YANG PUSH INTEGRATION INTO APACHE KAFKA

INF 690: Intership of IOT

Author: Zhuoyao Lin

31 août 2023

TABLE OF CONTENT

I. INTRODUCTION: Motivation

Challenge of Network Management

Networks are growing and being more and more complex to manage.

A costly task for network operators is to identify and correct network issues.

This involves three steps:

- **Detect** that there is an issue,
- **Identify** the cause of issue,
- **Correct** the issue

Today, it is often the customer that notifies the operator about an issue. Then the cause and correction of an issue is done by expert engineers.

The challenge of the network management industry today, is to automate the detection, identification and correction of the problem.

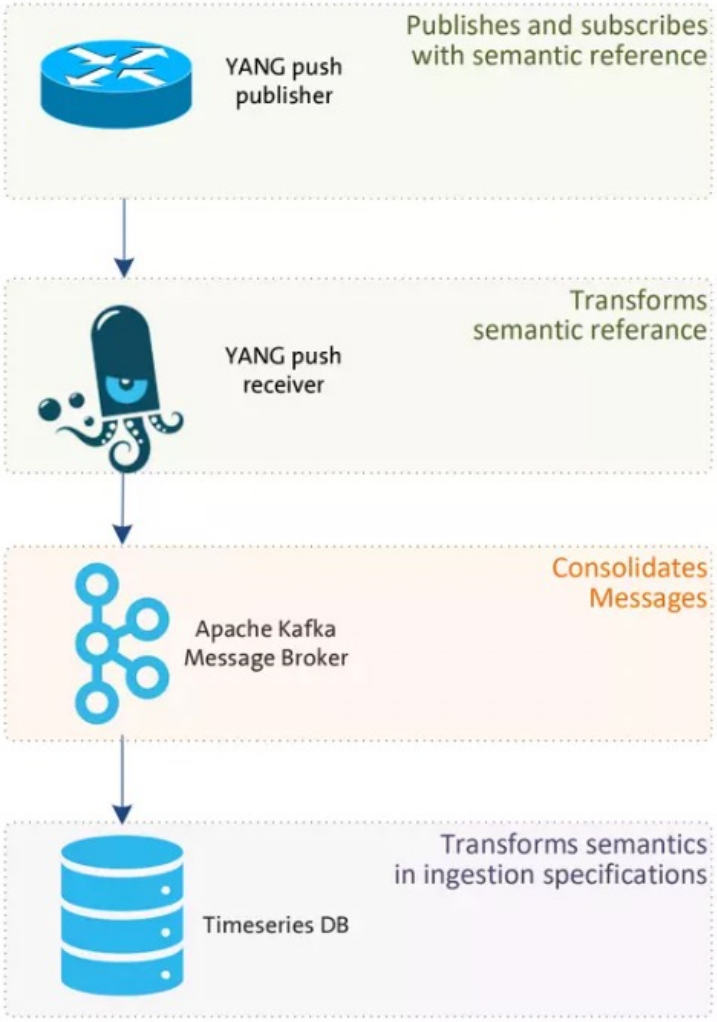
I. INTRODUCTION: Motivation

Network Analytics

Nowadays Network Analytics requires to correlate metrics from various sources to make **accurate detection/prediction**.

There are different sources and protocols to collect metrics. Here we focus on the **YANG push protocol**.

The metrics are the foundation to network analytics. We must have **reliable and meaningful data collection.**



Missing Semantics

In the TSDB, YANG model semantics might be lost. It is not clear how to interpret the data.

An example: Temperature

Data Mesh

The principle of Data Mesh is to present the data as a product. The quality of the data is guarantee by the team preparing the data.

In our case, the collected data comes with the YANG model.

I. INTRODUCTION: Overview

YANG Schema Registry

A solution to keep the semantics is to have a schema registry. Each message in Kafka contains a schema id pointing to the original YANG model.

Schema will be used during serialization and deseralization.

YANG push Receiver

Apart from collecting YANG data, the receiver needs to be extended to register schema for YANG subscription.

This is the goal of **libyangpush**.

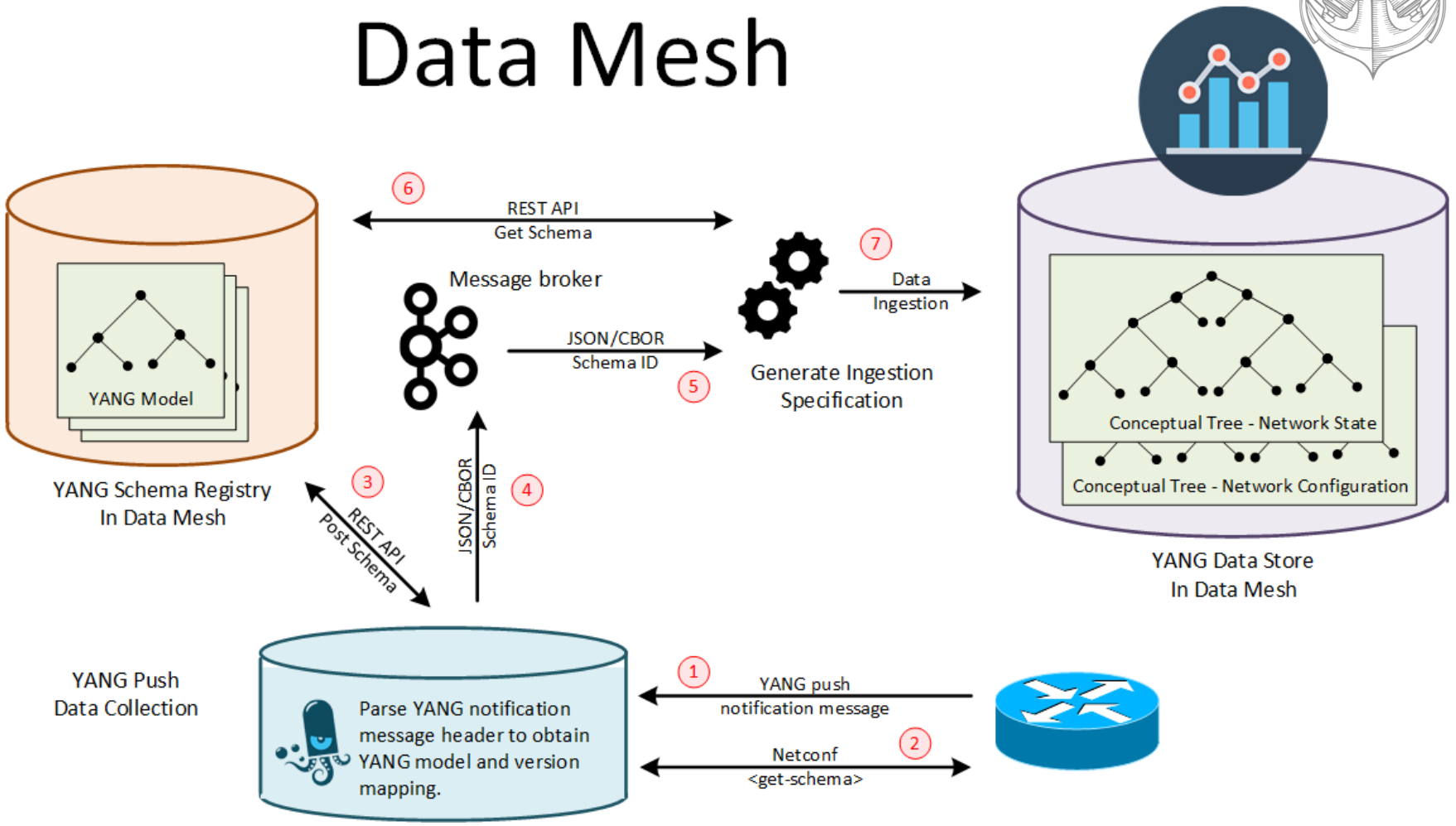
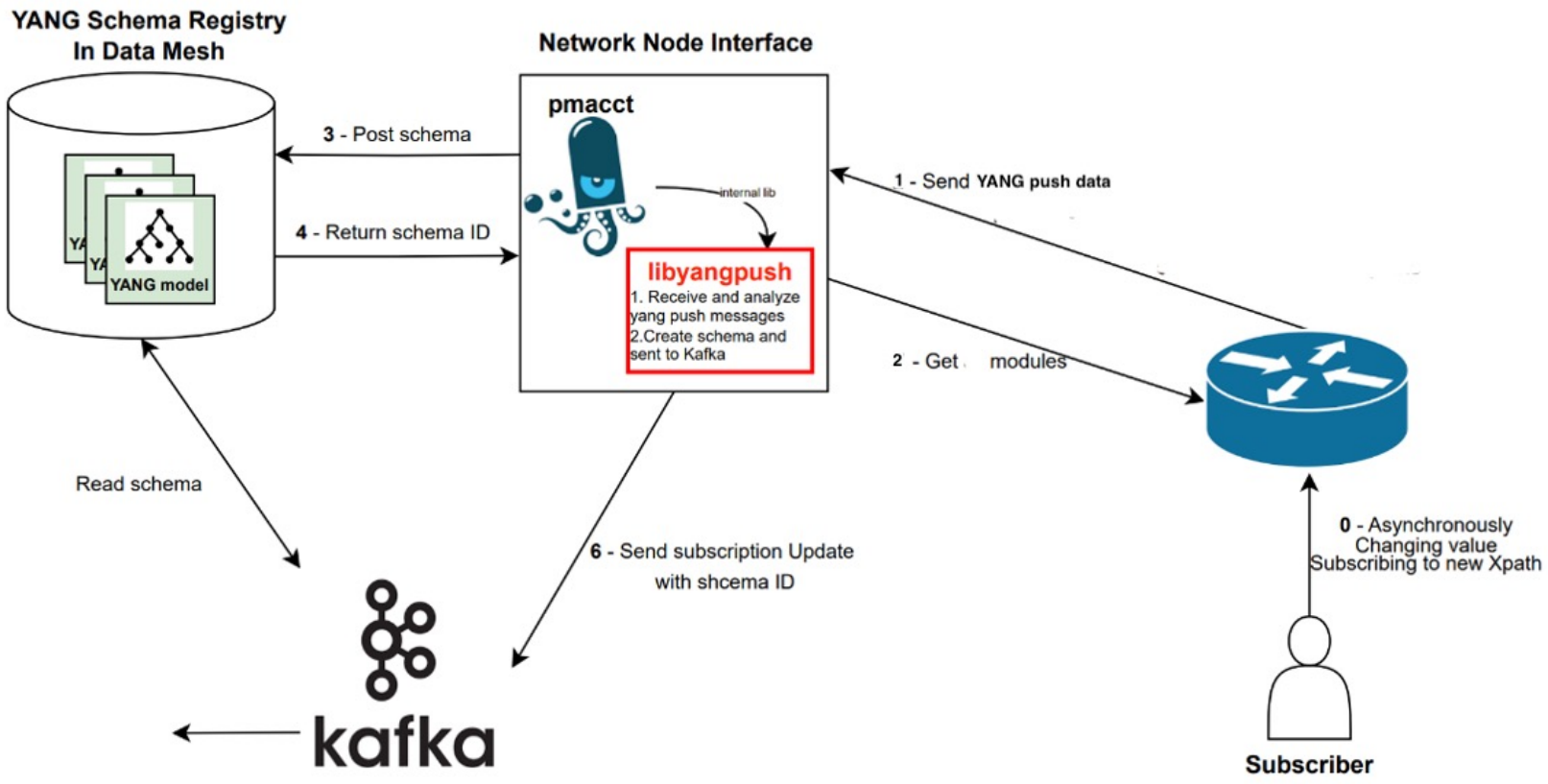


TABLE OF CONTENT

II. Design: High level view



pmacct: A multi-purpose Network Monitoring tool(https://github.com/pmacct/pmacct)

Schema registry: Confluent pluggable schema registry has been extended to natively support YANG.

YANG push:

1. Subscription to YANG Notifications for Datastore Updates - RFC 8641

2. draft-ietf-netconf-udp-notif-10

It is partially being supported in the device

II. Design: YANG dependencies in a nutshell

Yang is a schema language. It represents data structures in a tree format.

Dependency types:

Import: The "import" statement makes definitions from one module available inside another module or submodule.

Include: The "include" statement is used to make content from a submodule available to that submodule's parent module.

Augment: The "augment" statement allows a module or submodule to add to the schema tree defined in an external module

Deviate: The "deviate" statement defines how the device's implementation of the target node deviates from its original definition.

```
"ietf-interfaces:interfaces": [
 {
  "interface": {
   "name": "eth0",
   "type": "iana-if-type:ethernetCsmacd",
   "oper-status": "up",
   "speed": "1000000"
   "ietf-ip:ipv4": {
    "enabled": true,
    "forwarding": true
   }
  }
 }
]
```

```
identity ethernetCsmacd {
  base iana-interface-type;
  description
    "For all Ethernet-like interfaces, regardless of speed,
     as per RFC 3635.";
  reference
    "RFC 3635 - Definitions of Managed Objects for the
                Ethernet-like Interface Types";
}
```

```
augment /if:interfaces/if:interface:
    +--rw ipv4!
    |  +--rw address* [ip]
    |  |  +--rw ip                    inet:ipv4-address-no-zone
    |  |  +--rw (subnet)
    |  |  |  +--:(prefix-length)
    |  |  |  |  +--rw prefix-length?  uint8
    |  |  |  +--:(netmask)
    |  |  |     +--rw netmask?        yang:dotted-quad
```

```
module: ietf-interfaces
  +--rw interfaces
  |  +--rw interface* [name]
  |     +--rw name                    string
  |     +--rw type                    identityref
  |     +--ro oper-status             enumeration
  |     +--ro speed?                  yang:gauge64
  | ...
```

II. DESIGN: Identify models involved in a subscription

Use YANG to get subscription information:

Subscribed YANG models can be known by parsing fields: ***datastore-xpath-filter*** or ***datastore-subtree-filter***. The subscription is identified by a sub-id.

This information is exposed as YANG model. It can be obtain:

- **As a subscription.** Subscription will be synced through the first push-update, and updated through the push-change-update(new added, edit, remove etc.)
- **Via NETCONF <get> operation.** Send a <get> to obtain the same information.

```
<subscriptions>
  <subscription>
    <id>6666</id>
    <datastore>ds:operational</datastore>
    <datastore-xpath-filter>
       /a-module:a
    </datastore-xpath-filter>
    <encoding>encode-xml</encoding>
    <periodic>
      <period>30000</period>
    </periodic>
  </subscription>
</subscriptions>
```

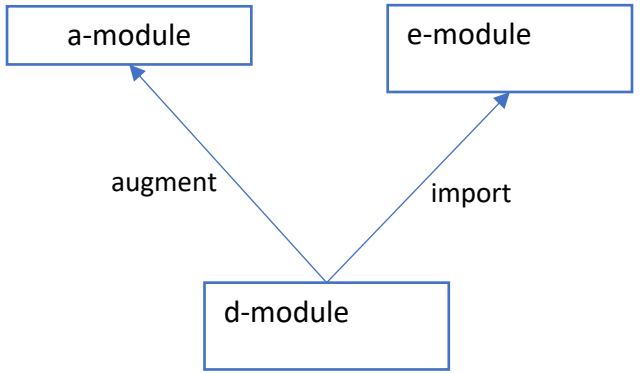
The YANG source can be obtained via NETCONF <get-schema>. Now we need to obtain the full schema and dependencies.

II. DESIGN: Algorithm for finding dependency

Schema Registry: Registration list

The schema registry needs to all dependencies registered before registering the dependent schema.

The augmented model needs to be registered once without reference to the augmenting model, and twice with the reference to the augmenting model, which is registered as a schema.



Schema registration Order:

a-module, reference{}

e-module, reference{}

d-module, reference{e-module, a-module}

a-module, reference{d-module}

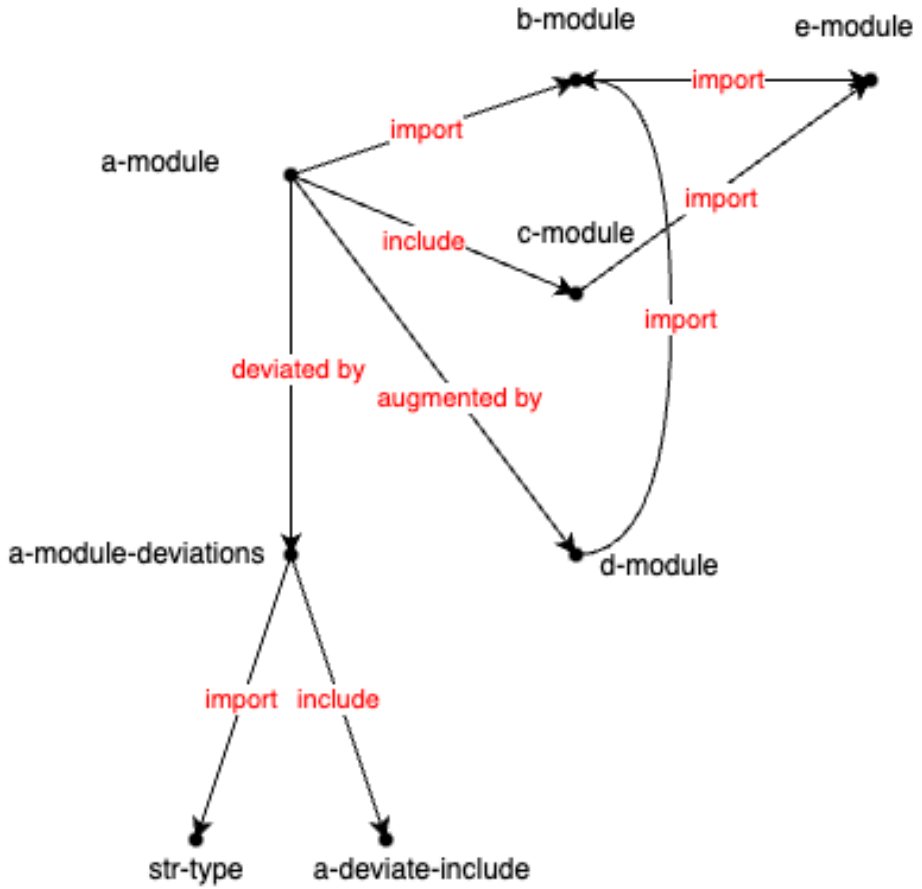
II. DESIGN: Algorithm for finding dependency

Algorithm for finding dependency

Using DFS to traverse the YANG model dependency tree, for the main model: sequentially search for its **import, include, augment and deviate**. For its dependency model, we only search for their **import and include**.

The model with the **greatest depth** will be put into register list first, then the top level module, in order to ensure that each register model has the dependency to refers to in the schema registry.

A hash map is used throughout the traverse with the index being djb2 hash of the model name to make sure that the model will not be put into register list twice.



II. DESIGN: Obtaining the involved models and their dependencies

There are two possible implementation solutions, each with pros and cons

Solution 1:
On-demande Downloading

The idea of on-demand downloading is to send get-schema request to get the YANG models based on the model name we obtain. For the main model in the subscription, we can obtain its name using method in “identify model”.

Import and include can parsed from the YANG code. Deviate can be known by subscribing to */ietf-yang-library:modules-state*.

Pros: schema is update-to-date

Cons: Cannot handle augment

```
x--ro modules-state
  x--ro module-set-id    string
  x--ro module* [name revision]
    x--ro name             yang:yang-identifier
    x--ro revision         union
    +--ro schema?            inet:uri
    x--ro namespace          inet:uri
    x--ro feature*         yang:yang-identifier
    x--ro deviation* [name revision]
    |  x--ro name       yang:yang-identifier
    |  x--ro revision   union
    x--ro conformance-type    enumeration
    x--ro submodule* [name revision]
      x--ro name       yang:yang-identifier
      x--ro revision   union
      +--ro schema?    inet:uri
```

Solution 2:
Get-all-schema

The main idea of get-all-schemas is to get all models, store them in the disk, and analyse their full dependencies (import, include, deviate and augment) at the beginning of connection.

Pros: Can handle all dependencies

Cons: Downloading all schemas takes a lot of time

TABLE OF CONTENT

III. Demo

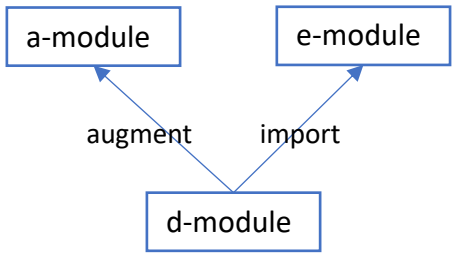
Obtaining YANG semantics and Module dependencies

Overview - Step 1 and 2 in the workflow

Goal: Based on semantic reference in YANG push message, find YANG module dependencies

```
<subscriptions>
  <subscription>
    <id>6666</id>
    <datastore>ds:operational</datastore>
    <datastore-xpath-filter>
       /a-module:a
    </datastore-xpath-filter>
    <encoding>encode-xml</encoding>
    <periodic>
      <period>30000</period>
    </periodic>
  </subscription>
</subscriptions>
```

```
module: a-module
 +--rw a
   +--rw a-instance* [name]
   |  +--rw name     string
   |  +--rw state?   string
   +--rw d:y
     +--rw d:y-leaf?   e:e-enum
```



The schema for a-module require to register a-modules, e-module and d-module.

Schema registration Order:

1. a-module, reference{}
2. e-module, reference{}
3. d-module, reference{e-module, a-module}
4. a-module, reference{e-module, d-module}

III. Demo

Obtaining YANG semantics and Module dependencies

Step 1 – Receiving the YANG Push Message

```
jean@jean-vm:~/code/libyangpush/build$ cat push-update.xml
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2022-09-02T10:59:55.32Z</eventTime>
  <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>2222</id>
    <datastore-contents>
      <subscriptions xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
        <subscription>
          <id>6666</id>
          <datastore xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push"
            xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">ds:operational</datastore>
          <datastore-xpath-filter xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
              /a-module:a
          </datastore-xpath-filter>
          <transport xmlns:unt="urn:ietf:params:xml:ns:yang:ietf-udp-notif-transport">unt:udp-notif</transport>
          <encoding>encode-xml</encoding>
          <receivers>
            <receiver>
              <name>subscription-specific-receiver-def</name>
              <receiver-instance-ref xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notif-receivers">global-udp-notif-receiver-def</receiver-instance-ref>
            </receiver>
          </receivers>
          <periodic xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
            <period>30000</period>
          </periodic>
        </subscription>
      </subscriptions>
    </datastore-contents>
  </push-update>
</notification>
```

This message is passed to libyangpush. It indicates that a new subscription with 6666 is created and that subscription targets the xpath /a-module:a.

III. Demo

Obtaining the YANG semantics and Module dependencies

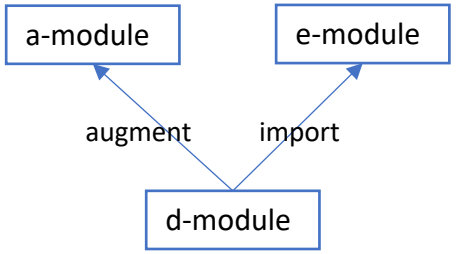
Step 2 – Obtaining the YANG semantics

The demo fetches all YANG modules from the NETCONF server, parses the input messages and selects the corresponding YANG module needed to build the schema for incoming messages on that subscription to register in the Confluent schema register for serializing the messages in Apache Kafka.

```
Base module: a-module
url: http://127.0.0.1:5000/subjects/hackathon_demo_a-module/versions
{
  "schemaType": "YANG",
  "references": [],
  "schema": "module a-module {\n  yang-version 1.1;\n  namespace \"urn:example:yang:a-module\";\n  prefix a;\n\n  container a {\n    list a-instance {\n      key \"name\"
;\n      leaf name {\n        type string;\n      }\n      leaf state {\n        type string;\n      }\n    }\n  }\n}\n"
}
=> schema id 200


Found augmenting module: d-module
Found imported module: e-module
```

Starting from a-module, we discover d-module(augment) then e-module(import for d-module).



III. Demo

Obtaining the YANG semantics and Module dependencies

Step 3 – Register the YANG Semantics in Confluent Schema Registry

```
url: https://schema-registry.app-dev.zhh.sbd.corproot.net/subjects/hackathon_demo_a-module/versions
{
  "schemaType": "YANG",
  "references": [],
  "schema": "module a-module {\n  yang-version 1.1;\n  namespace \"urn:example:yang:a-module\";\n  prefix a;\n\n  container a {\n    list a-instance {\n      key \"name\";\n      leaf name {\n        type str
ing;\n      }\n      leaf state {\n        type string;\n      }\n    }\n  }\n}\n"
}
=> schema id 17


url: https://schema-registry.app-dev.zhh.sbd.corproot.net/subjects/hackathon_demo_e-module/versions
{
  "schemaType": "YANG",
  "references": [],
  "schema": "module e-module {\n  yang-version 1.1;\n  namespace \"urn:example:yang:e-module\";\n  prefix e;\n\n  revision 2023-06-13;\n\n  typedef e-enum {\n    type enumeration {\n      enum \"zero\";\n
}\n  }\n\n  grouping some-groups {\n    leaf e-leaf {\n      type e-enum;\n    }\n  }\n}\n"
}
=> schema id 7


url: https://schema-registry.app-dev.zhh.sbd.corproot.net/subjects/hackathon_demo_d-module/versions
{
  "schemaType": "YANG",
  "references": [
    {
      "subject": "hackathon_demo_a-module",
      "name": "a-module",
      "version": 1
    },
    {
      "subject": "hackathon_demo_e-module",
      "name": "e-module",
      "version": 1
    }
  ],
  "schema": "module d-module {\n  yang-version 1.1;\n  namespace \"urn:example:yang:d-module\";\n  prefix d;\n\n  import a-module {\n    prefix a;\n  }\n  import e-module {\n    prefix e;\n    revision-date 2
023-06-13;\n  }\n\n  augment \"/a:a\" {\n    container y {\n      leaf y-leaf {\n        type e:e-enum;\n      }\n    }\n  }\n}\n"
}
=> schema id 19
```

III. Demo

Obtaining the YANG semantics and Module dependencies

Step 3 – Register the YANG Semantics in Confluent Schema Registry

```
url: https://schema-registry.app-dev.zhh.sbd.corproot.net/subjects/hackathon_demo_augment_a-module/versions
{
  "schemaType": "YANG",
  "references": [
    {
      "subject": "hackathon_demo_e-module",
      "name": "e-module",
      "version": 1
    },
    {
      "subject": "hackathon_demo_d-module",
      "name": "d-module",
      "version": 1
    }
  ],
  "schema": "module a-module {\n  yang-version 1.1;\n  namespace \"urn:example:yang:a-module\";\n  prefix a;\n\n  container a {\n    list a-instance {\n      key \"name\";\n      leaf name {\n        type str
ing;\n      }\n      leaf state {\n        type string;\n      }\n    }\n  }\n}\n"
}
=> schema id 20
```

IV. Conclusion & Future Work

Outcomes

Demo is presented during **IETF117 Hackathon**: https://wiki.ietf.org/en/meeting/117/hackathon#network-telemetry-yang-push-integration-into-apache-kafka

Demonstrate the interaction with a working YANG schema registry during the demo

Github repository for **libyangpush** in organization network-analytics.

Gaps to fill

- Create a YANG model for notification message
- Integration with pmacct
- Test with device as YANG push becomes better supported
- Augment YANG model ***ietf-yang-library*** to support the record of augmentations(to support the on-demand downloading)

Future Work

- Deploy the functionalities into Swisscom lab.
- Deliver as an internal product for Huawei.

Industry Post:
https://github.com/graf3net/draft-daisy-kafka-yang-integration/blob/main/draft-daisy-kafka-yang-integration-03.md
https://www.linkedin.com/pulse/network-analytics-ietf-115-london-thomas-graf/
https://www.linkedin.com/pulse/network-analytics-ietf-116-yokohama-thomas-graf/
https://www.linkedin.com/pulse/network-analytics-ietf-117-san-francisco-thomas-graf/

Thank You