SESSION 2
YANG, OpenConfig, and gNMI

Session 2 Overview

1. YANG: Configuration Modeling Language
2. OpenConfig: Configuration and telemetry model instances
3. gNMI: Runtime Configuration and Monitoring Interface
4. gNOI: Runtime Operations Interface

YANG Overview

YANG is a data modeling language for network configuration

- Used to express the structure of data, NOT the data itself
- Instances of data can be expressed in XML, JSON, Protobuf, etc. and are considered valid if they adhere to the YANG data model (schema)

From a YANG model, we care about 2 things:

1. Data tree organization (from which we get the paths and leaf data types)
2. Semantics of the leaf nodes (from the description field, usually in English)

History: YANG was originally designed as a data modeling language for NETCONF. It borrows the syntactic structure and base types from SMIng, which is an evolution of SMI, the data modeling language for SNMP used for MIBs.

YANG Module

A **module** is a self-contained tree of nodes.
Modules are the smallest unit that can be “compiled” by YANG tools.

```
// A module is a self-contained tree of nodes
module demo-port {

   // YANG Boilerplate
   yang-version "1";
   namespace "https://opennetworking.org/yang/demo";
   prefix "demo-port";
   description "Demo model for managing ports";
   revision "2019-09-10" {
     description "Initial version";
     reference "1.0.0";
   }

   // ... insert rest of model here ...
}
```

A **module** contains:

- boilerplate, like a namespace, prefix for reference in other modules, description, version / revision history, etc.
- identities and derived types
- modular groupings
- a top-level container that defines tree of data nodes

YANG Types

YANG defines several built-in types (including `binary`, `bits`, `boolean`, `decimal64`, `empty`, `enumeration`, `identityref`, `int8`, `int16`, `int32`, `int64`, `string`, `uint8`, `uint16`, `uint32`, `uint64`, `decimal64`).

For the complete list, check out RFC 6020 for YANG 1.0 or RFC 7950 for YANG 1.1

```
// Identities and Typedefs
identity SPEED {
  description "base type for port speeds";
}

identity SPEED_10GB {
  base SPEED;
  description "10 Gbps port speed";
}

typedef port-number {
  type uint16 {
    range 1..32;
  }
  description "New type for port number that ensure
    the number is between 1 and 32, inclusive";
}
```

An **identity** is a globally unique, abstract, and untyped. Identities are used to identify something with explicit semantics and can be hierarchical.

Derived types enable constraint of build-int types or other derived types, and they are defined using **typedef**.

YANG Groupings

A **grouping** is a reusable set of nodes (containers and leaves) that can be included in a **container**. However, on its own a grouping does not add any nodes to the module in which it is defined or imported.

```
// Reusable groupings for port config and state
grouping port-config {
    description "Set of configurable attributes / leaves";
    leaf speed {
        type identityref {
            base demo-port:SPEED;
        }
        description "Configurable speed of a switch port";
    }
}

grouping port-state {
    description "Set of read-only state";
    leaf status {
        type boolean;
        description "Number";
    }
}
```

A **leaf** is a node that contains a value (built-in type or derived type) and has no children.

YANG Containers

A **container** is a node with a set of children. Each module has one top-level or root container.

```
container ports {
  description "The root container for port configuration and state";
    list port {
        key "port-number";
        description "List of ports on a switch";

        leaf port-number {
            type port-number;
            description "Port number (maps to the front panel port of a switch);
              also the key for the port list";
        }

        // each individual will have the elements defined in the grouping
        container config {
            description "Configuration data for a port";
            uses port-config; // reference to grouping above
        }
        container state {
            config false; // makes child nodes read-only
            description "Read-only state for a port";
            uses port-state; // reference to grouping above
        }
    }
}
```

A **list** is a node that contains a set of multiple children of the same type. Lists elements are identified by a **key**.

Containers marked **`config false`** are state data that is read-only from a clients perspective. Typically, it is used for status or statistics.

Other YANG Terminology

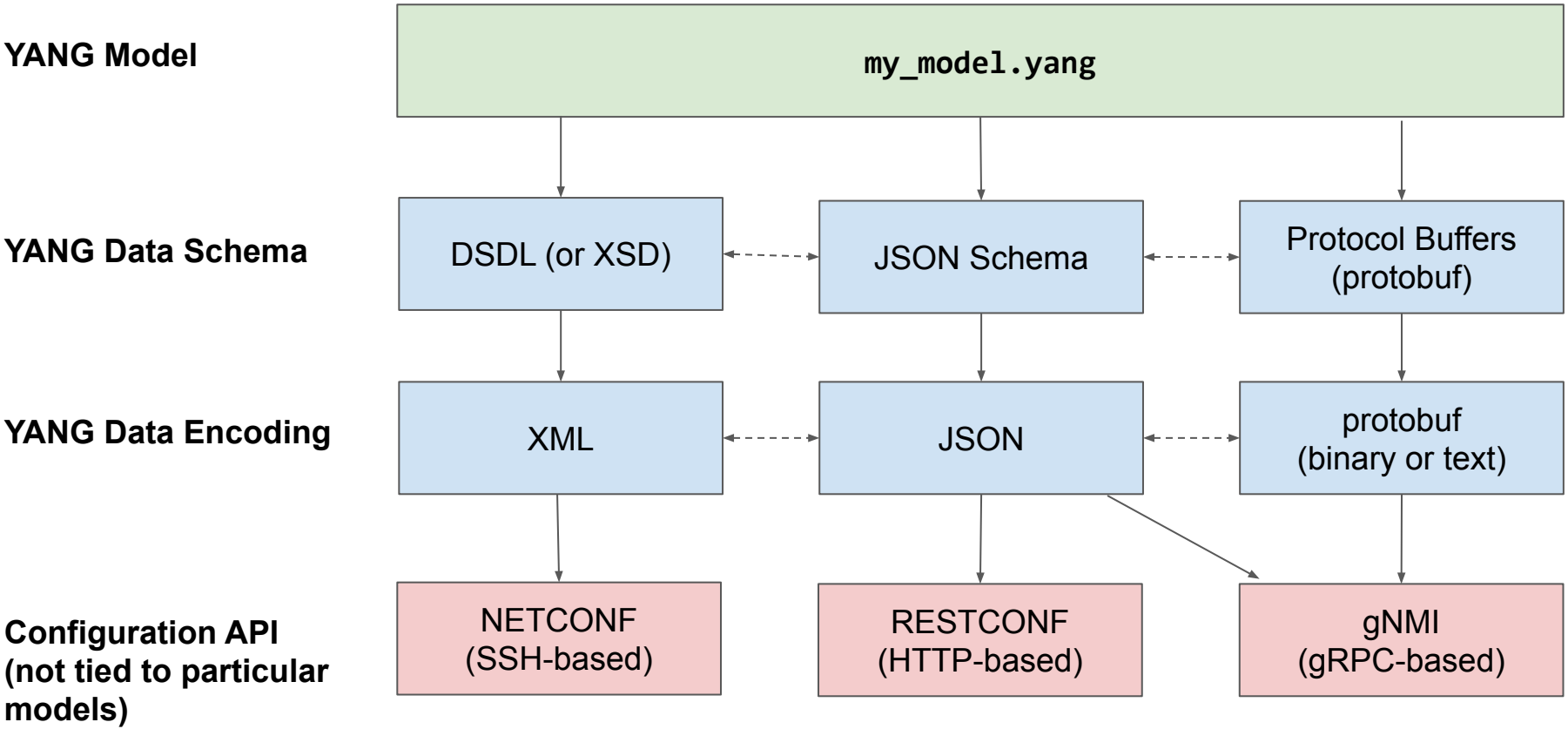
For a complete example, see:

https://github.com/opennetworkinglab/ngsdn-tutorial/blob/master/yang/demo-port.yang

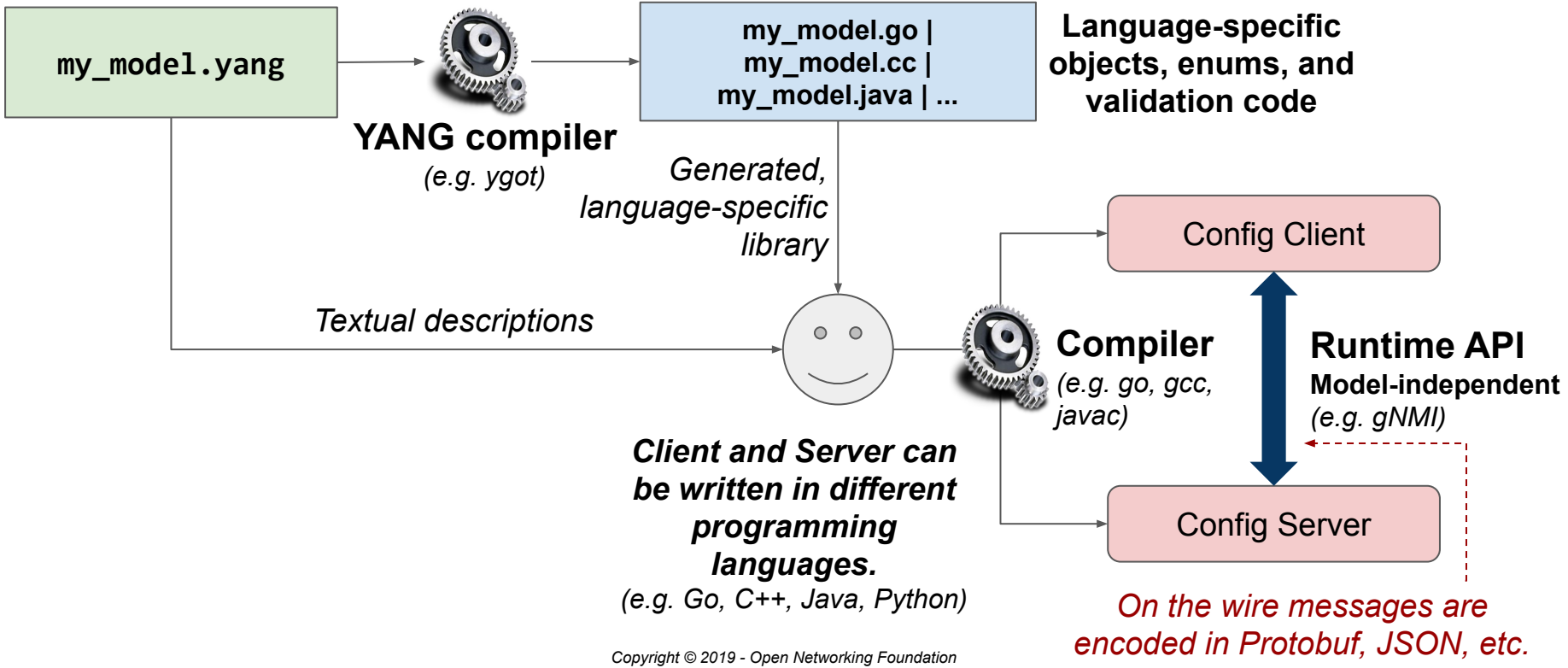
Other interesting terminology:

- **import** is used to include types, groupings, or other models
- **augment** is used to add to a previously defined schema
- **deviation** is used to indicate a device doesn't implement part of a schema

YANG Data Representations



YANG Workflow



OPENCONFIG

Vendor-neutral, model-driven network management designed by users

What is OpenConfig?

OpenConfig is an informal working group of network operators sharing the goal of moving our networks toward a more dynamic, programmable infrastructure by adopting software-defined networking principles such as declarative configuration and model-driven management and operations.

Common data models

Our initial focus in OpenConfig is on compiling a consistent set of vendor-neutral data models (written in YANG) based on actual operational needs from use cases and requirements from multiple network operators.

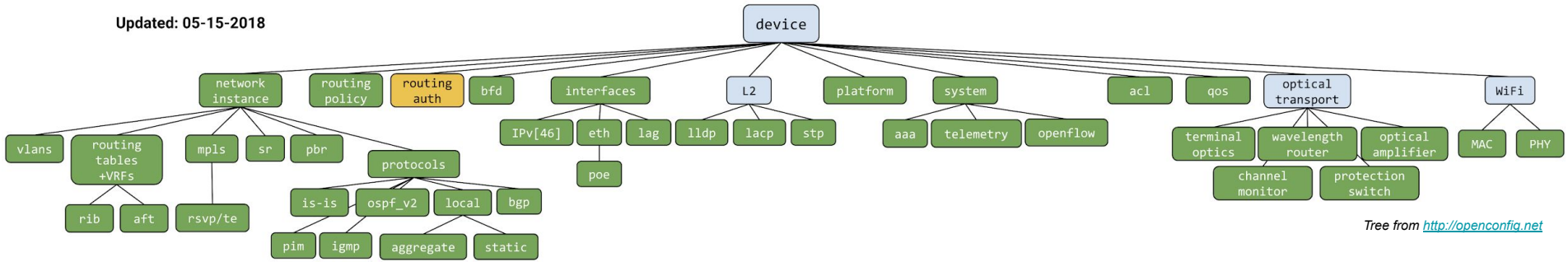
Streaming telemetry

Streaming telemetry is a new paradigm for network monitoring in which data is streamed from devices continuously with efficient, incremental updates. Operators can subscribe to the specific data items they need, using OpenConfig data models as the common interface.

OpenConfig Introduction

- Goal: Develop Vendor-Neutral Data Models for Configuration and Management that are supported natively by network hardware and software devices
- Goal: Develop modern and efficient transport protocols for networking configuration, telemetry, and operations (**gNMI** & **gNOI**)
- Represents a variety of network operators' use cases

Current models: https://github.com/openconfig/public



Tree from http://openconfig.net

OpenConfig Usage in Stratum

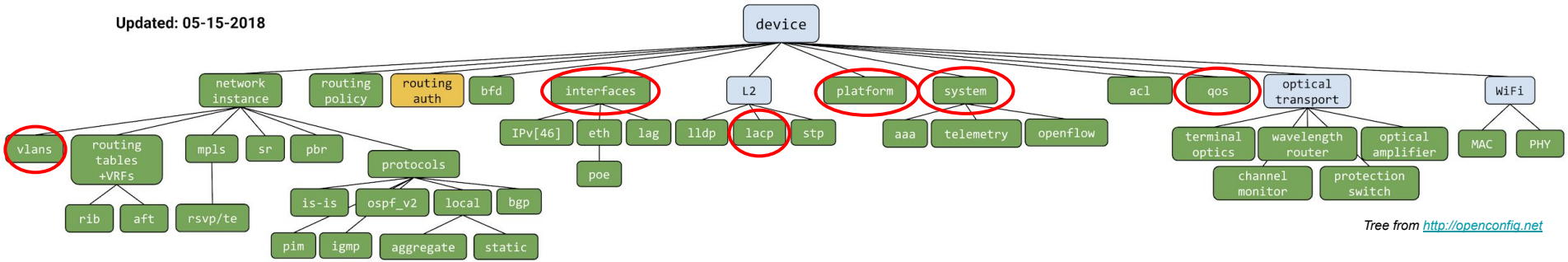
OpenConfig defines a lot of models, but only a subset are relevant to the data plane.

These are the models that Stratum is interested in:

interfaces*, *lacp*, *platform*, *qos*, *vlan*, *system

Stratum also uses some augmentations defined in **openconfig/hercules**

Vendors can also provide augmentations and deviations on top of this.



Tree from http://openconfig.net

openconfig-interfaces.yang

https://github.com/openconfig/public/blob/master/release/models/interfaces/openconfig-interfaces.yang

Tree representation generated by pyang

```
module: openconfig-interfaces
  +--rw interfaces
     +--rw interface* [name]
        +--rw name              -> ../config/name
        +--rw config
        |  +--rw name?            string
        |  +--rw type             identityref
        |  +--rw enabled?         Boolean
        |     ...
        +--ro state
        |  +--ro name?            string
        |  +--ro type             identityref
        |  +--rw enabled?         Boolean
        |  +--ro ifindex?         uint32
        |  +--ro admin-status     enumeration
        |  +--ro oper-status      enumeration
        |     ...
        |  +--ro counters
        |     +--ro in-octets?             oc-yang:counter64
        |     +--ro in-pkts?               oc-yang:counter64
        |     +--ro in-unicast-pkts?       oc-yang:counter64
        |     +--ro out-octets?            oc-yang:counter64
        |     +--ro out-pkts?              oc-yang:counter64
        |     +--ro out-unicast-pkts?      oc-yang:counter64
        |  ...
```

gNMI (gRPC Network Management Interface)

- Generic API to read and write configuration state
- Suitable for any tree-based data model
  - YANG as a possible data model
- Successor of NETCONF

	gNMI	NETCONF
Serialization	**Protobuf** Compact binary format	**XML**
Transport	**gRPC** (HTTP/2.0)	**SSH**
Diff oriented	**Yes** Returns only elements of the tree that have changed from last read	**No** Always returns entire sub-tree snapshot
Native support for streaming telemetry	**Yes** gRPC natively supports bidirectional streaming	**No** Needs YANG Push extension

gnmi.proto (gRPC service)

Provides an interface for:

- retrieving device capabilities (e.g., models supported)
- reading/writing configuration
- receiving streaming telemetry updates

```
service gNMI {
 rpc Capabilities(CapabilityRequest) returns (CapabilityResponse);
 rpc Get(GetRequest) returns (GetResponse);
 rpc Set(SetRequest) returns (SetResponse);
 rpc Subscribe(stream SubscribeRequest) returns (stream SubscribeResponse);
}
```

Protobuf Definition:
https://github.com/openconfig/gnmi/blob/master/proto/gnmi/gnmi.proto

Service Specification:
https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md

gNMI Get

```
message GetResponse {
 repeated Notification notification = 1;   // Data values.
 // Extension messages associated with the GetResponse. See the
 // gNMI extension specification for further definition.
 repeated gnmi_ext.Extension extension = 3;
}

message Notification {
 int64 timestamp = 1;          // Timestamp in nanoseconds since Epoch.
 Path prefix = 2;              // Prefix used for paths in the message.
 // An alias for the path specified in the prefix field.
 // Reference: gNMI Specification Section 2.4.2
 string alias = 3;
 repeated Update update = 4;   // Data elements that have changed values.
 repeated Path delete = 5;     // Data elements that have been deleted.
}

message Update {
 Path path = 1;                        // The path (key) for the update.
 TypedValue val = 3;                   // The explicitly typed update value.
 uint32 duplicates = 4;                // Number of coalesced duplicates.
}
```

gNMI Updates

- **Batches of configuration can be written or read using a list of Updates**
- **Updates consist of two parts:**
  - Relative path to data node
  - Value associated with node
- **Updates can transmit either:**
  - Configuration snapshots of a tree or sub-tree (encoded in Protobuf or JSON)
  - Leaf values only (encoded as `[path, value]` entries)
- **Get, Set, and Subscribe use the same messages to send or receive updates**

gNMI Updates using Snapshots

```
message Update {
 Path path = 1;                        // The path (key) for the update.
 TypedValue val = 3;                   // The explicitly typed update value.
 uint32 duplicates = 4;                // Number of coalesced duplicates.
}
message TypedValue {
 oneof value {
   // ... basic / scalar types ...
   google.protobuf.Any any_val = 9;  // protobuf.Any encoded bytes.
   bytes json_val = 10;              // JSON-encoded text.
   bytes json_ietf_val = 11;         // JSON-encoded text per RFC7951.
   string ascii_val = 12;            // Arbitrary ASCII text.
   // Protobuf binary encoded bytes. The message type is not included.
   // See the specification at
   // github.com/openconfig/reference/blob/master/rpc/gnmi/protobuf-vals.md
   // for a complete specification.
   bytes proto_bytes = 13;
 }
}
```

gNMI Get Example (OpenConfig Interfaces)

Response to Get: *`/interfaces/interface[1/0]`*

```
notification {
  update {
    path {
      elem { name: "interfaces" }
      elem {
        name: "interface"
        key { key: "name", value: "1/0" }
      }
    }
    val { any_val {
      type_url: "type.googleapis.com/openconfig.Interface"
      value: { // message openconfig.Interface
        enabled { value: true }
        ethernet { port_speed: SPEED_100GB }
        ifindex { value: 128 }
      }
    }}
  }
}
```

Actual message is binary encoded by protobuf

Note: Today, Stratum only supports returning Protobuf encoded data for ***get*** *requests on the root path (“/”).*

gNMI Subscribe

- **Client first submits a `SubscriptionList` containing the following:**
  - Path in the config tree
  - Subscription type (target defined, on change, sample)
  - Subscription mode (once, stream, poll)
- **Server immediately sends snapshot of current state for requested subscriptions (unless suppressed)**
- **Server sends updates per the subscription type and mode**
- **Schema-less [path, value] updates minimize data transmission by only supplying the leaf nodes that have changed (a diff-oriented approach)**

gNMI Updates using Schema-less Mode

The other half of Update's TypedValue:

```
message TypedValue {
 oneof value {
   string string_val = 1;            // String value.
   int64 int_val = 2;                // Integer value.
   uint64 uint_val = 3;              // Unsigned integer value.
   bool bool_val = 4;                // Bool value.
   bytes bytes_val = 5;              // Arbitrary byte sequence value.
   float float_val = 6;              // Floating point value.
   Decimal64 decimal_val = 7;        // Decimal64 encoded value.
   ScalarArray leaflist_val = 8;     // Mixed type scalar array value.
   // ...
 }
}
```

Note: The scalar or leaflist values require a path that identifies a leaf or leaflist node.

gNMI Get / Subscribe “Schema-less”

Response to Get or Subscribe: *`/interfaces/interface[1/0]`*

```
notification {
  timestamp: 100
  prefix {
    elem { name: "interfaces" }
    elem {
      name: "interface"
      key { key: "name", value: "1/0" }
    }
  }
  update {
    path { elem { name: "config" }, elem { name: "enabled" } }
    val { bool_val: true }
  }
  update {
    path { elem { name: "config" }, elem { name: "port-speed" } }
    val { string_val: "SPEED_100GB" }
  }
  update {
    path { elem { name: "state" }, elem { name: "ifindex" } }
    val { uint_val: 128 }
  }
}
```

Actual message is binary encoded by protobuf

Device can elect to use schema-less mode when responding. Stratum does so for non-root paths.

Note: Multiple notifications can be used to avoid path duplication. Today, Stratum doesn’t use use prefixes to optimize paths.

gNMI Set

- **The client can perform three types of Set in a single request:**
  - Remove a node from the tree
  - Replace a node (completely overwrite)
  - Update a node (only update values explicitly included in message)

```
message SetRequest {
 Path prefix = 1;                // Prefix used for paths in the message.
 repeated Path delete = 2;       // Paths to be deleted from the data tree.
 repeated Update replace = 3;    // Updates specifying elements to be replaced.
 repeated Update update = 4;     // Updates specifying elements to updated.
 // Extension messages associated with the SetRequest. See the
 // gNMI extension specification for further definition.
 repeated gnmi_ext.Extension extension = 5;
}
```

gNMI Capabilities

Allows to client to ask a device for the device's gNMI version, set of supported YANG models, supported encodings, and extensions.

```
message CapabilityResponse {
 repeated ModelData supported_models = 1;
 repeated Encoding supported_encodings = 2;
 string gNMI_version = 3;
 repeated gnmi_ext.Extension extension = 4;
}
message ModelData {
 string name = 1;
 string organization = 2;
 string version = 3;
}
```

Example response:

```
supported_models {
  name: "openconfig-interfaces"
  organization: "OpenConfig working group"
  version: "2.4.1"
}
supported_models {
  name: "openconfig-platform"
  organization: "OpenConfig working group"
  version: "0.12.2"
}
...
supported_encodings: PROTO
gNMI_version: "0.7.0"
```

gNOI Overview

- **Collection of micro-services for operations / runtime management, for example:**
  - Device reboots, pushing/rotating SSL keys/certs, BERT (bit error rate testing on a link/port), ping testing
  - Ephemeral state management (clearing L2 neighbor discovery/spanning tree, resetting a BGP neighbor session)
- **Each service contains multiple RPCs, and is modeled in Protobuf (similar to NETCONF/YANG rpc)**
- **Allow some control of legacy network protocols (not used in Stratum)**
  - E.g., spanning tree, MPLS LSPs, BGP
- **Stratum has initial support for cert, file, diag, system**

https://github.com/openconfig/gnoi

Example: gNOI System Service

```
service System {
 rpc Ping(PingRequest) returns (stream PingResponse) {}
 rpc Traceroute(TracerouteRequest) returns (stream TracerouteResponse) {}
 rpc Time(TimeRequest) returns (TimeResponse) {}
 rpc SetPackage(stream SetPackageRequest) returns (SetPackageResponse) {}
 rpc Reboot(RebootRequest) returns (RebootResponse) {}
 // ...
}

message RebootRequest {
 RebootMethod method = 1; // COLD, POWERDOWN, HALT, WARM, NSF, ...
 uint64 delay = 2; // Delay in nanoseconds before issuing reboot.
  string message = 3; // Informational reason for the reboot.
  repeated types.Path subcomponents = 4; // Optional sub-components to reboot.
  bool force = 5; // Force reboot if sanity checks fail. (ex. uncommited configuration)
}
```

https://github.com/openconfig/gnoi/blob/master/system/system.proto

Exercise 2 overview

Exercise 2: Goals and Steps

1. **Understanding the YANG language**
   a. Look at a simple model (**demo-port.yang**) and a more complicated set of models (subset of OpenConfig YANG models)
   b. Use **pyang**, a YANG compiler framework, to visualize the models
2. **Understand YANG encoding**
   a. Use pyang to generate an XML Schema (DSDL) and skeleton XML model
   b. Use ygot's **proto_generator**, another YANG compiler, to generate Protobuf messages based on the models
3. **Understanding YANG-enabled transport protocols (using gNMI)**
   a. Use **gnmi-cli**, a simple gNMI client, to get, set, and subscribe to configuration from a Stratum BMv2 switch

Exercise 2: Tools

Docker container 1: `bocon/yang-tools:latest`

- Start with **make yang-tools**
- Contains pyang, proto_generator, the OpenConfig models, and a OpenConfig Protobuf binary decoder utility

Docker container 2: `opennetworking/mn-stratum:latest`

- We'll use the same Mininet setup as Exercise 1, and you can leave it running
- This is where Stratum BMv2 is running; only used for part 3

Docker container 3: `bocon/gnmi-cli:latest`

- Start with **util/gnmi-cli <args>**
- Contains the simple gNMI client used for part 3

Exercise 2: Get Started

These slides:
http://bit.ly/ngsdn-tutorial-slides

Open lab README on GitHub:

http://bit.ly/ngsdn-tutorial-lab

Or open in text editor:

`~/ngsdn-tutorial/README.md`

`~/ngsdn-tutorial/EXERCISE-2.md`

Before starting!
Update tutorial repo
(requires Internet access)

```
cd ~/ngsdn-tutorial
git pull origin master
make pull-deps
```

You can work on your own using the instructions.
You have time until 12:30pm. Lunch is 12:30-1:30pm.