BGP Analytics with OpenConfig Telemetry and gRPC

DENOG11 – 11 and 12 November 2019

Peter Sievers

Agenda

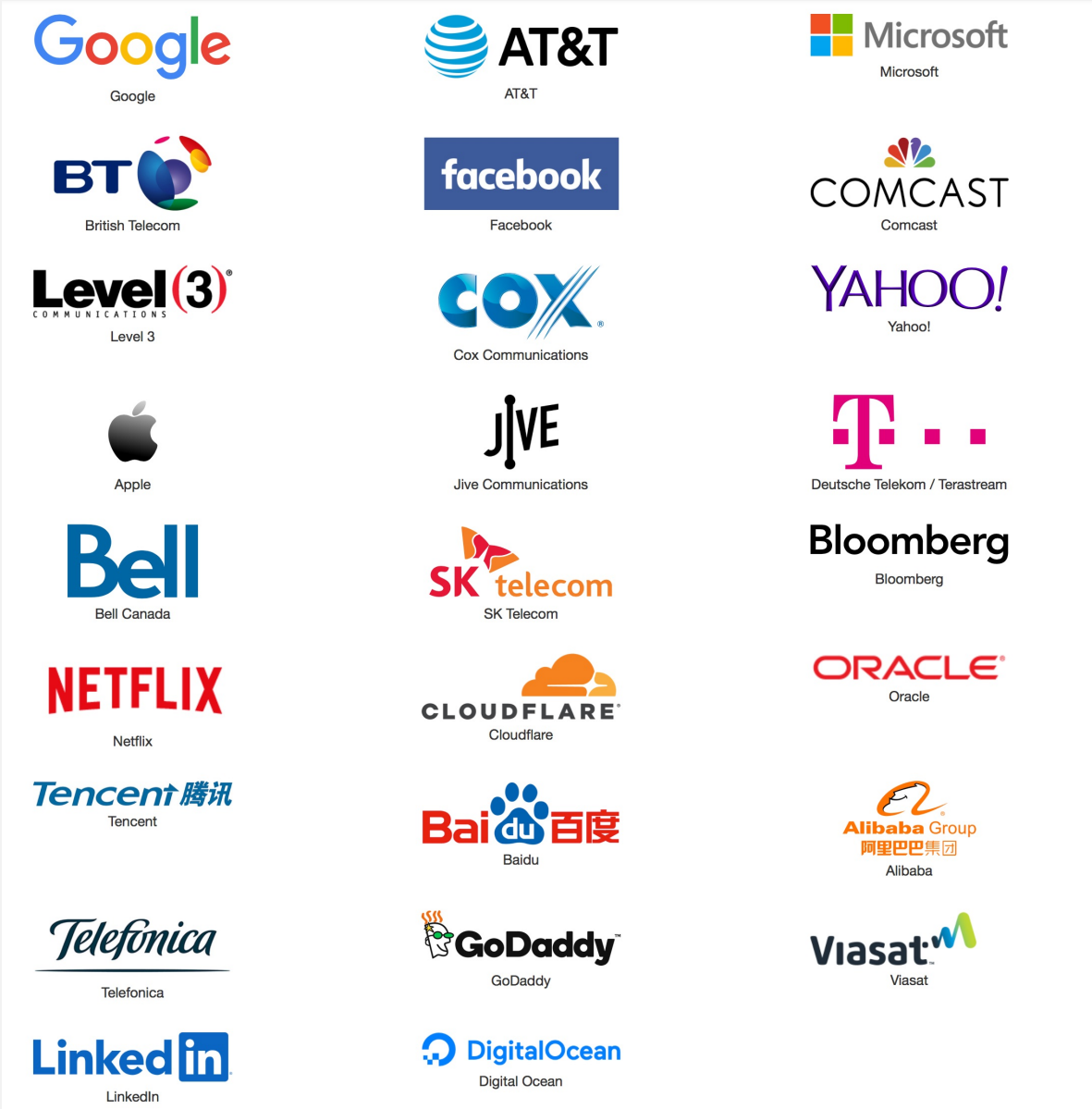
- OpenConfig
- gRPC & protobuf
- data models / structured data – Key to Automation
- TICK Stack / user-defined functions
- Anomalies and Outliers within received routes
- Resources

OpenConfig Overview

OpenConfig overview

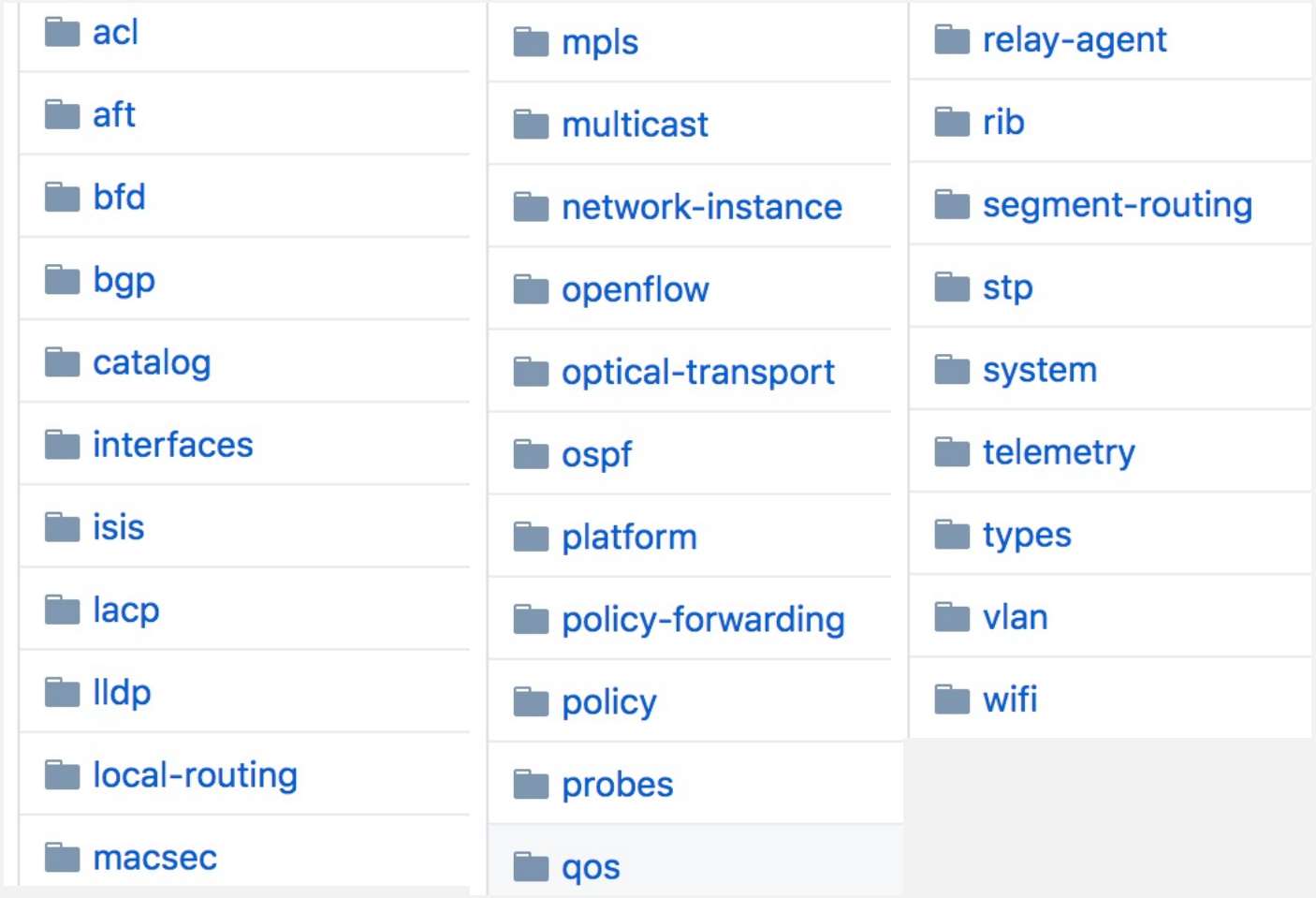
- collaboration of **network operators**
- **Vendor-neutral** data models written in **YANG**
- configuration and operational states
- develop **programmatic** interfaces and tools for managing networks **dynamically**
- based on actual operational **needs**
- data models are published on github:

  https://github.com/openconfig/public



OpenConfig overview

- Published versions of OpenConfig modules can be found in the release/models directory



- **bgp** – covering configuration and state
- **interfaces** – provides configuration and state for physical and logical device interfaces and ip addressing
- **local-routing** – static and aggregate routes
- **mpls** – configuration and operational state to MPLS/TE, including RSVP, LDP and SR
- **optical-transport** – configuration and state with client and line-side parameters
- **policy** – routing policies
- **rib** – BGP-4 RIB contents
- **telemetry** – state and configuration parameters
- **vlan** – parameters to 802.1Q
- **network-instance** – Layer 2 or Layer 3 forwarding
- **rpc** – set of operations between NE and a device supporting OpenConfig models
- …

One protocol, many syntaxes

```
# router bgp as-number 1234
```

```
# router bgp 1234
```

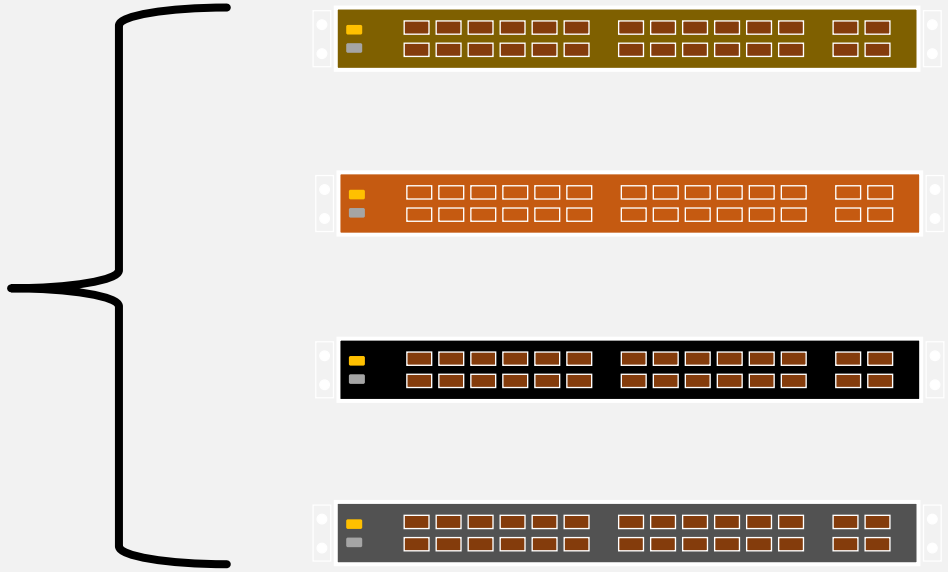
```
# configure router autonomous-system 1234
```

```
# set routing-options autonomous-system 1234
```

- each vendor has different configuration and command syntax
- NMS needs to understand and use device specific syntax for every device

One protocol, one syntax

```
#openconfig-bgp:bgp global as 12345
```



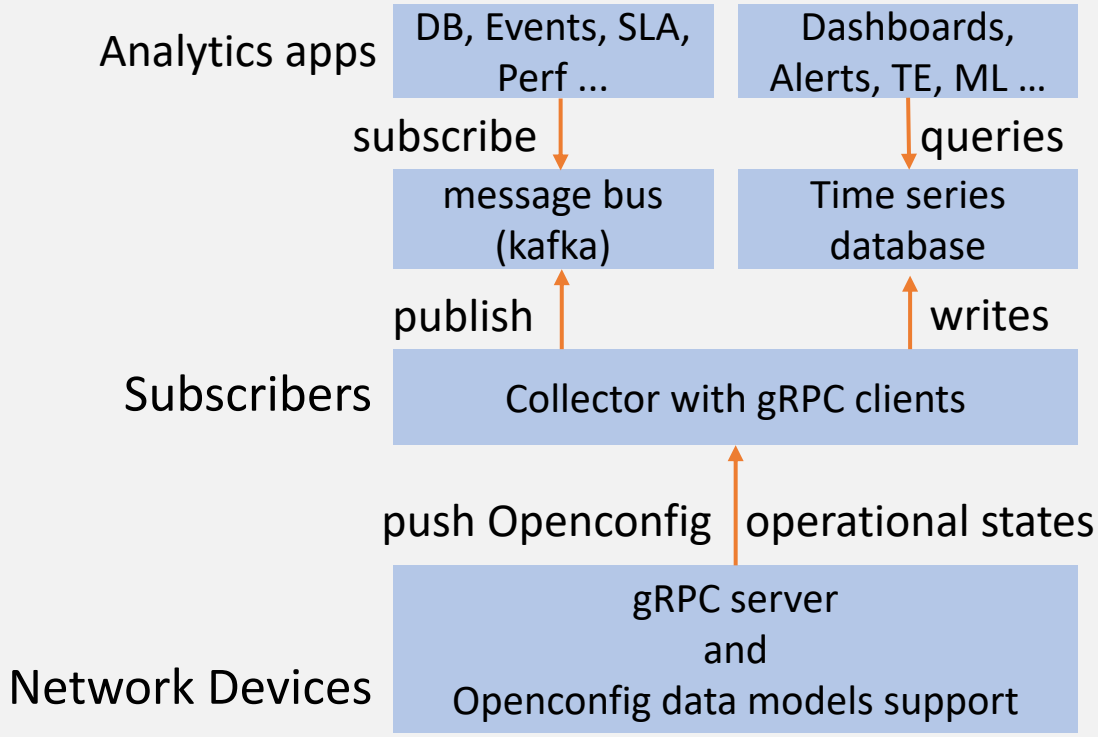
Example on Junos – BGP as number

```
psievers@vmx# set openconfig-bgp:bgp global config as 12345
psievers@vmx# set openconfig-bgp:bgp global config router-id 172.24.99.220
psievers@vmx# show openconfig-bgp:bgp
global {
    config {
        as 12345;
        router-id 172.24.99.220;
    }
}
```

```
psievers@vmx# run show configuration |display translation-scripts translated-config
routing-options {
    router-id 172.24.99.220;
    autonomous-system 12345;
}
```

OpenConfig and streaming Telemetry

- **push model –** network device streams the information to the requester
- **distributed** approach, every line card or every software module is responsible to stream its own data
- **high frequency** streaming with less CPU cycles, push every few seconds
- **data model–driven telemetry** a modern approach to “replace” legacy monitoring infrastructure i.e. CLI scraping, NETCONF, SNMP, …
- subscribed set of **YANG objects** are streamed to subscribers



OpenConfig and streaming Telemetry

```
[psievers@jtimon]$ grep path vmx3.json
    "paths": [
            "path": "/network-instances/network-instance[instance-name='master']\
                     /protocols/protocol/bgp/peer-groups/peer-group[peer-group-name='upstream1']/"
```

```
...
   {
     "key": "state/peer-as",
     "Value": {
       "UintValue": 64512
     }
    },
    {
     "key": "state/local-as",
     "Value": {
       "UintValue": 65000
     }
    },
    {
     "key": "state/peer-type",
     "Value": {
       "StrValue": "EXTERNAL"
     }
    },
...
```

- telemetry can also export information about the applied **configuration**
- telemetry helps stream data out of the network **faster**
- **large** amounts of data close to real time
- replaces periodic polling of network elements

OpenConfig and streaming Telemetry


```
    {
      "key": "state/prefixes/received",
      "Value": {
        "UintValue": 774137
      }
    },
    {
      "key": "state/prefixes/sent",
      "Value": {
        "UintValue": 0
      }
    },
    {
      "key": "state/prefixes/installed",
      "Value": {
        "UintValue": 774136
      }
    },
    {
      "key": "state/prefixes/accepted",
      "Value": {
        "UintValue": 774137
      }
    },
```

```
    {
      "key": "state/prefix-limit-exceeded",
      "Value": {
        "BoolValue": false
      }
    },
    {
      "key": "state/queues/input",
      "Value": {
        "UintValue": 0
      }
    },
    {
      "key": "state/queues/output",
      "Value": {
        "UintValue": 0
      }
    },
    {
      "key": "add-paths/state/receive",
      "Value": {
        "BoolValue": false
      }
    },
```

gRPC and protobuf

gRPC overview

- originally developed and implemented internally at Google
  - Codename Stubby RPC
  - Part of Cloud Native Computing Foundation
- SSL/TLS integration
- architecture of gRPC is layered
  - gRPC uses HTTP/2 as transport protocol
  - next layer is the channel, defines calling conventions and implements the mapping of an RPC onto the underlying transport
  - last layer is the stub, which defines interface constraints, data types and message encoding
- another key component of gRPC is a technology called protocol buffers – not required but handy
- protobufs are an IDL for describing services, methods and messages
- channel and the transport layer are IDL-agnostic

https://github.com/grpc/grpc/blob/master/doc/g_stands_for.md

'g' stands for something different every gRPC release:

- 1.0 'g' stands for 'gRPC'
- 1.1 'g' stands for 'good'
- 1.2 'g' stands for 'green'
- 1.3 'g' stands for 'gentle'
- 1.4 'g' stands for 'gregarious'
- 1.6 'g' stands for 'garcia'
- 1.7 'g' stands for 'gambit'
- 1.8 'g' stands for 'generous'
- 1.9 'g' stands for 'glossy'
- 1.10 'g' stands for 'glamorous'
- 1.11 'g' stands for 'gorgeous'
- 1.12 'g' stands for 'glorious'
- 1.13 'g' stands for 'gloriosa'
- 1.14 'g' stands for 'gladiolus'
- 1.15 'g' stands for 'glider'
- 1.16 'g' stands for 'gao'
- 1.17 'g' stands for 'gizmo'
- 1.18 'g' stands for 'goose'
- 1.19 'g' stands for 'gold'
- 1.20 'g' stands for 'godric'
- 1.21 'g' stands for 'gandalf'
- 1.22 'g' stands for 'gale'
- 1.23 'g' stands for 'gangnam'
- 1.24 'g' stands for 'ganges'
- 1.25 'g' stands for 'game'

protobuf

- originally developed and implemented internally at Google (early in 2001)
- Google Protocol Buffers or GPB or protobuf
- process for serializing data, minimize the data representation
- low serializer code footprint, suitable for NPUs
- de-serialization in one function call
- minimal size “on the wire” to support transmission of large data sets e.g. streaming telemetry

https://developers.google.com/protocol-buffers/

```
syntax = "proto3";

package telemetry;

// Interface exported by Agent
service OpenConfigTelemetry {
    // Request an inline subscription for data at the specified path.
    // The device should send telemetry data back on the same
    // connection as the subscription request.
    rpc telemetrySubscribe(SubscriptionRequest)
        returns (stream OpenConfigData) {}
...
```

```
...
import "telemetry_top.proto";

message network_instances_ni_bgp {
   message network_instance_list {
      optional string instance_name = 51 [(telemetry_options).is_key =
true];
      optional protocols_type protocols = 151;
      message protocols_type {
         optional protocol_type protocol = 151;
         message protocol_type {
...
 message state_type {
                     optional uint32 as = 51;
                     optional string router_id = 52;
                     optional uint32 total_paths = 81;
                     optional uint32 total_prefixes = 82;
                  }
```

data models – YANG

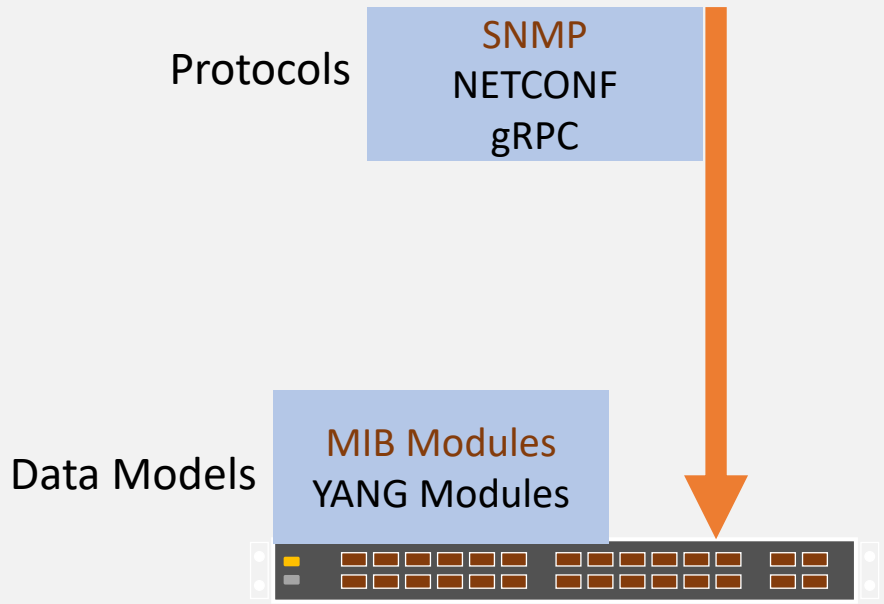
What are data-models?

A data-model explicitly and precisely determines the **structure**, **syntax** and **semantics** of the data

(how data is represented and accessed)

```
module: openconfig-bgp
  +--rw bgp
     +--rw neighbors
        +--rw neighbor* [neighbor-address]
           +--rw afi-safis
              +--rw afi-safi* [afi-safi-name]
                 +--rw afi-safi-name           -> ../config/afi-safi-name
                 +--rw config
                 |  +--rw afi-safi-name?   identityref
                 |  +--rw enabled?         boolean
                 +--ro state
                 |  +--ro afi-safi-name?   identityref
                 |  +--ro enabled?         boolean
                 |  +--ro active?          boolean
                 |  +--ro prefixes
                 |     +--ro received?              uint32
                 |     +--ro received-pre-policy?   uint32
                 |     +--ro sent?                  uint32
                 |     +--ro installed?             uint32
```

schema tree snippet from OpenConfig BGP data-model



Extensible –
no SNMP enterprise versus IETF MIBs

YANG overview

- **Y**et **A**nother **N**ext **G**eneration data modeling language. IETF netmod working group
- RFC 6020 YANG 1.0
- RFC 7950 YANG 1.1 (does not replace RFC 6020)
- RFC ...
- a data modeling language used to model both **configuration data** and **state data**
- Data model written in YANG are defined in **YANG modules**
- YANG modules can be categorized as
  - IETF defined YANG modules
  - OpenConfig defined YANG modules
  - Vendor specific YANG models
  - Customer defined YANG modules
- YANG modules can be translated into an equivalent XML syntax called YANG Independent Notation (YIN)

YANG – data definition statements

- **container** – interior data node in a schema tree, has no data value, a set of child nodes
- OpenConfig example for bgp snippet from – **openconfig-bgp.yang**

```
grouping bgp-top {
    description
      "Top-level grouping for the BGP model data";
    container bgp {
      description
        "Top-level configuration and state for the BGP router";
      container global {
        description
          "Global configuration for the BGP router";
      }
      container neighbors {
        description
          "Configuration for BGP neighbors";
      }
      container peer-groups {
        description
          "Configuration for BGP peer-groups";
      }
```

- grouping defines a reusable collection of nodes

```
$grep "bgp-top" */*.yang

bgp/openconfig-bgp.yang: uses bgp-top;

network-instance/openconfig-network-
instance.yang: uses oc-bgp:bgp-top {
```

YANG – data definition statements

- **Leaf** – is defined by an idenfier and has a value and no child nodes
- OpenConfig example for bgp snippet from – **openconfig-bgp-global.yang**

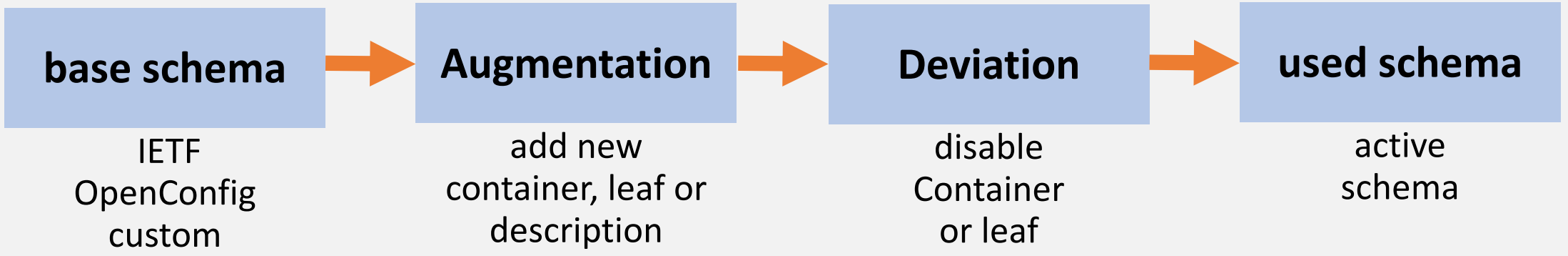
```
grouping bgp-global-config {
    description
      "Global configuration options for the BGP router.";
    leaf as {
      type oc-inet:as-number;
      mandatory true;
      description
        "Local autonomous system number of the router.  Uses the 32-bit as-number type from the model
         in RFC 6991.";
    }
    leaf router-id {
      type oc-yang:dotted-quad;
      description
        "Router id of the router - an unsigned 32-bit integer expressed in dotted quad notation.";
         reference
        "RFC4271 - A Border Gateway Protocol 4 (BGP-4), Section 4.2";
    }
  }
```

YANG – data definition statements

- **list** – interior data node in a schema tree, list node may have multiple instances (list entries are identified by keys that distinguish them from each other)
- OpenConfig example for bgp snippet from – **openconfig-bgp.yang**

```
grouping bgp-peer-group-afi-safi-list {
    description
      "List of address-families associated with the BGP peer-group";
    list afi-safi {
      key "afi-safi-name";
      description
        "AFI,SAFI configuration available for the
        neighbour or group";
      leaf afi-safi-name {
        type leafref {
          path "../config/afi-safi-name";
        }
        description
          "Reference to the AFI-SAFI name used as a key
          for the AFI-SAFI list";
      }
```

YANG Augmentation and Deviation



- add containers and leafs (even into existing containers)
- disable not supported containers and leafs
- maintained in separate text files
- naming of YANG modules must be unique within the same “world”

YANG tools – pyang

- validate YANG modules against YANG RFC
- convert YANG modules into equivalent YIN module (YANG to XML)
  - YIN is an equivalent XML syntax for YANG (pyang openconfig-bgp.yang -f yin ...)
- generate a tree representation of YANG models for quick visualization
- Tutorial: https://github.com/mbj4668/pyang/wiki/Tutorial

```
[psievers@yang_modules]$ pyang openconfig-bgp.yang -f tree --tree-
path=/bgp/neighbors/neighbor/afi-safis/afi-safi/state/prefixes/received
module: openconfig-bgp
  +--rw bgp
     +--rw neighbors
        +--rw neighbor* [neighbor-address]
           +--rw afi-safis
              +--rw afi-safi* [afi-safi-name]
                 +--ro state
                    +--ro prefixes
                       +--ro received?   uint32
```

YIN

```
$ pyang openconfig-bgp.yang -f yin --tree-path=/bgp/neighbors/neighbor/afi-safis/afi-safi/state/
                                                prefixes/received
<grouping name="bgp-top">
    <description>
      <text>Top-level grouping for the BGP model data</text>
    </description>
    <container name="bgp">
      <description>
        <text>Top-level configuration and state for the BGP router</text>
      </description>
      <container name="global">
        <description>
          <text>Global configuration for the BGP router</text>
        </description>
        <uses name="bgp-global-base"/>
      </container>
      <container name="neighbors">
        <description>
          <text>Configuration for BGP neighbors</text>
        </description>
        <uses name="bgp-neighbor-list"/>
      </container>
      <container name="peer-groups">
...
```

OpenConfig Telemetry with gRPC and GPB

No.	Time	Source	Destination	Protocol	Length	Info
16	2019-10-04 16:00:07,482090	172.24.2.102	172.24.2.101	HTTP2	75	SETTINGS[0]
17	2019-10-04 16:00:07,512944	172.24.2.102	172.24.2.101	GRPC	203	HEADERS[1]: 200 OK, DATA[1], HEADERS[1] (GRPC) (PROTOBUF)
18	2019-10-04 16:00:07,513016	172.24.2.101	172.24.2.102	TCP	66	46822 → 32768 [ACK] Seq=181 Ack=187 Win=30336 Len=0 TSval=267229551 TSecr=2416069711
19	2019-10-04 16:00:07,513346	172.24.2.101	172.24.2.102	GRPC	200	HEADERS[3]: POST /telemetry.OpenConfigTelemetry/telemetrySubscribe, DATA[3] (GRPC) (PROTOB…
20	2019-10-04 16:00:07,513494	172.24.2.102	172.24.2.101	HTTP2	81	SETTINGS[0]
21	2019-10-04 16:00:07,513549	172.24.2.101	172.24.2.102	HTTP2	75	SETTINGS[0]

```
▶ Frame 19: 200 bytes on wire (1600 bits), 200 bytes captured (1600 bits)
▶ Ethernet II, Src: Vmware_9b:e4:71 (00:50:56:9b:e4:71), Dst: Vmware_9b:c1:51 (00:50:56:9b:c1:51)
▶ Internet Protocol Version 4, Src: 172.24.2.101, Dst: 172.24.2.102
▶ Transmission Control Protocol, Src Port: 46822, Dst Port: 32768, Seq: 181, Ack: 187, Len: 134
▼ HyperText Transfer Protocol 2
  ▼ Stream: HEADERS, Stream ID: 3, Length 43, POST /telemetry.OpenConfigTelemetry/telemetrySubscribe
      Length: 43
      Type: HEADERS (1)
    ▶ Flags: 0x04
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0011 = Stream Identifier: 3
      [Pad Length: 0]
      Header Block Fragment: 838645a36125a0b495367a5f5565aaf1ea94d36f2d05a4a9...
      [Header Length: 222]
      [Header Count: 7]
    ▶ Header: :method: POST
    ▶ Header: :scheme: http
    ▶ Header: :path: /telemetry.OpenConfigTelemetry/telemetrySubscribe
    ▶ Header: :authority: 172.24.2.102:32768
    ▶ Header: content-type: application/grpc
    ▶ Header: user-agent: grpc-go/1.11.3
    ▶ Header: te: trailers
  ▶ Stream: DATA, Stream ID: 3, Length 73
  ▶ GRPC Message: /telemetry.OpenConfigTelemetry/telemetrySubscribe, Request
  ▶ Protocol Buffers: application/grpc,/telemetry.OpenConfigTelemetry/telemetrySubscribe,request
```

```
0000  00000000 01010000 01010110 10011011 11000001 01010001 00000000 01010000   ·PV··Q·P
0008  01010110 10011011 11100100 01110001 00001000 00000000 01000101 00000000   V··q··E·
0010  00000000 10111010 01001001 00100110 01000000 00000000 00111111 00000110   ··I&@·?·
0018  10010101 00011100 10101100 00011000 00000010 01100101 10101100 00011000   ·····e··
0020  00000010 01100110 10110110 11100110 10000000 00000000 01001000 00001111   ·f····H·
0028  11010001 00000110 10100100 10011011 10101111 11111010 10000000 00011000   ········
0030  00000000 11101101 01011101 10101000 00000000 00000000 00000001 00000001   ··]·····
0038  00001000 00001010 00001111 11101101 10011001 01101111 10010000 00000010   ·····o··
0040  01001100 01001111 00000000 00000000 00101011 00000001 00000100 00000000   LO··+···
0048  00000000 00000000 00000011 10000011 10000110 01000101 10100011 01100001   ·····E·a
0050  00100101 10100000 10110100 10010101 00110110 01111010 01011111 01010101   %···6z_U
0058  01100101 10101010 11110001 11101010 10010100 11010011 01101111 00101101   e·····o-
0060  00000101 10100100 10101001 10110011 11010011 00001001 00101101 00000101   ·····-·
0068  10100100 10101001 10110011 11010110 11101011 01100011 01000001 00101100   ·····cA,
0070  00110100 01100101 11000010 11000001 11000000 10111111 00000000 00000000   4e·····
0078  01001001 00000000 00000001 00000000 00000000 00000000 00000011 00000000   I·····
0080  00000000 00000000 00000000 01000100 00010010 01000000 00001010 00111011   ···D·@·;
0088  00101111 01101110 01100101 01110100 01110111 01101111 01110010 01101011   /network
0090  00101101 01101001 01101110 01110011 01110100 01100001 01101110 01100011   -instanc
0098  01100101 01110011 00101111 01101110 01100101 01110100 01110111 01101111   es/netwo
```

OpenConfig Telemetry with gRPC and GPB

No.	Time	Source	Destination	Protocol	Length	Info
1	2019-10-04 16:14:02.554041	172.24.2.101	172.24.2.102	TCP	74	40104 → 32768 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=268064592 TSecr=0 WS=128
2	2019-10-04 16:14:02.554238	172.24.2.102	172.24.2.101	TCP	78	32768 → 40104 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=2 TSval=2416904783 TSecr=268064592…
3	2019-10-04 16:14:02.554276	172.24.2.101	172.24.2.102	TCP	66	40104 → 32768 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=268064592 TSecr=2416904783
4	2019-10-04 16:14:02.554474	172.24.2.101	172.24.2.102	HTTP2	90	Magic
5	2019-10-04 16:14:02.554503	172.24.2.101	172.24.2.102	HTTP2	81	SETTINGS[0]
6	2019-10-04 16:14:02.554550	172.24.2.102	172.24.2.101	TCP	66	32768 → 40104 [ACK] Seq=1 Ack=40 Win=66568 Len=0 TSval=2416904783 TSecr=268064592
7	2019-10-04 16:14:02.554683	172.24.2.102	172.24.2.101	HTTP2	106	SETTINGS[0], WINDOW_UPDATE[0]
8	2019-10-04 16:14:02.554698	172.24.2.101	172.24.2.102	GRPC	198	HEADERS[1]: POST /authentication.Login/LoginCheck, DATA[1] (GRPC) (PROTOBUF)
9	2019-10-04 16:14:02.554717	172.24.2.101	172.24.2.102	TCP	66	40104 → 32768 [ACK] Seq=172 Ack=41 Win=29312 Len=0 TSval=268064592 TSecr=2416904783
10	2019-10-04 16:14:02.554758	172.24.2.102	172.24.2.101	HTTP2	75	SETTINGS[0]
11	2019-10-04 16:14:02.554781	172.24.2.101	172.24.2.102	TCP	66	40104 → 32768 [ACK] Seq=172 Ack=50 Win=29312 Len=0 TSval=268064592 TSecr=2416904783
12	2019-10-04 16:14:02.554833	172.24.2.101	172.24.2.102	HTTP2	75	SETTINGS[0]
13	2019-10-04 16:14:02.554891	172.24.2.102	172.24.2.101	TCP	66	32768 → 40104 [ACK] Seq=50 Ack=181 Win=66598 Len=0 TSval=2416904783 TSecr=268064592
14	2019-10-04 16:14:02.580176	172.24.2.102	172.24.2.101	GRPC	203	HEADERS[1]: 200 OK, DATA[1], HEADERS[1] (GRPC) (PROTOBUF)
15	2019-10-04 16:14:02.580543	172.24.2.101	172.24.2.102	GRPC	200	HEADERS[3]: POST /telemetry.OpenConfigTelemetry/telemetrySubscribe, DATA[3] (GRPC) (PROTOBUF)
16	2019-10-04 16:14:02.580749	172.24.2.102	172.24.2.101	HTTP2	81	SETTINGS[0]
17	2019-10-04 16:14:02.580844	172.24.2.101	172.24.2.102	HTTP2	75	SETTINGS[0]
18	2019-10-04 16:14:02.680068	172.24.2.102	172.24.2.101	TCP	66	32768 → 40104 [ACK] Seq=202 Ack=324 Win=66608 Len=0 TSval=2416904909 TSecr=268064619
19	2019-10-04 16:14:02.861916	172.24.2.102	172.24.2.101	HTTP2	232	HEADERS[3]: 200 OK
20	2019-10-04 16:14:02.901869	172.24.2.101	172.24.2.102	TCP	66	40104 → 32768 [ACK] Seq=324 Ack=368 Win=31360 Len=0 TSval=268064940 TSecr=2416905090
21	2019-10-04 16:14:04.984436	172.24.2.102	172.24.2.101	HTTP2	5858	DATA[3][Packet size limited during capture]
22	2019-10-04 16:14:04.984491	172.24.2.101	172.24.2.102	TCP	66	40104 → 32768 [ACK] Seq=324 Ack=6160 Win=43008 Len=0 TSval=268067022 TSecr=2416907213
23	2019-10-04 16:14:04.984542	172.24.2.102	172.24.2.101	HTTP2	2962	Unknown type (110)[1953064553]
24	2019-10-04 16:14:04.984559	172.24.2.102	172.24.2.101	HTTP2	1704	Unknown type (112)[1701981543]
25	2019-10-04 16:14:04.984565	172.24.2.101	172.24.2.102	TCP	66	40104 → 32768 [ACK] Seq=324 Ack=9056 Win=48768 Len=0 TSval=268067022 TSecr=2416907213
26	2019-10-04 16:14:04.984576	172.24.2.101	172.24.2.102	TCP	66	40104 → 32768 [ACK] Seq=324 Ack=10694 Win=52096 Len=0 TSval=268067022 TSecr=2416907213
31	2019-10-04 16:14:05.983935	172.24.2.102	172.24.2.101	HTTP2	10202	DATA[3][Packet size limited during capture]
32	2019-10-04 16:14:05.983993	172.24.2.101	172.24.2.102	TCP	66	40104 → 32768 [ACK] Seq=324 Ack=20830 Win=72320 Len=0 TSval=268068022 TSecr=2416908212
33	2019-10-04 16:14:05.984055	172.24.2.102	172.24.2.101	TCP	258	32768 → 40104 [PSH, ACK] Seq=20830 Ack=324 Win=66608 Len=192 TSval=2416908213 TSecr=268068022 [TCP …
34	2019-10-04 16:14:05.984078	172.24.2.101	172.24.2.102	TCP	66	40104 → 32768 [ACK] Seq=324 Ack=21022 Win=75264 Len=0 TSval=268068022 TSecr=2416908213
35	2019-10-04 16:14:05.984187	172.24.2.101	172.24.2.102	HTTP2	79	WINDOW_UPDATE[0]
36	2019-10-04 16:14:06.011461	172.24.2.102	172.24.2.101	HTTP2	81	SETTINGS[0]
37	2019-10-04 16:14:06.011547	172.24.2.101	172.24.2.102	HTTP2	75	SETTINGS[0]
38	2019-10-04 16:14:06.111952	172.24.2.102	172.24.2.101	TCP	66	32768 → 40104 [ACK] Seq=21037 Ack=346 Win=66608 Len=0 TSval=2416908340 TSecr=268068049

```
▶ Frame 8: 198 bytes on wire (1584 bits), 198 bytes captured (1584 bits)
▶ Ethernet II, Src: Vmware_9b:e4:71 (00:50:56:9b:e4:71), Dst: Vmware_9b:c1:51 (00:50:56:9b:c1:51)
▶ Internet Protocol Version 4, Src: 172.24.2.101, Dst: 172.24.2.102
▶ Transmission Control Protocol, Src Port: 40104, Dst Port: 32768, Seq: 40, Ack: 1, Len: 132
▼ HyperText Transfer Protocol 2
  ▶ Stream: HEADERS, Stream ID: 1, Length 78, POST /authentication.Login/LoginCheck
  ▶ Stream: DATA, Stream ID: 1, Length 36
  ▶ GRPC Message: /authentication.Login/LoginCheck, Request
  ▶ Protocol Buffers: application/grpc,/authentication.Login/LoginCheck,request
```

```
0040  00001010 01001111 00000000 00000000 01001110 00000001 00000100 00000000   ·O··N···
0048  00000000 00000000 00000001 10000011 10000110 01000101 10010111 01100000   ·····E·`
0050  01110110 10100110 01110010 11010100 10010011 00010000 01101001 00110001   v·r···i1
0058  11101010 01011111 00111001 11100110 00110101 01001100 01100111 00111100   ·_9·5Lg<
0060  11000110 10101010 11110100 11100101 00100111 01011111 01000001 10001101   ····'_A·
0068  00001011 10100010 01011100 01001101 00101110 00100101 11000010 00000001   ··\M.%··
0070  01011100 01100100 01001110 10111000 11110111 01011111 10001011 00011101   \dN··_··
```

TICK Stack – example for analytics

https://www.influxdata.com/time-series-platform/

Telegraf with gRPC and Junos

```
[[inputs.jti_openconfig_telemetry]]
  servers = ["172.24.2.102:32768"]
  username = "automation"
  password = …
  client_id = "telegraf"
  sample_frequency = "10000ms"
  sensors = [
   "10000ms interface /interfaces/",
   "10000ms bgp /network-instances/network-instance/protocols/protocol/bgp/"
  ]
```

```
request-response {
    grpc {
        clear-text {
            port 32768;
        }
        skip-authentication;
    }
}
notification {
    allow-clients {
        address 0.0.0.0/0;
    }
}
```

```
psievers@vmx> show system connections | match 32768
tcp4       0      0  172.24.2.102.32768                     172.24.2.101.39334                     ESTABLISHED
tcp46      0      0  *.32768                                *.*                                    LISTEN
```

```
psievers@vmx> show agent sensors
Sensor Information :
    Name                                : sensor_1007
    Resource                            : /network-instances/network-instance/protocols/protocol/bgp/
    Version                             : 1.0
    Sensor-id                           : 539528116
    Subscription-ID                     : 1007
    Parent-Sensor-Name                  : Not applicable
    Component(s)                        : rpd
    Profile Information :
        Name                            : export_1007
        Reporting-interval              : 10
        Payload-size                    : 5000
        Format                          : GPB
```

Kapacitor with UDF

```
# kapacitor list tasks
ID               Type      Status    Executing Databases and Retention Policies
vmx_3sigma_1day  batch     disabled  false     ["vMX:vmx"."vMX:vmx"]
vmx_3sigma_3days batch     disabled  false     ["vMX:vmx"."vMX:vmx"]
vmx_3sigma_5days batch     disabled  false     ["vMX:vmx"."vMX:vmx"]
vmx_3sigma_7days batch     disabled  false     ["vMX:vmx"."vMX:vmx"]
```

```
# kapacitor show vmx_3sigma_3days
ID: vmx_3sigma_3days
...
TICKscript:
dbrp "vMX:vmx"."vMX:vmx”
var from_database = 'vMX:vmx’
var to_database = 'vMX:vmx’
batch
    |query('''
            SELECT *
            FROM "vMX:vmx"."vMX:vmx"."bgp/bgp-routes"
        ''')
        .period(259200s)
        .every(4320s)
        .groupBy('neighbor-address')
        .alignGroup()
    @train()
        .algorithm('3sigma')
        .pattern_periodicity('1H')
        .field_name('received-routes')
        .device_id('vmx’)
...
```

kapacitor for **data processing**

- create Alerts
- run Extract, Transform, Load Jobs
- detect anomalies

```
# kapacitord config | sed -n -e "/udf/,\$p”
[udf]
  [udf.functions]
    [udf.functions.train]
      # Run python
      timeout = “10m0s"
      socket = ""
      prog = "/usr/bin/python”
      # Pass args to python
      # -u for unbuffered STDIN and STDOUT
      # and the path to the script
      args = ["-u", "/etc/kapacitor/udfs/training.py"]
      [udf.functions.train.env]
        PYTHONPATH = "/usr/lib/kapacitor/udf"
```

Prepare data – collection

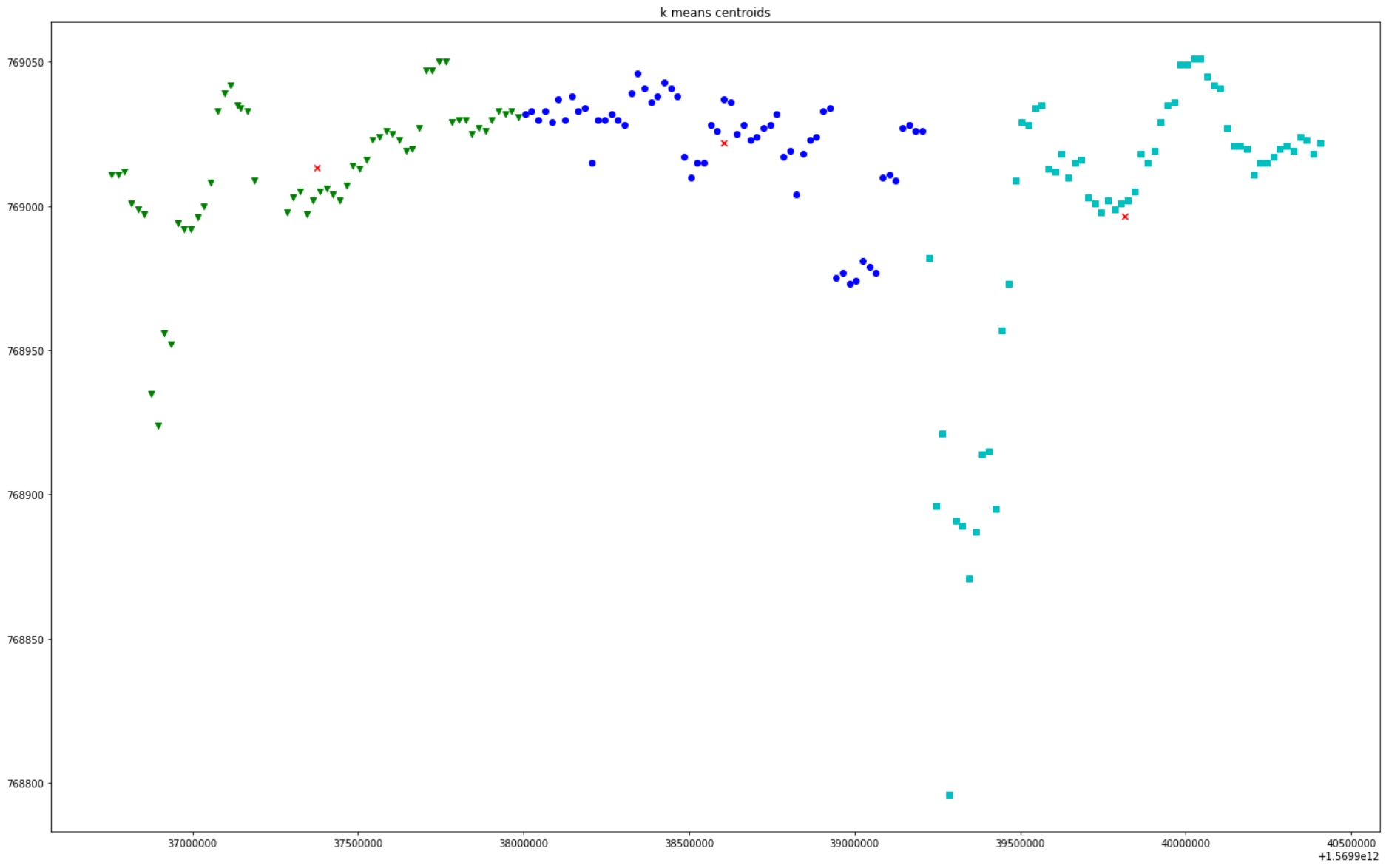
- historical structered data used for analysis and model training
- learn patterns in data
  - algorithm – used to train the model
    - 3sigma
      - gaussian distribution, standard deviation
      - 68–95–99.7 rule
    - K-means clustering
      - find groups of data which are similar to one another called clusters
      - “k” centroids (for example k=5)
      - depends on the distance between points to be clustered
- Overfitting
  - model describes random errors or noise instead of the underlying relationships

Python with sklearn, numpy, pandas – pickle

```
# python2
>>> import os
>>> import pandas as pd
>>> import collections
>>> import operator
>>> import pprint
>>> unpickled_df = pd.read_pickle("./1day_pickle")
>>> pprint.pprint(unpickled_df)
{(0,): {'avg': 772705.2444444444, 'std': 30.948006976257275},
 (1,): {'avg': 772733.9888888889, 'std': 57.31016866227813},
 (2,): {'avg': 772783.1388888889, 'std': 14.360735004746212},
 (3,): {'avg': 772781.6861111111, 'std': 16.80240810746711},
 (4,): {'avg': 772788.9777777778, 'std': 42.3026076889351},
 (5,): {'avg': 772742.1, 'std': 17.665911933564157},
 (6,): {'avg': 772717.2361111111, 'std': 41.202107907092014},
 (7,): {'avg': 772501.6111111111, 'std': 112.14558439263416},
 (8,): {'avg': 772443.0944444444, 'std': 54.54582143301601},
 (9,): {'avg': 772425.4388888889, 'std': 17.6423303729313},
 (10,): {'avg': 772479.5805555555, 'std': 42.90453433330823},
 (11,): {'avg': 772574.5194444444, 'std': 65.23597899355013},
 (12,): {'avg': 772611.5305555556, 'std': 44.430959673060585},
 (13,): {'avg': 772537.1722222222, 'std': 47.06801586304601},
 (14,): {'avg': 772478.7444444444, 'std': 32.991012618291165},
 (15,): {'avg': 772469.7583333333, 'std': 31.823557617658725},
 (16,): {'avg': 772532.9166666666, 'std': 58.9065054886885},
 (17,): {'avg': 772396.5583333333, 'std': 30.721001457562473},
 (18,): {'avg': 772510.3361111111, 'std': 100.67348666957889},
 (19,): {'avg': 772630.4694444444, 'std': 39.19657240856969},
 (20,): {'avg': 772606.3611111111, 'std': 35.49738079365683},
 (21,): {'avg': 772652.8777777777, 'std': 37.08139029449732},
 (22,): {'avg': 772655.5555555555, 'std': 33.67091099025359},
 (23,): {'avg': 772684.225, 'std': 28.406805473727985}}
```

```
# python2
>>> import os
>>> import pandas as pd
>>> import collections
>>> import operator
>>> import pprint
>>> unpickled_df = pd.read_pickle("./3days_pickle")
>>> pprint.pprint(unpickled_df)
{(0,): {'avg': 772327.7851851851, 'std': 128.11257406036736},
 (1,): {'avg': 772366.6055555556, 'std': 168.2699714817505},
 (2,): {'avg': 772381.6231481482, 'std': 199.0378793775853},
 (3,): {'avg': 772346.4981481482, 'std': 199.55844775565058},
 (4,): {'avg': 772352.3962962963, 'std': 194.75334919666878},
 (5,): {'avg': 772343.7981481481, 'std': 206.04103779545932},
 (6,): {'avg': 772303.6138888889, 'std': 142.8161081006785},
 (7,): {'avg': 772337.0242990655, 'std': 161.33095770566888},
 (8,): {'avg': 772281.050925926, 'std': 116.7961511569135},
 (9,): {'avg': 772296.3935185185, 'std': 138.57995337172298},
 (10,): {'avg': 772344.1148148148, 'std': 163.13031533262378},
 (11,): {'avg': 772403.9388888889, 'std': 165.30547471011297},
 (12,): {'avg': 772406.2462962962, 'std': 199.8359245619812},
 (13,): {'avg': 772243.5055555556, 'std': 198.82456897575577},
 (14,): {'avg': 772152.3175925926, 'std': 223.44188917624385},
 (15,): {'avg': 772210.2851851851, 'std': 189.28649695281862},
 (16,): {'avg': 772206.4064814815, 'std': 207.96487985784208},
 (17,): {'avg': 772283.4796296296, 'std': 169.41362803158955},
 (18,): {'avg': 772293.7074074075, 'std': 195.18573666377924},
 (19,): {'avg': 772266.925, 'std': 188.27604567496098},
 (20,): {'avg': 772303.6916666667, 'std': 150.346286473865},
 (21,): {'avg': 772323.812037037, 'std': 173.48203248708634},
 (22,): {'avg': 772320.4805555556, 'std': 174.52361506256446},
 (23,): {'avg': 772340.0462962963, 'std': 150.99801226673418}}
```

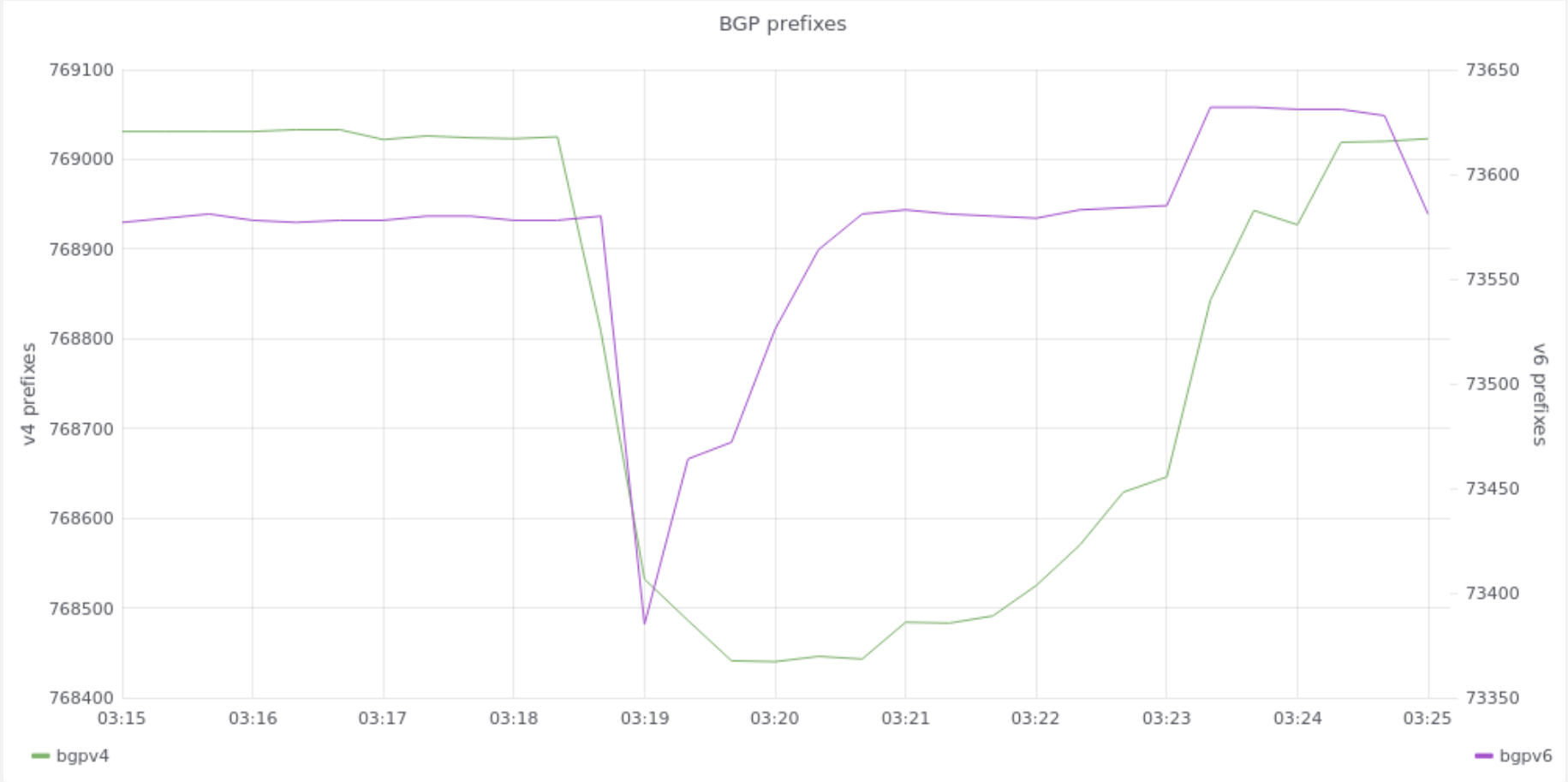
K-means for received routes – centroids



statistical anomalies detection for received routes

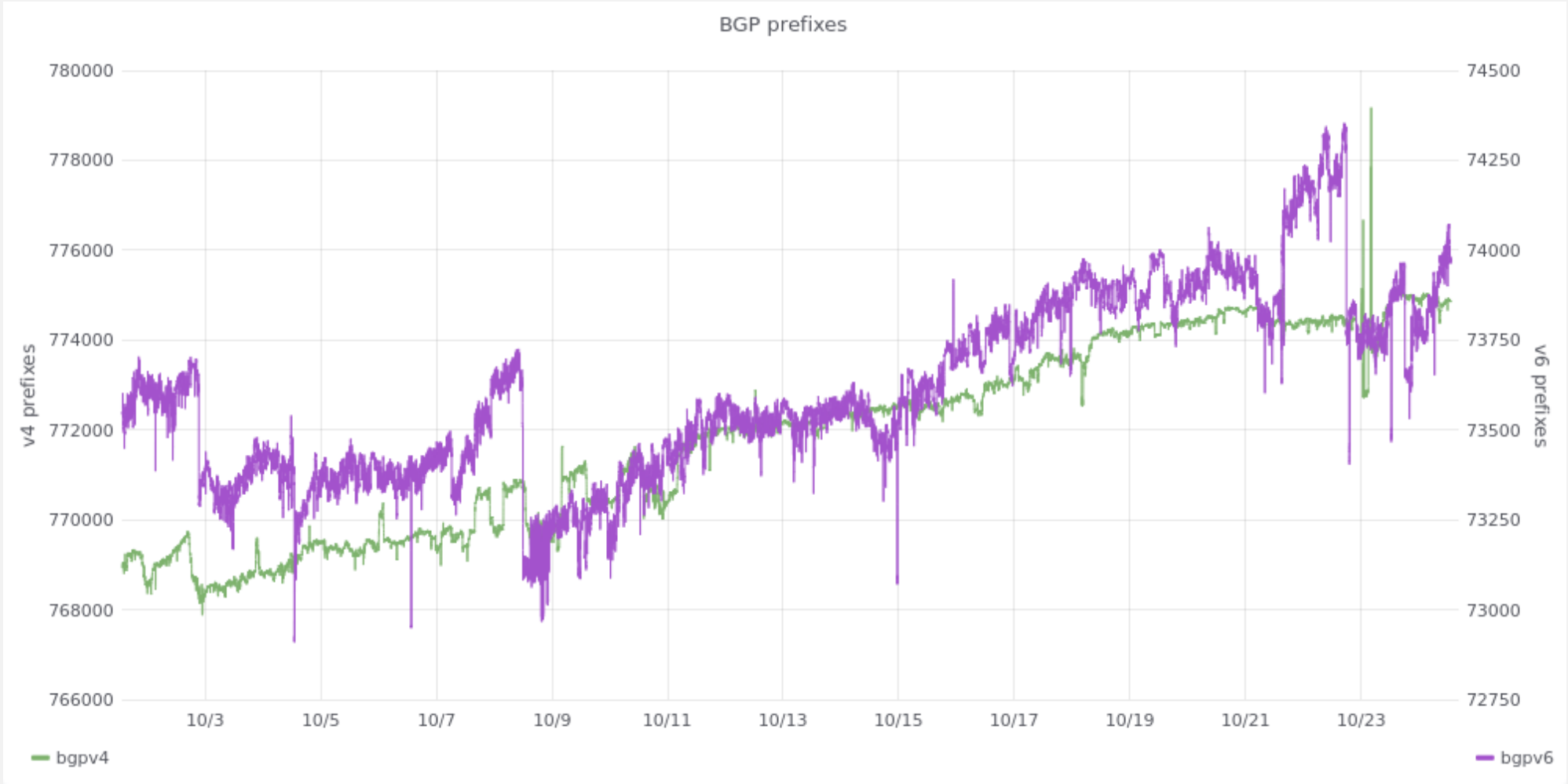
Anomaly – in context of received bgp routes

- number of received routes is higher or lower as expected*
- see graph, approximately >500 v4 prefixes



Anomaly – model meets real life

- v6 NLRIs look like the playground for BGP routing...

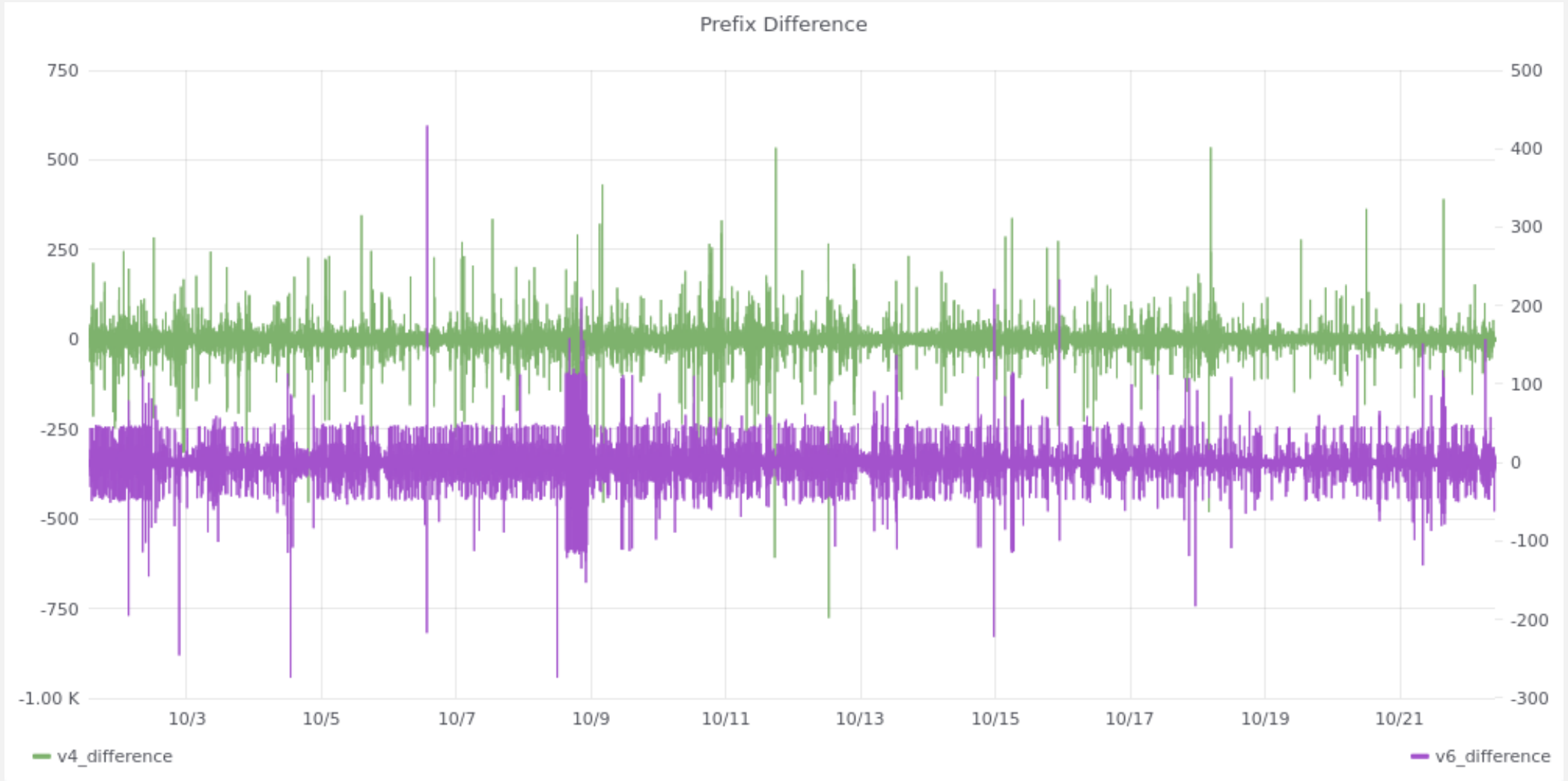


Anomaly – seasonality

- seasonal factors – days vs. nights – weekdays vs. weekends
- but what when it happens every day at the same timeframe – still an anomaly?

Anomaly or normality?

- v4 and v6 prefixes - SELECT DIFFERENCE(LAST("received-routes")) FROM…..



Moving on – Ideas – Remarks

- cross-correlation between two different BGP upstream feeds
  - for example compare standard deviation for received routes
- cross-correlation between different metrics
  - PFE memory allocation and number of routes in fib
  - corresponding interface output packets and “received routes anomaly”

- analysis was done on the basis of one BGP full feed
- TICK was just an example, use whatever TSDB you like
- for further debugging which prefixes are involved use also bmp, exabgp, ….

Resources

- https://www.juniper.net/us/en/products-services/sdn/contrail/contrail-healthbot/
- https://github.com/juniper/yang
- https://github.com/openconfig
- https://github.com/YangModels/yang
- https://docs.influxdata.com/kapacitor/v1.5/guides/anomaly_detection/
- https://github.com/ksator/openconfig-demo-with-juniper-devices
- https://learning.oreilly.com/library/view/network-programmability-with/9780135180471/
- https://scikit-learn.org/stable/
- https://pandas.pydata.org/

Thank You