# 4.1t Transcript

**What are Software Defined Networks?**

Software Defined Networking is defined as: ‘The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices.’ Fundamentally, SDN decouples the data and control planes with centralised control of forwarding decisions and enabling a holistic view of the network to SDN applications. The following table describes the high-level purpose of each plane of SDN.

The layers, as defined in shown table, operate over an open programmable interface, or more commonly known as the API. The southbound interface API/connector provides an interface between the SDN controller and switches, and thus enables the separation of control from the data plane below the southbound API. The northbound area of the controller enables applications to interact with the controller, such as firewalls, obtaining network state or statistics, whilst the east and westbound controllers facilitate the communication between multiple SDN controller clusters.

The Open Network Foundation (ONF) outline numerous limitations with current network infrastructures. Network protocols have been designed for performance, reliability and security; however, standardisation of these protocols have faced major scrutiny and, in many cases, thwarted innovation. Network administrators are required to fully understand operational knowledge of such protocols. Additionally, deployment of services across a wide range of vendors and equipment potentially results in inconsistency of network-based policies. Therefore, conventional network infrastructures lack flexibility, often cumbersome to manage and are highly reliant on closed-source vendor specific hardware.

This lack of flexibility is clearly evident in traditional network infrastructures. Packet routing decisions are made on the basis of a single piece of hardware, where the control and data plane co-exist within the same environment and within a containerised and proprietary platform. Software Defined Networks, however, split the control and data plane in to separate entities and facilitate development of new software modules, using high level programming languages through the use of Application Programming Interfaces (APIs). These modules can influence network functionality in various ways according to the requirements of the network. The SDN controller is responsible for control plane aspects of the network, making forwarding decisions and installing entries in to the flow table based upon flow matching rules.

**Decoupling Control Logic from the Data Plane**

As networks grew in the middle of the 1990’s, hardware processing power had to meet increasing traffic demands. This occurred primarily by designing hardware functions dedicated to packet switching that otherwise would have occurred in software, and thus Application Specific Integrated Circuits (ASIC) were developed to achieve this goal. The end result was the tight integration of the data/forwarding plane with the control plane. Incidentally, this would adversely affect Active Networking, as it relied on the ability to inspect flows per packet, which was not possible with proprietary ASICs. Subsequently, this would limit the scope of active networking until advancements in server hardware led to idea of the separation of control and data planes. The following demonstrates how conventional first hop routing architecture

**OpenFlow Switches and Flow Tables**

OpenFlow switches operate on a basis of ‘flows’. Flows are described as packets that share commonality in the packet header, for example the source/destination of the packet. These packets are matched according to the flows that are currently installed within the flow table. An OpenFlow switch may have numerous flow tables installed that operate in a pipeline manner. Each flow table contains the following components:

Any packets that do not match are defined through table-miss parameters that instruct how unmatched packets are processed – the packet, in this instance is forwarded to the controller for analysis.

These features are mandatory and must be supported within the OpenFlow protocol. There are, however, numerous fields that are optional such as designated ports, VLAN IDs, ICMPv4 and v6 and ARP, as well as numerous other options.

Flow Tables are efficient and are highly flexible as generalised entries at the top of the pipeline are tested first, whilst more traffic class specific information is tested lower down in the pipeline. Additionally, flow tables can support network functions such as NAT. Many implementations vary, however there is commonality shared amongst OpenFlow supported devices, with the Southbound API supporting existing infrastructural implementations. OpenFlow supported switches consist of three elements of communication – the OpenFlow protocol that operates a secure channel between the OpenFlow channel and the channel controller, alongside pipelined flow tables.

**OpenFlow Protocol**

OpenFlow is the open standard communications protocol for SDN networks managed by the Open Networking Foundation. It acts as the central intelligence towards relaying forwarding information to switches and routers via the southbound API and the applications

operating above it, known as the northbound API. In essence, OpenFlow opens access to the forwarding plane of a switch or router on a network. The OpenFlow protocol has undergone numerous revisions since its inception in 2008. The protocol specification defines how the SDN controller communicates with an OpenFlow compatible switch and enables the SDN controller to manipulate flow tables. Switches that use OpenFlow comprise of three elements, a) The OpenFlow protocol, b) A flow table and c) A secure channel established between the switch and the controller.

It is the responsibility of the Open Networking Foundation to maintain the OpenFlow protocol, with many revisions since its inception in 2008. The definitions of the OpenFlow message types are defined by publicly available documentation on the Open Networking Foundation and are subject to constant change as the OpenFlow protocol evolves. At the time of this writing, the latest version of the OpenFlow protocol is Version 1.5.1.**SDN Controllers**

There is now a considerable selection of SDN controllers, some of these recent solutions have been developed to address the scalability and redundancy issues surrounding centralised SDN controllers, new SDN distributions are now offering physically distributed, however logically centralised control plane offers a potentially greater. This enables many controllers to share the load between numerous OpenFlow switches and thus increase performance and also resiliency of the SDN based network.

For the sake of this presentation, we will use NOX/POX controller in particular to demonstrate how an SDN controller may work. Typically, traditional network infrastructures employ highly proprietary low-level operating systems that do not provide any form of abstraction layer towards physical resources. Nox is presented as a network operating system that offers a centralised API for controlling network resources, using high level abstractions such as usernames and hostnames rather than conventional MAC and IP addressing that have assumed to be the status quo in networks seen today. The operating system operates on the basis of holding an entire picture of the network and using Djikstra to compute the shortest path. It is important to stress that a network operating system does not provide functionality of network management but offers APIs to which applications perform management tasks. NOX utilises the OpenFlow protocol in order to control input and output processes of the network hardware. Event handlers built in to the network operating system use flow arrival detection and based upon the network state, proceeds to install or modify flow rules within flow tables of the switch. After a flow has been installed in the table, the SDN controller is subsequently no longer required that particular flow. A high-level overview of this process can be seen in the following diagram.

NOX, has numerous applications which have been suggested to control various functionality such as switching at Layer 2, network wide views and 802.1x based authentication. Numerous proposals for NOX have been suggested in datacentre environments, by modelling popular architectures such as VL2 and Portland.

The Network Operating System has resulted in the development of numerous SDN controllers such as OpenDayLight, Kandoo amongst many other centralised and distributed controllers. These controllers have developed to offer an extensive feature set and a diverse selection of network-based APIs.

**Simulation of SDN Networks**

Numerous open source tools exist for the purpose of simulation and measuring the performance characteristics of various SDN topology configurations using a variety of different SDN controllers. The subsequent sections provide a summary of these tools and their functionality.

The aim of Mininet is to create a fully operational, realistic network that runs a real-time kernel, switch and application code. This allows for the deployment of the OpenFlow protocol and Open vSwitches, with the virtualisation of hosts, switches and the controllers that communicate with them. In essence, this simulation environment mimics a real-life network and provides real world results towards a given SDN based network topology without the added overheads of running such systems on a fully virtualised machine environment, which adds complexity and, in many cases, unnecessary overheads.

Mininet is popular due to its open source nature and diverse API set that can be adapted to many situations. It also operates using the Python programming language, which is a high level and intuitive language for this purpose. The scripts determine the depth of hosts and switches in the tree, this can be altered to suit the testing environment.

The following image provides a simple demonstration of a topology that is used using multiple Mininet servers that are tunnelled to peer with each other. Each Mininet server acts as an SDN controller. A simple python script can be written to instruct Mininet how this topology will be built

Simple python scripts can be written to instruct Mininet how this topology will be built. This python script essentially replicates the topology as shown in the image that was shown earlier. It can also be represented in a more simplistic manner if necessary.

**Open vSwitch**

Open vSwitch is an open source Linux based multi-layer virtual switching software designed for virtualised environments. It can be used both as switch within a hypervisor or can act as a control stack to interface directly with silicon. Open vSwitch provides an efficient means for multi-server virtualisation deployments to bridge traffic between VMs and the outside world. Traditionally, Linux hosts have provided L2 bridging, however these stacks are not well suited for this environment. Many virtualised environments are often defined as being highly dynamic end-points, with the maintenance of logical abstractions, and, in many cases, the integration with or offloading to special purpose switching hardware. More relevantly, open vSwitch supports flow forwarding and fully supports the OpenFlow protocol. This enables a cost-effective way of utilising software-based switching rather than dedicated OpenFlow supported switches.