
How to Implement a Normalized Network Database?

- Collect
- Parse
- Normalize
- Query
We are not interested in re-inventing the wheel here.

We based our schemas on OpenConfig YANG models.

Operator-driven community, with operator-vetted models, with broad coverage.
Marrying OpenConfig with GraphQL

There are some mismatches

+ OpenConfig vs GraphQL naming requirements.
+ Simplified for read-only use case.
+ Leverage GraphQL’s graph database facilities to enable easier linking between objects.
+ Expose paging over large collections.
How to implement a normalized network database?

Core parts of FN Platform

Collect → Parse → Normalize → Query
Parsing: Millions of Patterns

**Scale:** 16 vendors, 23 Oses, 242 OS versions.

**Example:** On just a single device OS (Cisco NX-OS), there are 120k ways of combining keywords into valid top-level config commands.

**Critical:** streamlined way of ingesting text-based data into the model.

One of the major focus areas at Forward.
Collection: Getting the Data in the Real World

All sorts of surprising challenges lurk here.

+ No inventory, no topology

+ Complex infrastructure slows down collection

+ Device failures are common
NQE announced in January this year.

We continue to evolve and improve:

+ Continue to expand the data set

+ Explore ways to simplify and make it easier to query without dropping into scripting.
Thank You

andreasvoellmy@forwardnetworks.com  @AndreasVoellmy

We’re excited to see what the community does with NQE.

+ Blog post: https://forwardnetworks.com/blog/network-query-engine
+ Github repo: https://github.com/forwardnetworks/network-query-engine-examples