Evolvable Network Telemetry at Facebook

Yang Zhou*, Ying Zhang, Minlan Yu*, Guangyu Wang, Dexter Cao, Eric Sung, Starsky Wong
Network Telemetry is Critical for Network Management

- Alerting
- Traffic engineering
- Diagnosis
- Troubleshooting
- Verification
- Asset tracking

Network Telemetry

Billions of time-series counters, Deriving network topology

Millions of device interfaces

Data Centers, Backbones, Edges

Hundreds of thousands of switches/routers
Key Challenge for Telemetry in Production: Evolvability

- Network devices and management applications are constantly evolving.

Application: Increasing Requirements

Network Telemetry: Constantly Evolving

Networks: Heterogeneous HW & SW
Magnitude of Changes

Up to 30 code commits and 1000 LoC changes per week

- Application: Increasing Requirements
- Network Telemetry: Constantly Evolving
- Networks: Heterogeneous HW & SW
- Alerting rules
- Telemetry runtime
- Device-related schemas
Incident 1: Changes Affect Many Components

Takes multiple teams over three days to diagnose.

Even more frequent with FBOSS open switching

Frequent API changes of one component affect many other components
Incident 2: Data Misinterpretation

Frequent hardware and software changes affect data values and semantics.
Bringing Changes to First-Class Citizens in Telemetry

Change Propagation
API changes affect many other components

Data Misinterpretation
Caused by HW and SW changes

Track API changes across components
Build trustful telemetry data despite changes
PCAT: Production Change-Aware Telemetry System

Change abstraction:
❖ Representing changes in a uniform and generic way
✓ Track and use changes easily

Change attribution:
❖ Layering design to clearly attribute changes to the right components
✓ Limit change propagation; track changes clearer

Change exploration:
❖ Allowing applications to explore change dependencies
✓ Improve timeliness/accuracy
Change abstraction:
❖ Representing changes in a uniform and generic way
✓ Track and use changes easily

Change attribution:
❖ Layering design to clearly attribute changes to the right components
✓ Limit change propagation; track changes clearer

Change exploration:
❖ Allowing applications to explore change dependencies
✓ Improve timeliness/accuracy
Change Abstraction: Change Cube

Consider the incident 1: Linecard Version Change $\rightarrow$ Unbalanced Traffic

- **Device**: Linecard version: integer (3) to string (3.0.0)
- **Collection**: Post-processing code cannot recognize
- **Topology**: Missing some linecards and circuits
- **Traffic Engineering**: Unbalanced traffic distribution
Change Abstraction: Change Cube

Consider the incident 1: Linecard Version Change → Unbalanced Traffic
Change Abstraction: Change Cube

Consider the incident 1: Linecard Version Change → Unbalanced Traffic

Change cube: \(<\text{Time, Entity, Property, Layer, Dependency}>\)
PCAT: Production Change-Aware Telemetry System

**Change abstraction:**
- Representing changes in a uniform and generic way
  - ✓ Track and use changes easily

**Change attribution:**
- Layering design to clearly attribute changes to the right components
  - ✓ Limit change propagation; track changes clearer

**Change exploration:**
- Allowing applications to explore change dependencies
  - ✓ Improve timeliness/accuracy
Change Attribution: Three Generations of Telemetry Systems

Gen1: Deeply-coupled script (models + collection code)

Gen2: Semi-modular

Gen3: Fully modular
Change Attribution: Gen1 and Gen2

Gen1: Monolithic collection script

✗ Changes all over the place.

✓ Confine changes to one of the two layers.
✓ Track changes of two layers separately.

Gen2: Decoupled models from collection code

Models

Collection code

Applications

Collection & parsing primitives

Network-wide processing

Device-level processing

Collection infrastructure
Change Attribution: Gen2’s Problems

- Enormous number of models.
- Intents are deeply coupled with the vendor-dependent details: models become hard to define and evolve.

\[
\# \text{ models} = \# \text{ intents} \times \# \text{ vendors} \times \# \text{ device groups}
\]

\[
10k = 100 \text{ alarm rules} \times 10 \text{ vendors} \times 10 (\text{rack, spine, etc})
\]
### Change Attribution: Gen3 (PCAT) Layering Design

#### Gen3: Limiting the impact of changes

<table>
<thead>
<tr>
<th>Intent models</th>
<th>Alert if <strong>Interface.pkt_drops.Rate()</strong> &gt; 1k/s</th>
</tr>
</thead>
</table>
| Data models   | ModelDef(name='**Interface**', properties=[
|               |     PropertyDef(name='**pkt_drops**', type=INT), ...]) |
| Collection models | Vendor1 CLI: show interfaces {$if_name} drops
|                  | Vendor2 Thrift: getQueueDrops({$queues}) |
| Job models     | JobDef(model_name='**Interface**',
|               |   device_group='Rack switches',
<p>|               |   frequency='5min', ... |</p>
<table>
<thead>
<tr>
<th>Models</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intent models</td>
<td>Track change clearer, i.e., to specific layer.</td>
</tr>
<tr>
<td>Data models</td>
<td>Limit change propagation, e.g., easy to adjust collection frequency.</td>
</tr>
<tr>
<td>Collection models</td>
<td>Limit change propagation, e.g., easy to adjust collection frequency.</td>
</tr>
<tr>
<td>Job models</td>
<td>Track change clearer, i.e., to specific layer.</td>
</tr>
</tbody>
</table>
PCAT: Production Change-Aware Telemetry System

**Change abstraction:**
- Representing changes in a uniform and generic way
  - Track and use changes easily

**Change attribution:**
- Layering design to clearly attribute changes to the right components
  - Limit change propagation; track changes clearer

**Change exploration:**
- Allowing applications to explore change dependencies
  - Improve timeliness/accuracy
Change Exploration: Topology Derivation

Creates derived topology from normalized device-level data.

TopoGen V1:
Change Exploration: Topology Derivation

Creates derived topology from normalized device-level data.

TopoGen V2:

- Collection
- Collected interface data
- Dependency
- TopoGen V2 jobs
- Real-time derivation
- Derived circuit

Reduce derivation delay by 118 sec on average
PCAT: Production Change-Aware Telemetry System

**Change abstraction:**
- Representing changes in a uniform and generic way
  - Track and use changes easily

**Change attribution:**
- Layering design to clearly attribute changes to the right components
  - Limit change propagation; track changes clearer

**Change exploration:**
- Allowing applications to explore change dependencies
  - Improve timeliness/accuracy
Open Questions: Adaptive Telemetry Primitives

### Efficiency vs. Adaptivity

- **Efficiency**
  - New microburst detection solution

- **Adaptivity**
  - Only work for certain devices/vendors

**Vendor Comparison**

- **Vendor A**: ✗
  - Partial knowledge causes complexity for applications

- **Vendor B**: ✓
  - Widely adapt to various resource/programming conditions

- **Vendor C**: ✗

---

Need both efficient and adaptive telemetry primitives
Open Questions: Trustful Telemetry

- Telemetry data may get missed/corrupted in evolving environment.
- Business-critical applications (e.g., TE) rely on correct telemetry data.

One opportunity: **cross-validations** for counters

Need telemetry verification and validation
Summary

- Telemetry is critical for network management.
- Changes should be first-class citizens in evolvable telemetry.
- PCAT: Production change-aware telemetry system
  - Change abstraction: change cubes.
  - Change attribution: layering design.
  - Change exploration: change-aware applications.
Thank You!

Icons from Flaticon.com, pngitem.com, veryicon.com, onlinewebfonts.com