Mawlana Bhashani Science and Technology University

Santosh, Tangail-1902

Lab Report

Report No : 03

Report Title : Controller Rest API

Course Title : Telecommunication Engineering

Submitted By	Submitted To
Name: **Student ID: IT-17001** **Session: 2016-17** Dept. of Information & Communication Technology MBSTU, Tangail – 1902	**Nazrul Islam** **Assistant Professor** Dept. of Information & Communication Technology MBSTU, Tangail – 1902

Theory:

OpenFlow: Reactive versus Proactive

OpenFlow is still the only one wire protocol that has a reasonably good chance at becoming the de-facto open SDN southbound messaging standard. When using OpenFlow to populate tables in switches there are essentially three modes of operation:

- **Reactive Flow Instantiation** : When a new flow comes into the switch, the OpenFlow agent software on the switch does a lookup in the flow tables. If no match for the flow is found, switch creates an OFP packet-in packet and sends it off to the controlle for instructions. Reactive mode reacts to traffic, consults OpenFlow controller and creates a rule in the flow table based on the instruction. This behavior was tested on previous lab.
- **Proactive Flow Instantiation** : Rather than reacting to a packet, an OpenFlow controller could populate the flow tables ahead of time for all traffic matches that could come into the switch. By pre-defining all of the flows and actions ahead of time in the switches flow tables, the packet-in event never occurs. The result is all packets are forwarded at line rate. Proactive OpenFlow flow tables eliminate any latency induced by consulting a controller on every flow. This behavior will be tested on this lab.
- **Hybrid flow instantiation** : A combination of both would allow for flexibility of reactive for particular sets a granular traffic control that while still preserving low-latency forwarding for the rest of the traffic.

Controller: REST API

- **Application program interface (API)** is an interface presented by

software (such as a network operating system) that provides the capability to collect information from or make a change to an underlying set of resources.

 APIs in the context of SDN: In an open SDN model, a common interface discussed is the northbound interface (NBI). The NBI is the interface between software applications, such as operational support systems, and a centralized SDN controller. One of the common API technologies used at the northbound interface is the Representational State Transfer (REST) API. REST APIs use the HTTP/HTTPS protocol to execute common operations on resources represented by Uniform Resource Identifier (URI) strings. An application may use REST APIs to send an HTTP/HTTPS GET message via an SDN controller's IP address. That message would contain a URI string referencing the relevant network device and comprising an HTTP payload with a JSON header that has the proper parameters for a particular interface and statistic.

 Datapath Identifier of Openflow Switch: Each OpenFlow instance on a switch is identified by a Datapath Identifier. This is a 64 bit number determined as follows according to the OpenFlow specification: “The datapath_id fleld uniquely identifies a datapath. The lower 48 bits are intended for the switch MAC address, while the top 16 bits are up to the implementer. An example use of the top 16 bits would be a VLAN ID to distinguish multiple virtual switch instances on a single physical switch.”

5. Questions

1. Explain the advantages of REST API of the Controller.

Ans:

One of the most popular types of API is REST or, as they’re sometimes known, RESTful APIs. REST or RESTful APIs were designed to take advantage of existing protocols. While REST - or Representational State Transfer - can be used over nearly any protocol, when used for web APIs it typically takes advantage of HTTP. This means that developers have no need to install additional software or libraries when creating a REST API.

One of the key advantages of REST APIs is that they provide a great deal of flexibility. Data is not tied to resources or methods, so REST can handle multiple types of calls, return different data formats and even change structurally with the correct implementation of hypermedia. This flexibility allows developers to build an API that meets your needs while also meeting the needs of very diverse customers.

There are 6 key constraints to think about when considering whether a RESTful API is the right type of API for your needs:

Client-Server: This constraint operates on the concept that the client and the server should be separate from each other and allowed to evolve individually.

Stateless: REST APIs are stateless, meaning that calls can be made independently of one another, and each call contains all of the data necessary to complete itself successfully.

Cache: Because a stateless API can increase request overhead by handling large loads of incoming and outbound calls, a REST API should be designed to encourage the storage of cacheable data.

Uniform Interface: The key to the decoupling client from server is having a uniform interface that allows independent evolution of the application without having the application's services, or models and actions, tightly coupled to the API layer itself.

Layered System: REST APIs have different layers of their architecture working together to build a hierarchy that helps create a more scalable and modular application.

Code on Demand: Code on Demand allows for code or applets to be transmitted via the API for use within the application.

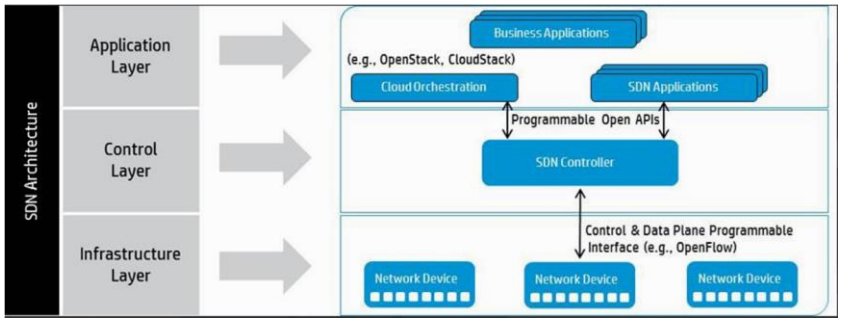
Unlike SOAP, REST is not constrained to XML, but instead can return XML, JSON, YAML or any other format depending on what the client

requests. And unlike RPC, users aren't required to know procedure names or specific parameters in a specific order.One of the disadvantages of RESTful APIs is that you can lose the ability to maintain state in REST, such as within sessions. It can also be more difficult for newer developers to use.It's important to understand what makes a REST API RESTful, and why these constraints exist before building your API.

Question 2: **What is the difference between sdn and openflow?**

Ans:-

Software defined networking architecture separates control plane (controller) and data plane (switches). The most fundamental rule of SDN architecture is:- Data plane is directly programmable by controller and centrally managed. Controller provide central view of the network and also allows switches to be directly controllable via network application written on top of Controller. This kind of abstraction is done via Northbound API. The Southbound API provides interaction betweencontroller and switches.



Now lets come to Openflow.

Openflow is a forwarding protocol, which is being used for the interaction between data plane and control plane. Any network is said to be openflow network only if the switches and controller in the network supports openflow.

SDN and OpenFlow are prone to be confused and misunderstood. Take a look at SDN vs. OpenFlow, the two are indeed interconnected. First of all, as an open protocol, OpenFlow underpins the various SDN controller solutions. The complete SDN solution is taking SDN controller as the core, backed by OpenFlow switches and NFV to offer bountiful SDN app for a new smart, dynamic, open, custom network

- **Conclusion**

Even though REpresentational State Transfer, also known as REST, is often referred to as a protocol, it's an architectural style. It defines how applications communicate over the Hypertext Transfer Protocol (HTTP). Applications that use REST are loosely-coupled and transfer information quickly and efficiently. While REST doesn't define data formats, it's usually associated with exchanging JSON or XML documents between a client and a server. This interface overcomes the disadvantages **SOAP** exhibited, such as the need for clients to know the operation semantics as a pre-requisite for its use, or the need for ports for different types of notifications. In addition, with a few operations, **REST** can handle many resources, while **SOAP** needs many operations to accomplish that.
These are its advantages:

- It is usually simple to build and adapt.
- Low use of resources.
- Process instances are created explicitly.
- With the initial URI, the client does not require routing information.
- Clients can have a generic 'listener' interface for notifications