Network Automation @ LinkedIn

Naufal Jamal

Staff Network Engineer @ LinkedIn

Agenda

- Some common network problem statements.
- Challenges in network maintenance and when it turns into network outages.
- NetSMART (Network Simplified Maintenance and Reporting Tool).
- StateDB. Network state information using REST API’s.
- Network Traffic Shift. Auto remediation of link failures in Data Centers.

Challenges in Network Maintenances

Problem Statement : A growing infrastructure is always changing. Maintenance is an inevitable part of our daily work.

How do I make sure that my maintenance doesn't cause any outage?

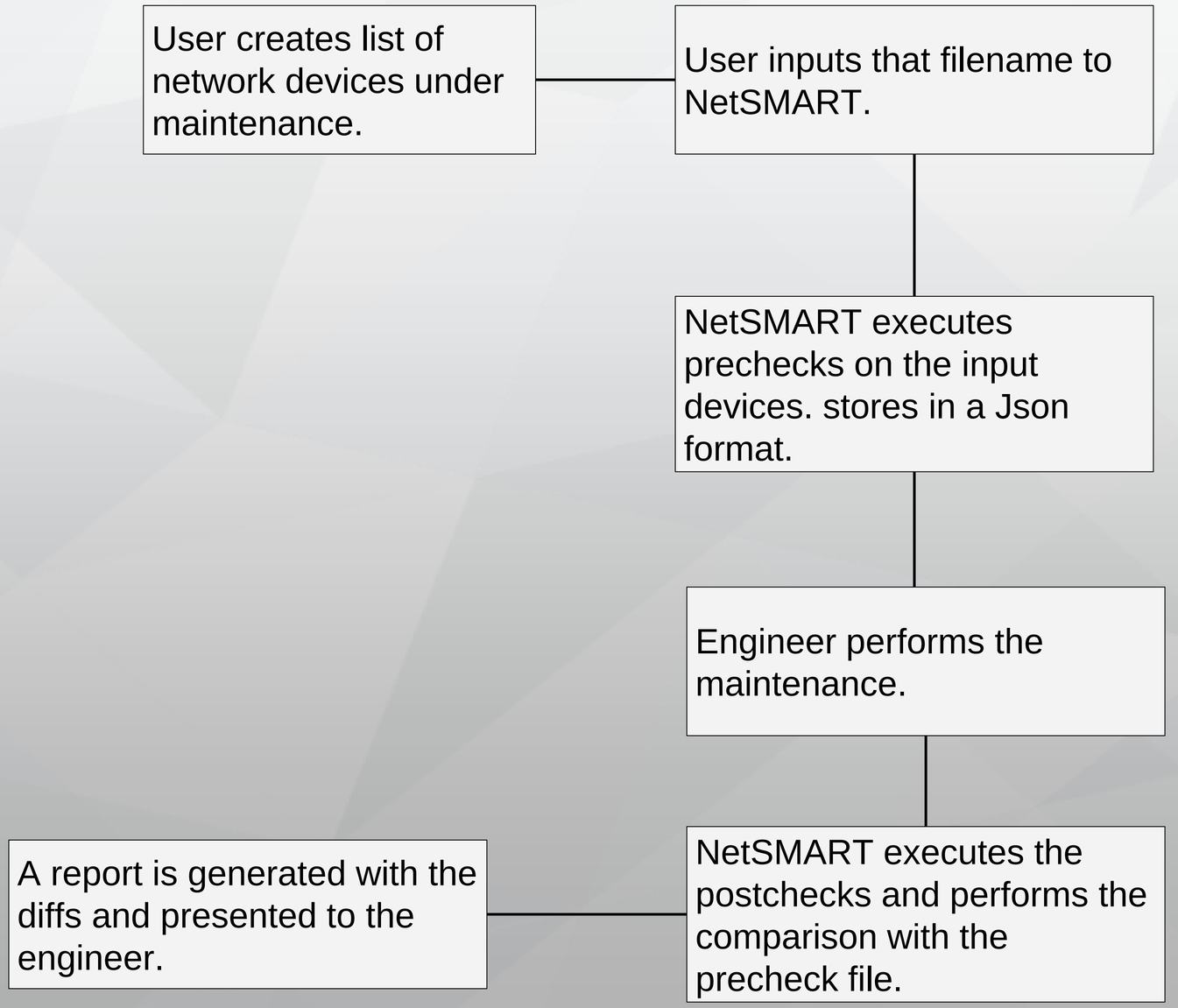
Sometimes, we may overlook certain things during the Maintenance. This can cause potential unforeseen outages.

Solution: So we need to have a tool in place that will show the state changes of the gears that were under maintenance. A tool which will give us a clear picture of network state change brought by a maintenance.

NetSMART (Network Simplified Maintenance and Reporting Tool)

An In-house tool to track the changes brought by any maintenance.

Simple workflow of NetSMART



List of commands for juniper platform

Platform	Command	Notes
Juniper	show bgp summary I match Estab I no-more	Captures established BGP peers
Juniper	show chassis routing-engine I match Idle	Capture CPU information
Juniper	show configuration I display set I except mpls I except bgp I except firewall I except policy I except secret I except password I except snmp I no-more	Captures config excluding sensitive info.
Juniper	show configuration I display set I match bgp I no-more	Captures BGP config
Juniper	show configuration I display set I match firewall I except count I no-more	Captures firewall config
Juniper	show configuration I display set I match mpls I no-more	Captures MPLS config
Juniper	show configuration I display set I match policy I except policy-options I no-more	Captures policy config
Juniper	show configuration I display set I match policy-options I no-more	Captures policy options config
Juniper	show isis adjacency I no-more	Captured ISIS adjacency
Juniper	show lldp neighbors I no-more	Captured LLDP neighbors
Juniper	show mpls lsp I match PRI I no-more	Captures LSP's which are primary
Juniper	show route 0.0.0.0 brief I match via I except label-switched-path I no-more	Captures default route information
Juniper	show route advertising-protocol bgp <peer> I except /32 I except Last I no-more	Captures route advertised to only e0/e1/crt
Juniper	show route receive-protocol bgp <peer> ' I except /32 I except Last I no-more	Captures route received only from e0/e1/crt
Juniper	show route receive-protocol bgp <transit ip> I match inet.0	Captures number of routes received from transit
Juniper	show rsvp neighbor I no-more	Captures RSVP neigbors
Juniper	show system memory I match Free	Captures free memory

cisco_functions.py

```python
def cisco_capture_bgp(device, username, password):
    result = {}
    connection = paramiko.SSHClient()
    connection.connect(device, port=22, username=username, password=password,
                       look_for_keys=False, timeout=None)
    output = connection.recv()
    result['device'] = output
    return result
```

juniper_functions.py

```python
def juniper_capture_bgp(device, username, password):
    result = {}
    connection = paramiko.SSHClient()
    connection.connect(device, port=22, username=username, password=password,
                       look_for_keys=False, timeout=None)
    output = connection.recv()
    result['device'] = output
    return result
```

Separate Modules for each platform which contains functions specific to that platform.

netsmart.py

```python
import juniper_functions
import cisco_functions

def functions(device, username, password):
    if platform = 'cisco':
        output = [cisco_functions.cisco_capture_bgp(args), ...]
    if platform = 'juniper':
        output = [juniper_functions.juniper_capture_bgp(args), ...]
    json_format[device] = output
    return json_format
```

```python
def run(device_list)
    pool_size = min(num_of_cpu, len(device_list))
    p = Pool(pool_size)
    result = p.map(functions, device_list)
    return result
    """result is a dictionary  of format {'device1': [output], 'device2': [output]}"""
```

Pool is used to process multiple devices at the same time.

```python
def json_compare(json1, json2):
    """Takes 2 json files as argument
        sorts each json file and compares the values for each key"""
    return diff
```

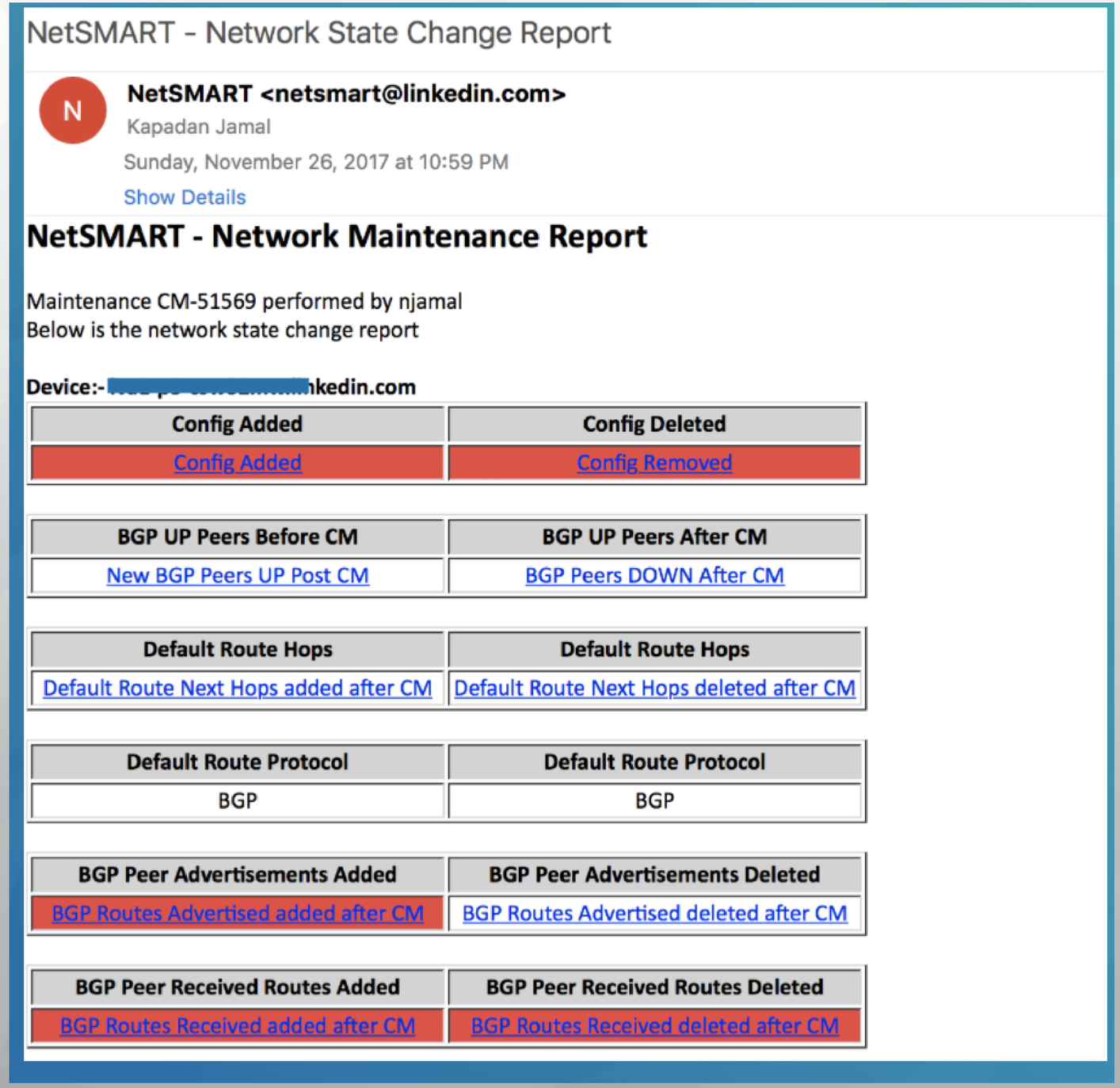
JSON compare compares 2 json files and return the diff.

Sample NetSMART output

```
[njamal@[redacted] ~]$ netsmart --snapshot maintenance_precheck.json --device_list netsmart_devices.txt --cm CM-50552


NetSMART: Network Simplified Maintenance and Reporting Tool

***Please Always use the FQDN for devices

Please visit [redacted] to see what parameters are covered

Questions? Reach out to Naufal Jamal njamal@linkedin.com


Below Gears will be Captured by NetSMART
--------------------------------
[redacted]nkedin.com
Capturing device config for [redacted]inkedin.com
Capturing device MPLS config for [redacted]inkedin.com
Capturing device BGP config for [redacted]nkedin.com
Capturing BGP peers information for [redacted]inkedin.com
```

<<Snipped>>

```
---Generating Devices Snapshot---

Attaching the report to the CM
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1118k    0   964  100 1117k    236   274k  0:00:04  0:00:04 --:--:--  299k
NetSMART report attached to CM-50552
```



NetSMART - Network State Change Report

N

NetSMART <netsmart@linkedin.com>

Kapadan Jamal

Sunday, November 26, 2017 at 10:59 PM

Show Details

NetSMART - Network Maintenance Report

Maintenance CM-51569 performed by njamal
Below is the network state change report

Device:- [redacted]nkedin.com

Config Added	Config Deleted
Config Added	Config Removed

BGP UP Peers Before CM	BGP UP Peers After CM
New BGP Peers UP Post CM	BGP Peers DOWN After CM

Default Route Hops	Default Route Hops
Default Route Next Hops added after CM	Default Route Next Hops deleted after CM

Default Route Protocol	Default Route Protocol
BGP	BGP

BGP Peer Advertisements Added	BGP Peer Advertisements Deleted
BGP Routes Advertised added after CM	BGP Routes Advertised deleted after CM

BGP Peer Received Routes Added	BGP Peer Received Routes Deleted
BGP Routes Received added after CM	BGP Routes Received deleted after CM

At the end of maintenance, a detailed report is generated and sent over email.

Red marked fields show the changes that happened due to the maintenance.

Helps Engineers to see any unexpected changes and take proactive steps.

StateDB

Problem Statement : With a growing number of network automation engineers, how do we avoid multiple scripts logging into the devices to extract the same information?

Imagine 10 different scripts logging into the devices at the same time. If the scripts don’t cleanly terminate the ssh connection, it may prevent legit users from accessing the devices.

Sometimes common operations like, fetching BGP/Link info etc. are repeated in many scripts resulting in duplication of code.

Solution: We need to store network state information of the devices in a central database and expose that information using REST API’s so that any application which wants to consume that data can do so without having to login to devices

Benefits:

- Enable automation engineers to write lightweight applications because most of the common network operations code logic is offloaded to stateDB.
- Smaller number of SSH sessions to the devices. No need to login to devices for getting network information everytime.
- Vendor agnostic. Automation engineers don’t need to write separate logic for different platforms. E.g Engineer says get me the BGP neighbors in device ‘X’. StateDB determines the platform and returns the data in a standard schema (json)

Example:

https://url/api/v1/device=xyz@linkedin.com

Would return data in JSON structure like below

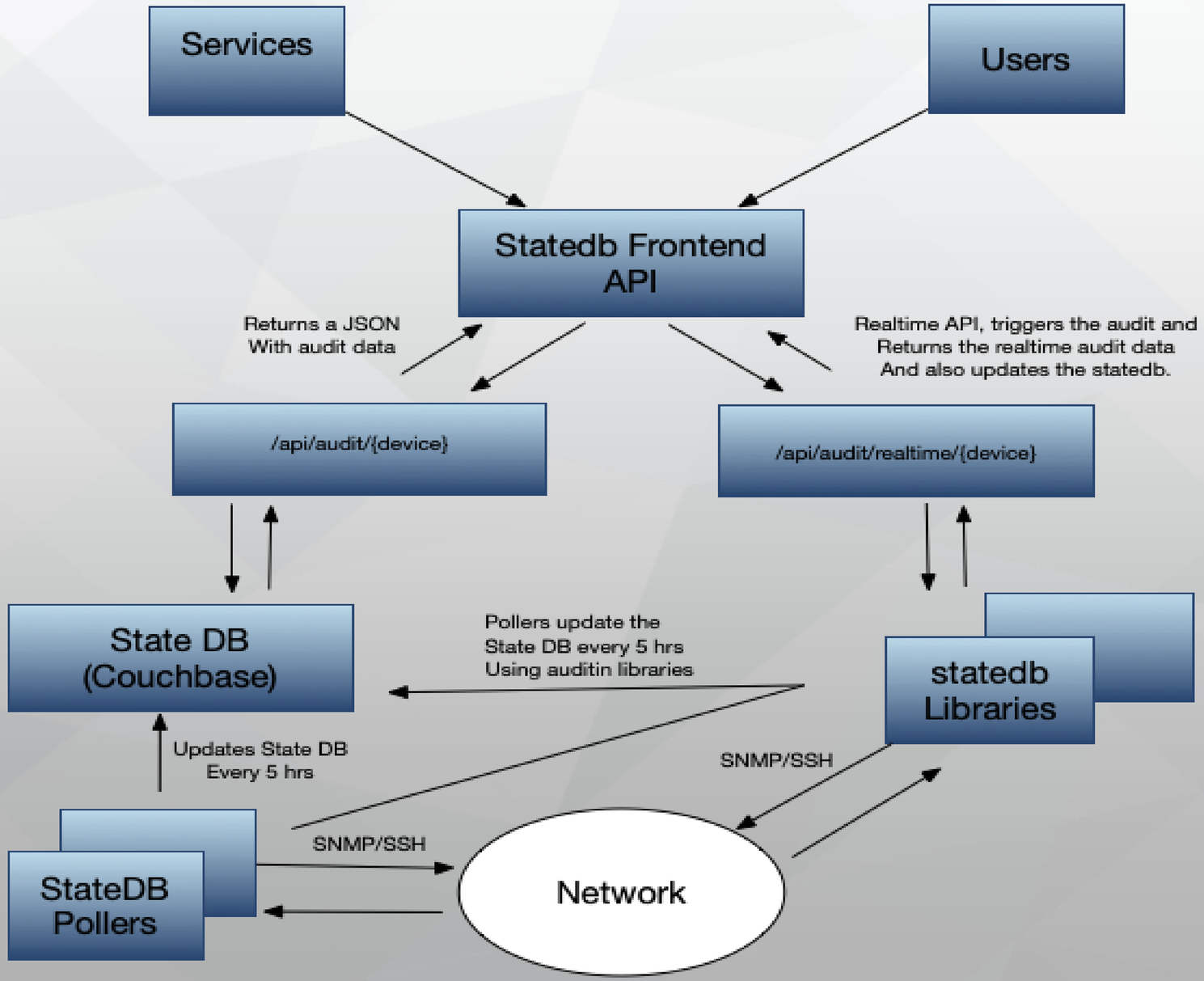

```
{
"xyz@.linkedin.com": [
{
"bgp":
{
"bgp_total_peer": 20,
"bgp_router_id": "12345",
"bgp_v4_peer": ["1.1.1.1", "2.2.2.2", "3.3.3.3"],
"bgp_v6_peer": ["a:a:a:a", "b:b:b:b", "c:c:c:c"],
<<Snipped>>
},
"bfd":
{
"bfd_v4_peer": ["1.1.1.1", "2.2.2.2", "3.3.3.3"],
"bfd_v6_peer": ["a:a:a:a", "b:b:b:b", "c:c:c:c"],
"bfd_interface": ["eth1/1", "eth11/2", "eth11/3"]
},
"lldp":
{
"lldp_interface": ["eth1/1", "eth11/2", "eth11/3"],
"lldp_peer": ["abc.linkedin.com", "xyz.linkedin.com"]
},
"interface":
{
"interfaces_status": ["eth1/1", "eth11/2", "eth11/3"],
"interfaces_description": [{"eth1/1": "test"}, {"eth11/2": "test"}]
}
]
}
```

REST API Endpoints:

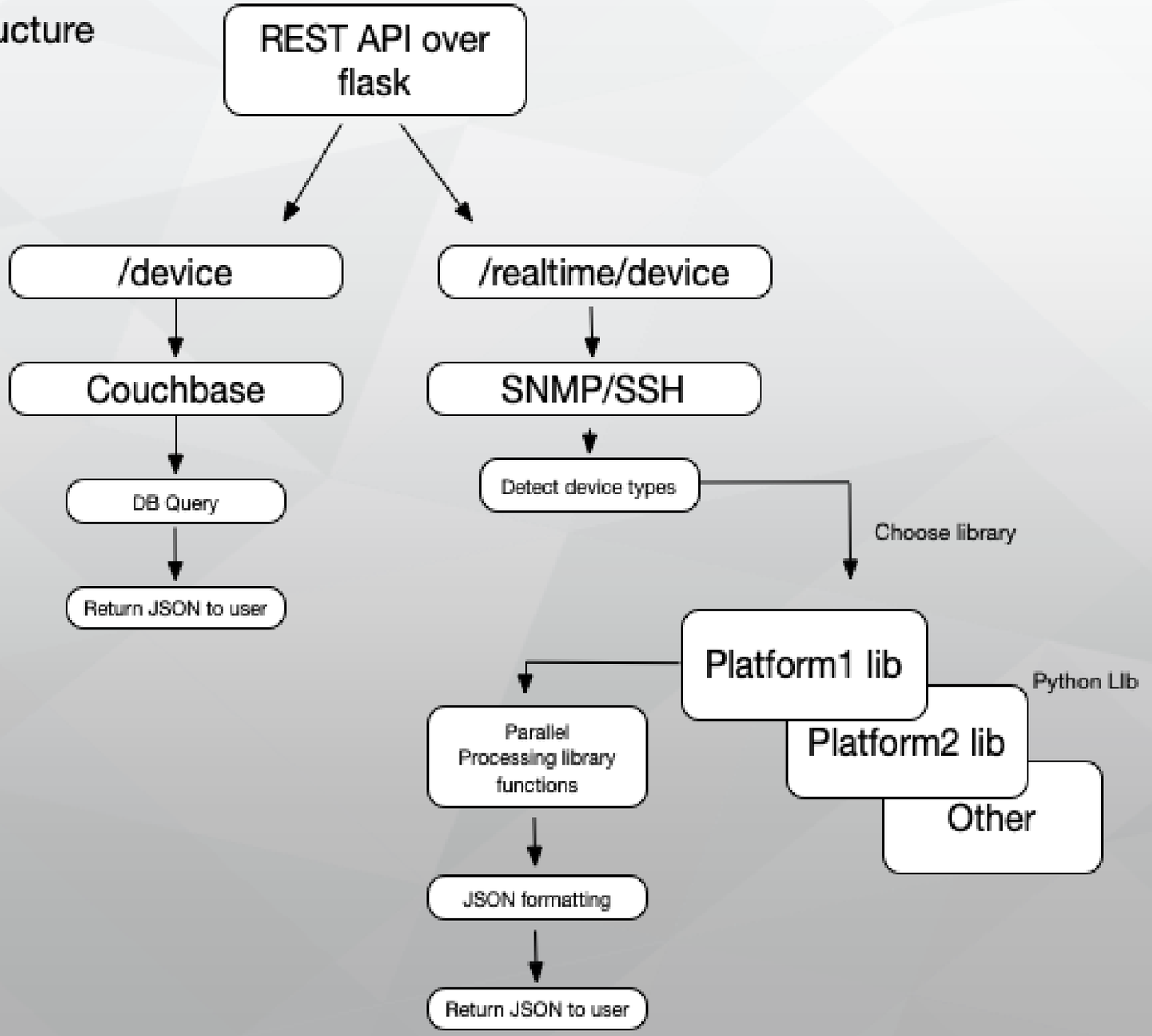
- /device - When used, it queries the stateDB and returns the results in JSON format. Used when the information needed is very less often changed in the network. E.g Code versions, NTP servers etc.

- /realtime/device - When used, it queries the device directly and returns the result in JSON format. Used when the information requested for is time sensitive. E.g BGP status, Links status.

High Level Design



Code structure



Technologies to be used

Python : All StateDB libraries for all platforms written in python.

Flask : All REST API endpoints to be written in flask.

Couchbase : Nosql data store for the statedb.

SNMP/SSH : Mechanisms to be used to pull data from statedb.

Multiprocessing : Pool feature used to spawn parallel threads for faster execution.

Network Traffic Shift

Problem Statement : Whenever links inside DC go problematic, there is a lot of back and forth between the network engineers and the DC techs to isolate and repair faulty links. DC team has to rely a lot on the Network team to repair any link. Network team has to manually monitor for link issues and drain the traffic manually.

Network engineers should not come to know when a link goes bad.
DC team should not rely on the network engineers to repair any link.

Current Workflow

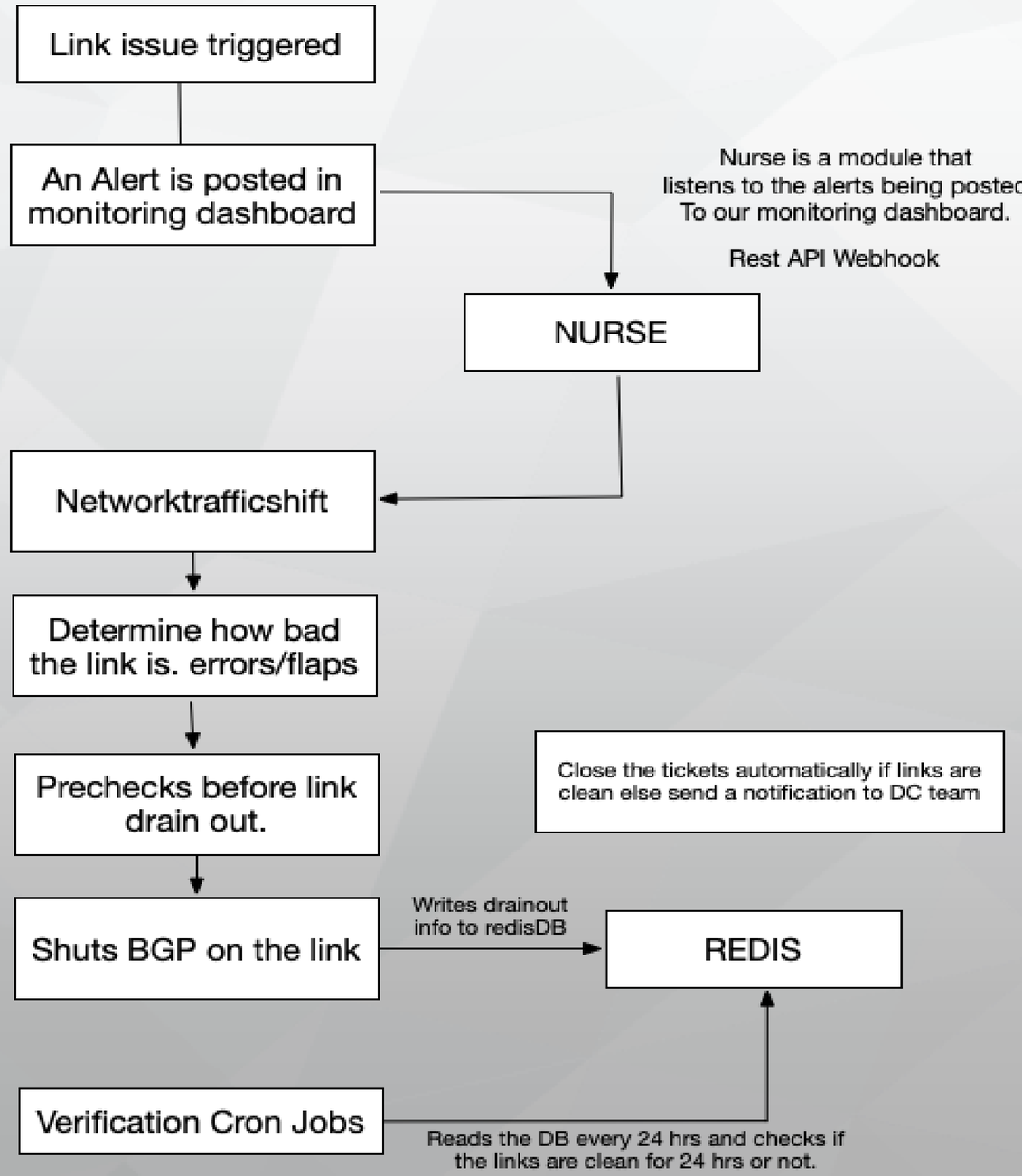
1. Link flap event occurs in DC. It can be either a link flaps or link errors.
2. An alert is generated.
3. Network engineer claims the alert and drains the traffic from the link by shutting BGP.
4. Network engineer creates a ticket for DC team for cabling repair.
5. DC team reaches out to network engineer to drain the traffic fully on the impacted ports in order to repair the cables.
6. DC team replaces the cables and optics.
7. network engineer then unshut the BGP on the links.
8. network engineer then has to manually monitor the link before closing the issue.

IN Short, A LOT OF BACK and FORTH between Network and DC Teams!!

Goals:

- We need to have a tool in place which should monitor for faulty links in the system
- Drains the traffic out of the link automatically if faulty.
- Reduce interactions between Network and DC teams on troubleshooting link issues.
- A tool intelligent enough to determine when to take a link in/out of service without causing any disruptions in the network.
- We need some reporting features to tell us stats like, optic types which flaps the most etc.
- Network engineers should not even come to know that a link has gone bad and should be auto-remediated.

Flowchart


Link issue triggered
An Alert is posted in monitoring dashboard
Nurse is a module that listens to the alerts being posted To our monitoring dashboard.
Rest API Webhook
NURSE
Networktrafficshift
Determine how bad the link is. errors/flaps
Prechecks before link drain out.
Close the tickets automatically if links are clean else send a notification to DC team
Shuts BGP on the link
Writes drainout info to redisDB
REDIS
Verification Cron Jobs
Reads the DB every 24 hrs and checks if the links are clean for 24 hrs or not.

Workflow (When the script is run manually)

python networktrafficshift.py --device xyz.linkedin.com --port 0/217

INFO:[trafficshift]:Fetching Flap count for xyz.linkedin.com
INFO:[trafficshift]:Alerts Supressed!
INFO:[trafficshift]:Fetching errors data for the last 24 hrs for xyz.linkedin.com
INFO:[trafficshift]:Fetching ports for xyz.linkedin.com
INFO:[trafficshift]:Fetching BGP V4/V6 Peers to be shut in xyz.linkedin.com
INFO:[trafficshift]:Checking ECMP for xyz.linkedin.com

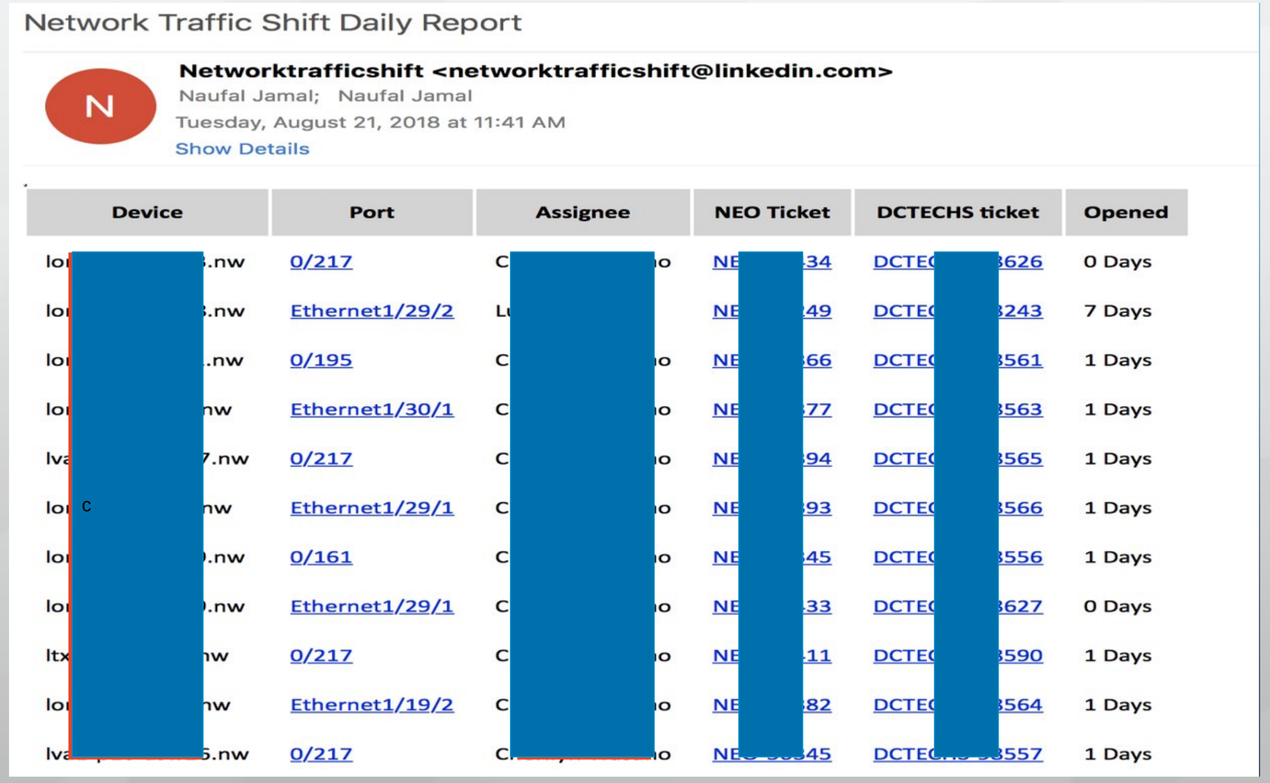
==Traffic Failout Summary==

Device : xyz.linkedin.com
Port : 0/217
Flap Count : 20
Input Errors : 0.4
Output Errors : 0.0

INFO:[trafficshift]:Config Push in Progress for xyz.linkedin.com
INFO:[trafficshift]:DCTECHS ticket DCTECHS-xxxxx created
INFO:[trafficshift]:Validating failout for xyz.linkedin.com
Link failed out for xyz.linkedin.com 0/217

How do we verify if the links are good or not?

- Every information (device, port, ticket) for a link drainout is stored a redisDB instance.
- A Cron job reads the redisDB every 24 hrs.
- For every entry in the DB, the script does the below checks:
  - If the port is UP
  - If there are any errors on the link
  - If there are any flaps on the link
- If the Link is clean, then the tickets are closed automatically.
- If the Link is not clean, then an automated notification is sent to the Data Center team.
- Network engineers don't even come to know a link going bad and being fixed :)



Network Traffic Shift Daily Report

Networktrafficshift <networktrafficshift@linkedin.com>
Naufal Jamal; Naufal Jamal
Tuesday, August 21, 2018 at 11:41 AM
Show Details

Device	Port	Assignee	NEO Ticket	DCTECHS ticket	Opened
lo[redacted].nw	0/217	C[redacted]o	NE[redacted]34	DCTE[redacted]3626	0 Days
lo[redacted].nw	Ethernet1/29/2	L[redacted]	NE[redacted]49	DCTE[redacted]3243	7 Days
lo[redacted].nw	0/195	C[redacted]o	NE[redacted]66	DCTE[redacted]3561	1 Days
lo[redacted]nw	Ethernet1/30/1	C[redacted]o	NE[redacted]77	DCTE[redacted]3563	1 Days
lva[redacted]7.nw	0/217	C[redacted]o	NE[redacted]94	DCTE[redacted]3565	1 Days
lo[redacted]nw	Ethernet1/29/1	C[redacted]o	NE[redacted]93	DCTE[redacted]3566	1 Days
lo[redacted].nw	0/161	C[redacted]o	NE[redacted]45	DCTE[redacted]3556	1 Days
lo[redacted].nw	Ethernet1/29/1	C[redacted]o	NE[redacted]33	DCTE[redacted]3627	0 Days
ltx[redacted]nw	0/217	C[redacted]o	NE[redacted]11	DCTE[redacted]3590	1 Days
lo[redacted]nw	Ethernet1/19/2	C[redacted]o	NE[redacted]82	DCTE[redacted]3564	1 Days
lva[redacted]6.nw	0/217	C[redacted]o	NE[redacted]45	DCTECHS-[redacted]3557	1 Days

Reporting and tracking

- A report like shown is sent every 24 hrs to respective teams to track each ticket
- Shows if any ticket is breaching the SLA or not
- Shows data like type of optics that errors the most, DC’s with most number of issues etc.

That’s IT!
Questions/Feedback?