OpenKilda

Stream Processing Meets OpenFlow

Jon Vestal

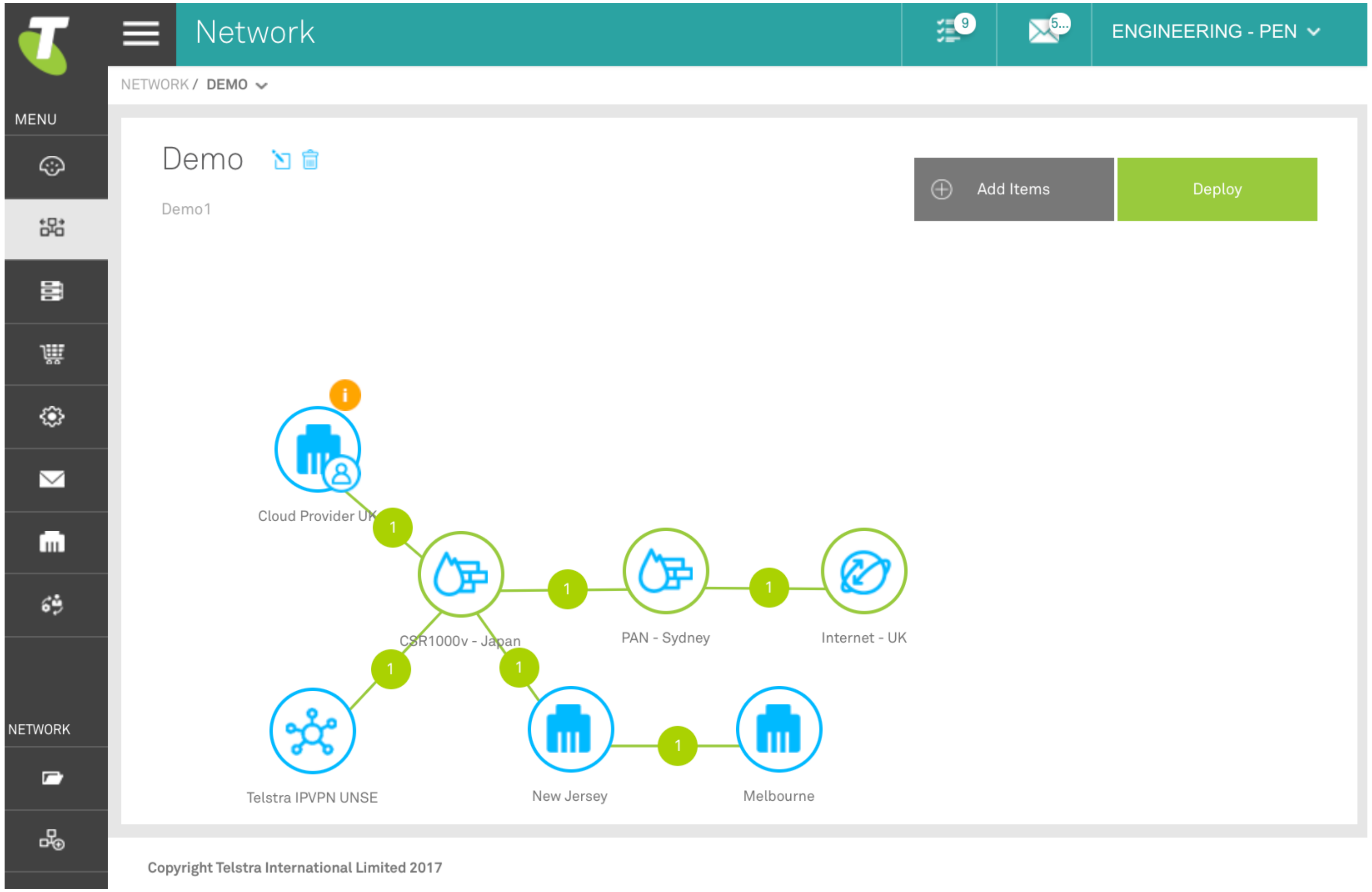
Head of Product Architecture, Global Platforms

TELSTRA

Why Build Yet Another OpenFlow Controller?

A few of the existing controllers available today

TPN Build Blocks



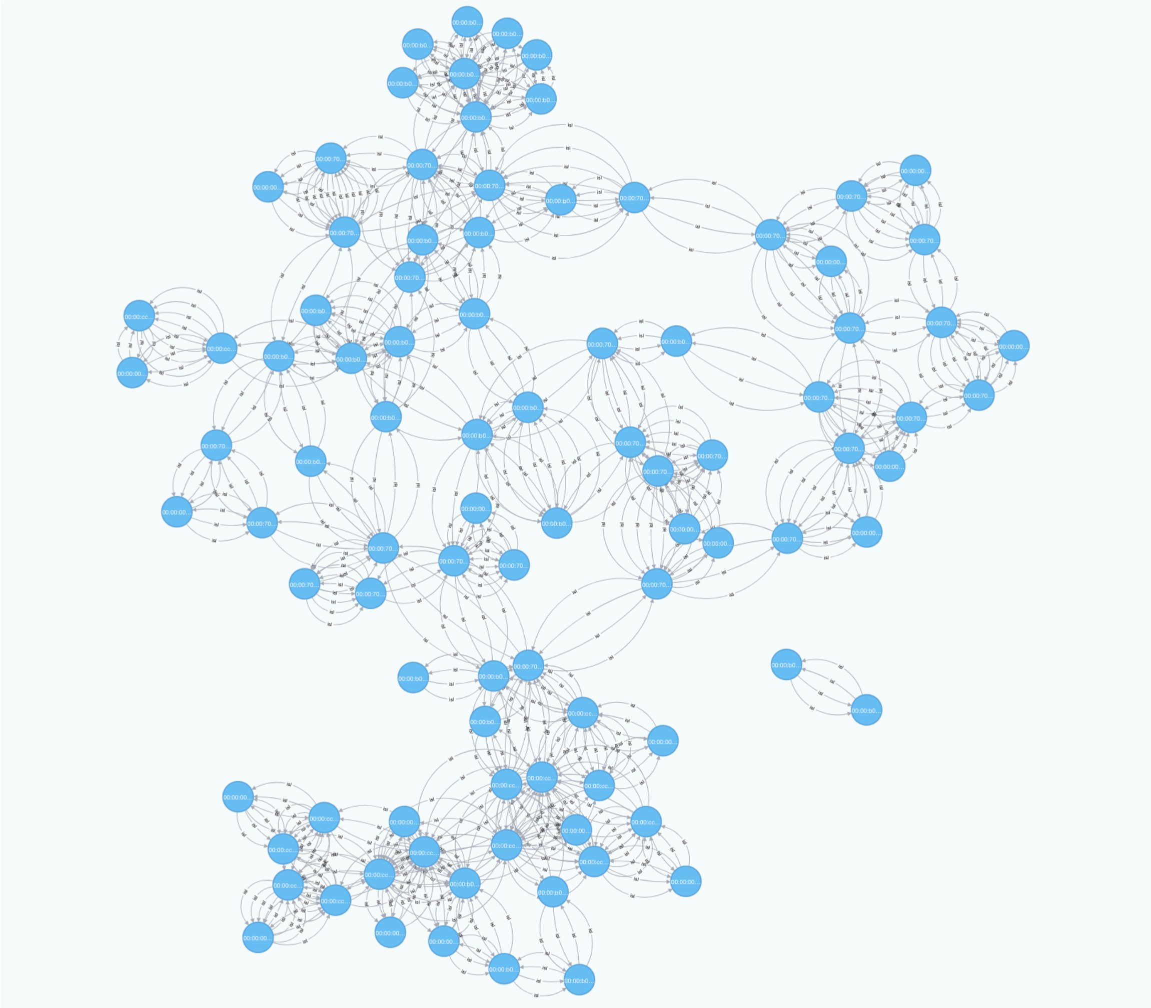
Copyright Telstra International Limited 2017

- Customer Driven
- Building Blocks:
  - IPVPN
  - Exchange
  - NFV
  - Internet
  - Switch Ports

TELSTRA

Our Challenge Was A Bit Unique

At least we thought it was



- Global network with POPs in Europe, US, Asia, Australia and Middle East
- Control Plane with >300ms of latency
- Controllers located in Hong Kong
- Combination of Dark Fiber and Lit Circuits that don't all support Link Loss Forwarding

Features We Wanted

Sub-Second Failover

Negative Affinity In Path Selection

Active Latency Measurement on ISL

End-to-End Latency Measurement on Flow

Path Selection Based on Latency

Auto-re-route based on real-time latency/packet loss/jitter measurements

Multiple data points for comprehensive end-to-end network state

Horizontal scale

- Number of switches
- Number of flows

Complex match/actions using experimenters

Stats collections at 1 second intervals

Self Healing/Optimizing Network

Zero Touch Controller Deployment/Upgrade

What We Found

- Constant topology changes
- Network changes increased with network complexity
- Correlation of multiple events

Convergence

- 100K’s messages into/out of the controller
- Managing >1M Flows

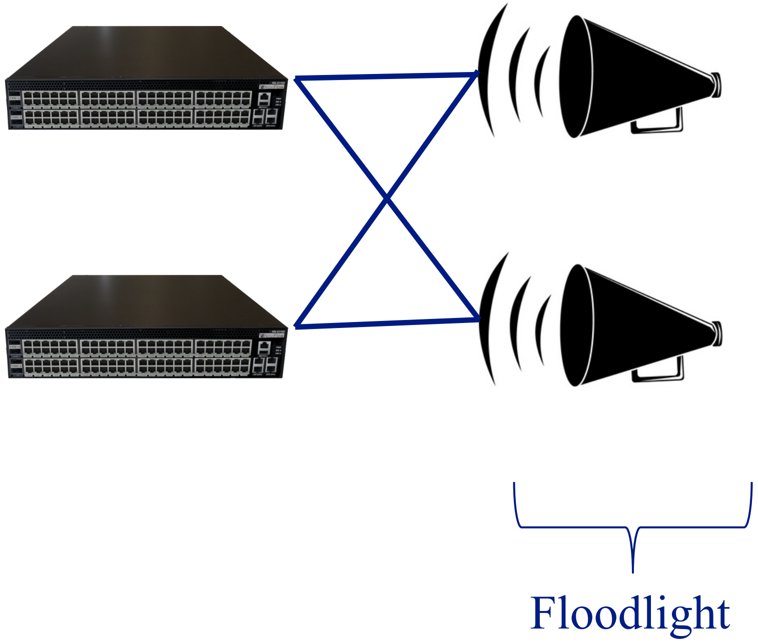
Events

- LAN based controllers
- High latency in Control Plane

WAN

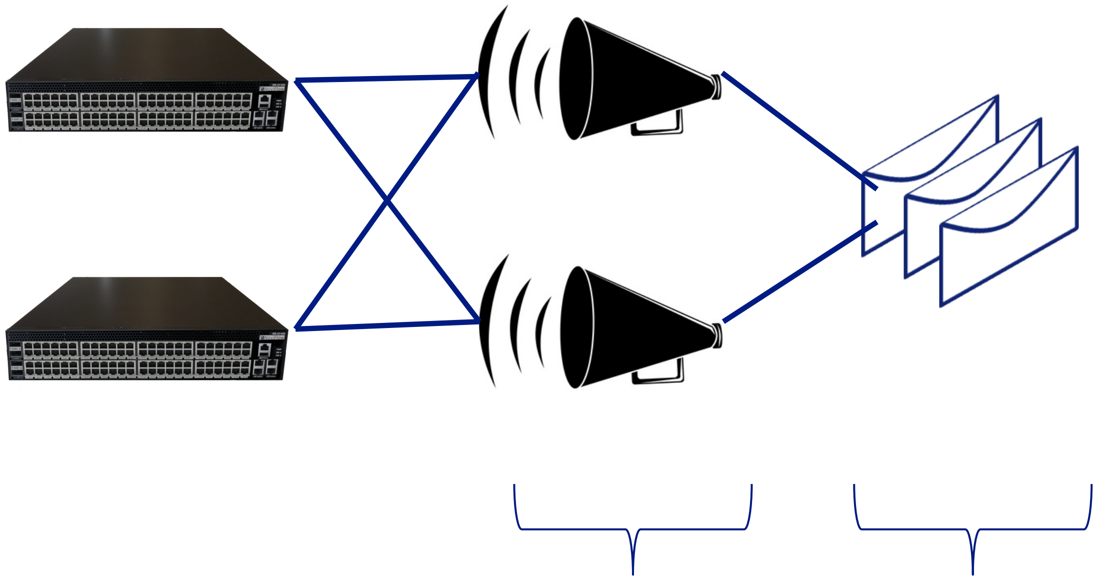
TELSTRA

Our Solution
Regionalized OpenFlow Speakers



Our Solution

Message Queue As ESP Bus

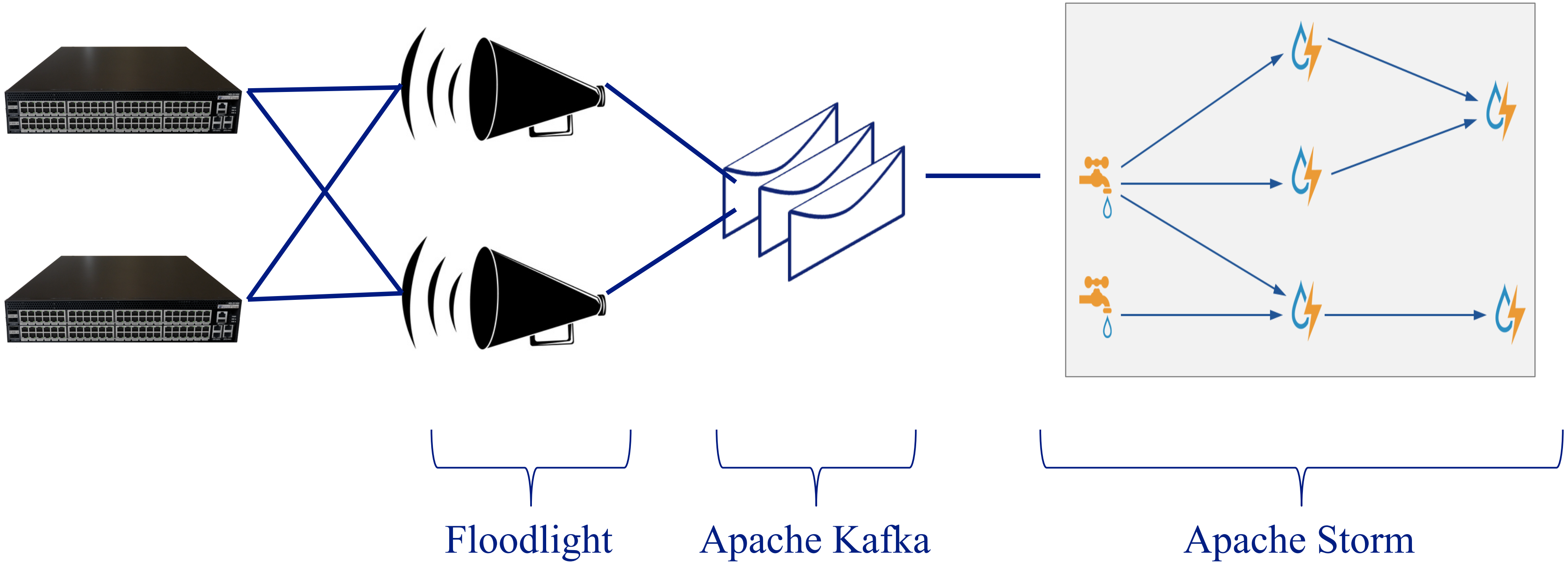


Floodlight

Apache Kafka

Our Solution

Realtime Stream Processing via Storm

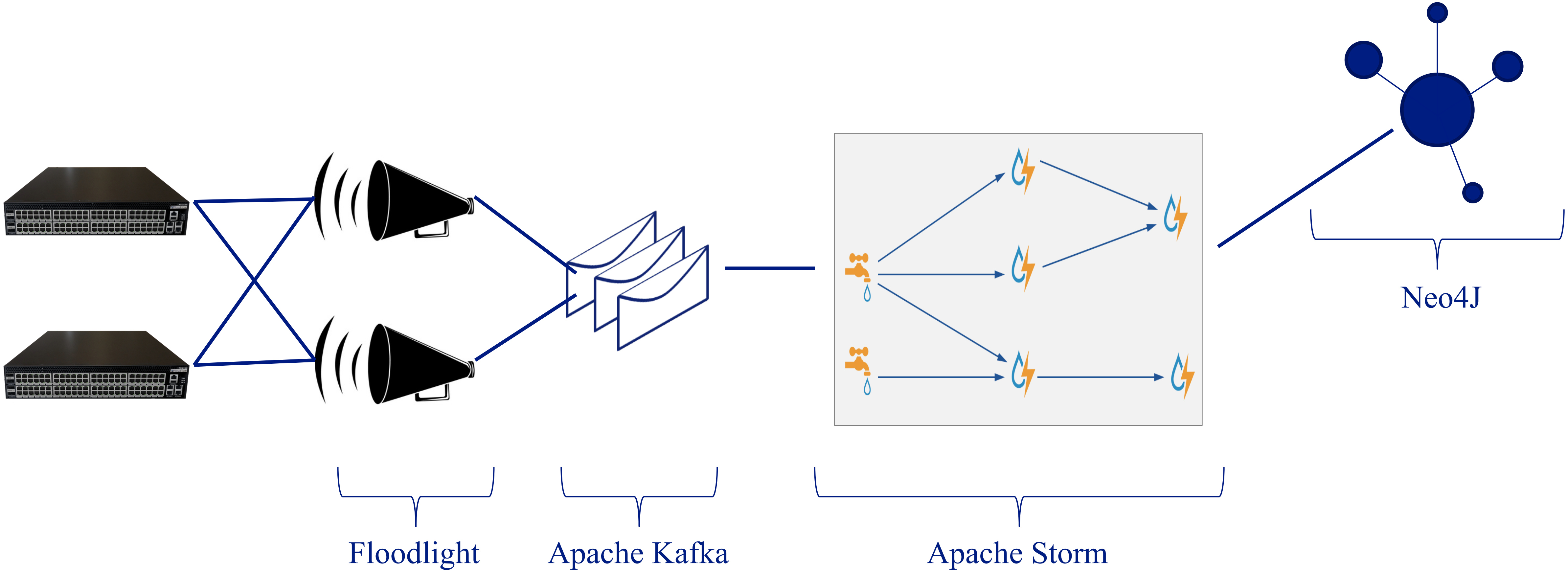


Floodlight

Apache Kafka

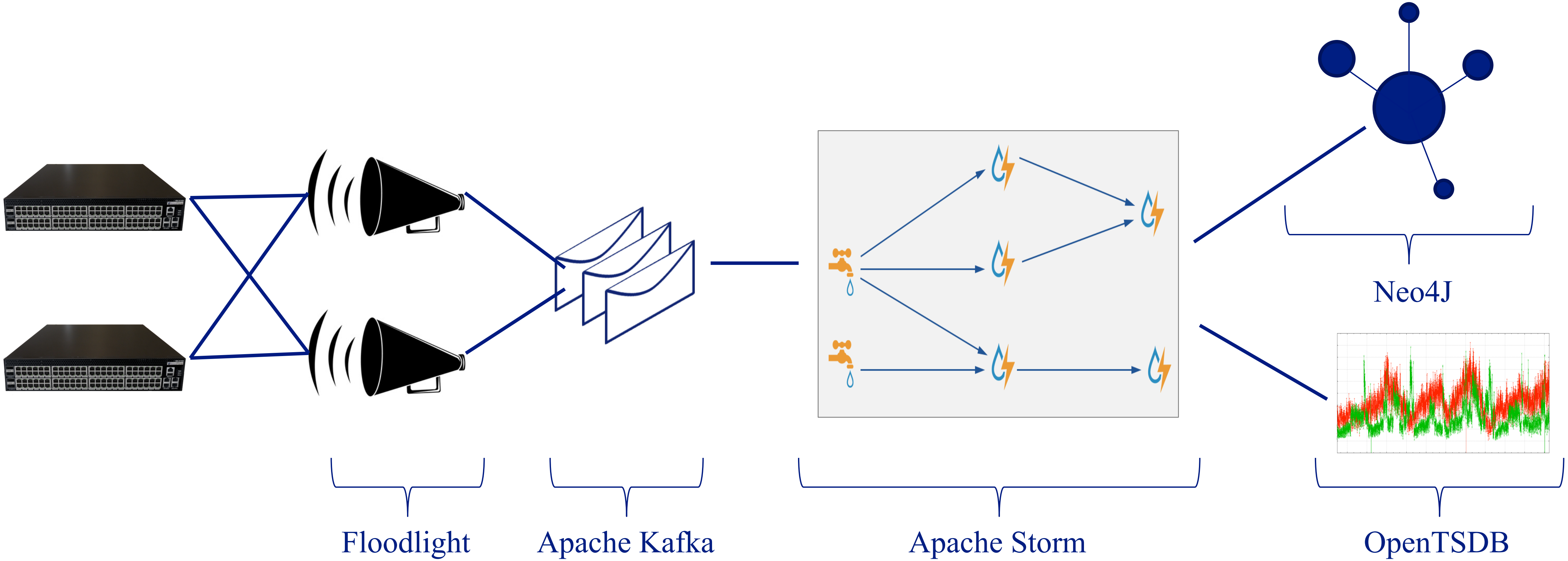
Apache Storm

Our Solution
GraphDB Based On Neo4j



TELSTRA

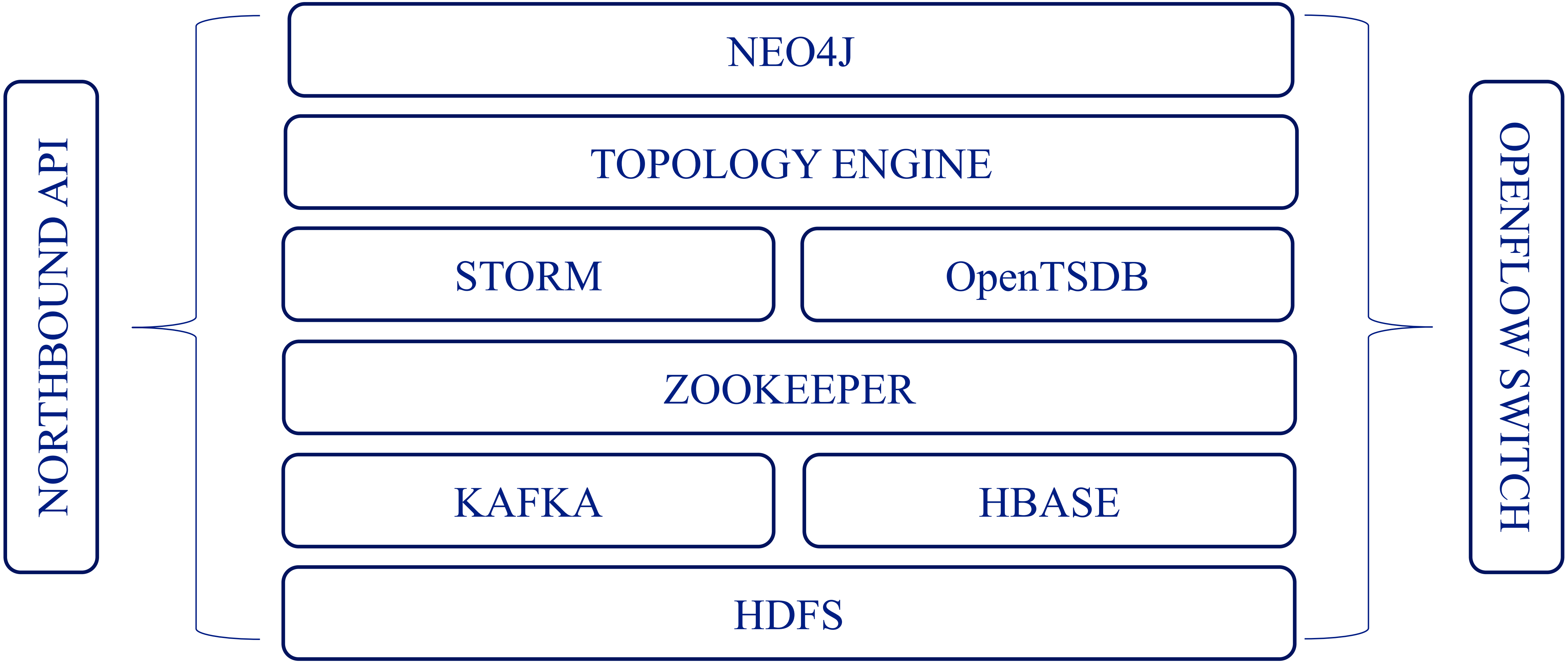
Our Solution
OpenTSDB and HBase



TELSTRA

Our Solution

Architecture



NORTHBOUND API

NEO4J

TOPOLOGY ENGINE

STORM

OpenTSDB

ZOOKEEPER

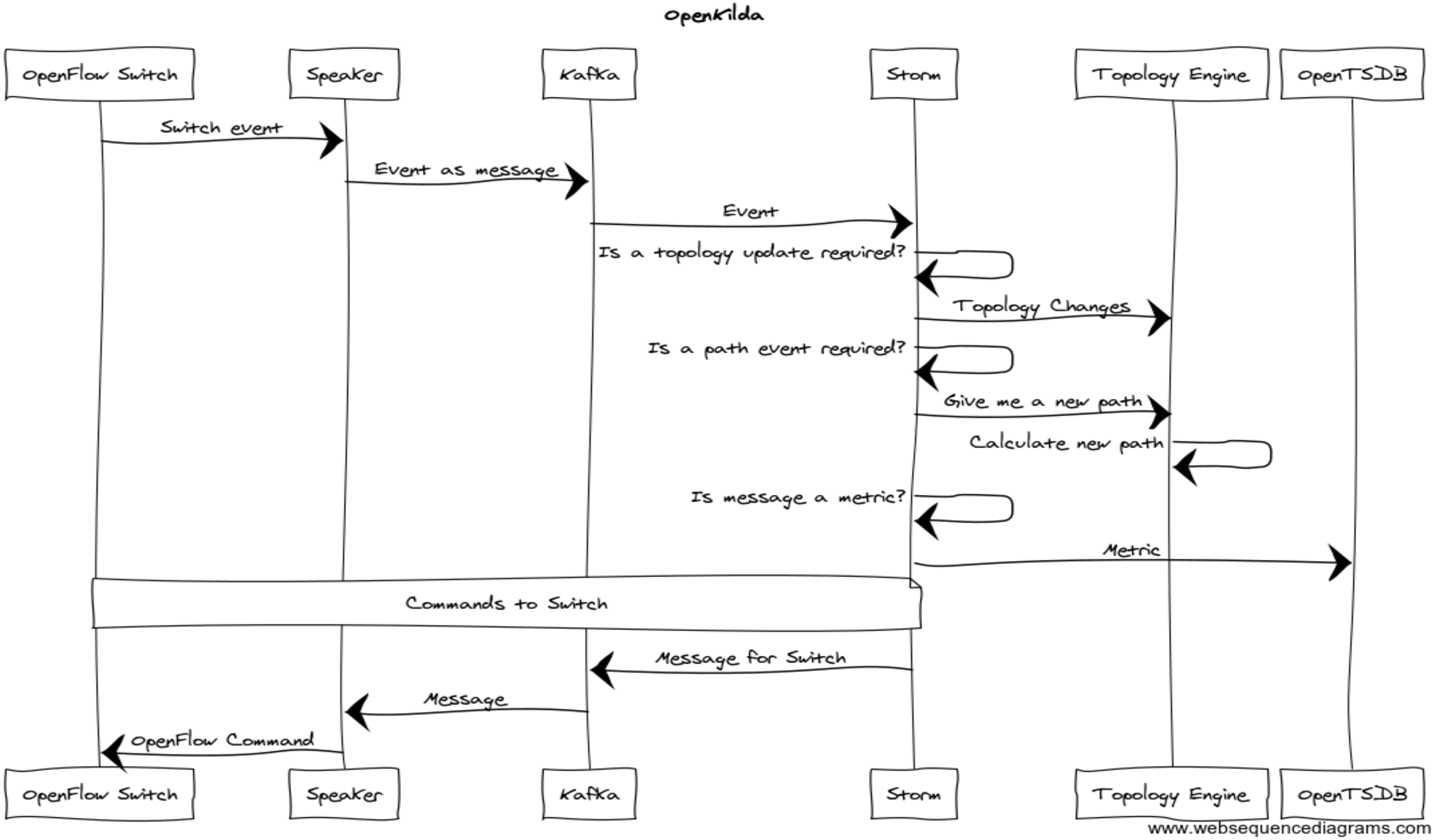
KAFKA

HBASE

HDFS

OPENFLOW SWITCH

Sequence Diagram



TELSTRA

Current State

API

Northbound Interface

- Restful
- Create/Modify/Delete Flow
- Push/Pop/Modify VLANs
- List Flows/Switches

Telemetry

- Flow stats
- Port stats
- Switch status

Operational

- Auto-discover network
- Active monitor of ISL with Latency
- Re-Flow when topology change occurs

How’d We Do?
Based On The Original Objectives

Sub-Second Failover – **NOT YET**

~~Negative Affinity In Path Selection~~

~~Active Latency Measurement on ISL~~

~~End-to-End Latency Measurement on Flow~~

~~Path Selection Based on Latency~~

~~Auto-re-route based on real-time latency/packet loss/jitter measurements~~

Multiple data points for comprehensive end-to-end network state – HALF DONE

Horizontal scale

~~Number of switches~~ - 10K Switches

~~Number of flows~~ – 16M Flows

Complex match/actions using experimenters – **NOT YET**

~~Stats collections at 1 second intervals~~

~~Self Healing/Optimizing Network~~

~~Zero Touch Controller Deployment/Upgrade~~

Whats Next

Features

- GUI
- Consolidated Northbound API
- Lightweight Speaker
- Documentation

Functionality

- Extend topology event logic
- Complex Match/Action
- BFD for ISL status
- Fast re-route
- Pre-emptive re-route

Build

- Shorten build time
- Extend build pipeline
- Test in sandbox

Get Involved
It’s OpenSource

Homepage: `https://github.com/telstra/open-kilda`

```
git clone https://github.com/telstra/open-kilda.git
```

Native Development Environment

```
# clone your GitHub fork
make build-latest
docker-compose up
```

Linux Based Environment

```
vagrant up
vagrant ssh
ssh-keygen -t rsa –C your_email@example.com
# update your GitHub fork with ssh key
# clone your GitHub fork
make build-latest
docker-compose up
```

Thank you

TELSTRA