Chapter 1

SDN Review

2020-11-02

## Network device evolution

Since the early 1990s network device manufacturers have made a lot of innovation in order to increase router speeds. They started from a router node in which everything was computed by the central CPU to a situation where the central CPU is used less and less due to a distributed architecture in which lots of action is done by line cards.

![image](data:image/emf;base64;base64,)

Figure 1.1 Network Device Evolution

This progress has been made thanks to the use of proprietary TCAM (Ternary Content-Addressable Memory) and ASICs (Application-Specific Integrated Circuit) which have been designed to perform table look up and data packet forwarding at extreme high speeds.

In early 2000, the virtualization of x86 computers led to lots of innovation in systems domain. Compute virtualization and high-speed network devices evolution has enabled the network cloud creation.

Since it isn’t convenient to manage several isolated network devices, each having their own configuration language, the following needs have emerged:

* Single point of configuration
* Configuration protocol standardization
* Network feature support on x86 servers
* Extensibility and ability to scale
* Good performance

Which called for more cloud and SDN technology development.

## Early Age of SDN

In Stanford University (US - CA), Clean Slate Research Projects program was initiated in 2006 in order to think about how to improve the internet’s network architecture. The Ethane project was part of this program. Its purpose was to Design networks where connectivity is governed by high-level, global policy. This project is generally known as the first implementation of SDN.

In 2008, a white paper was proposed by ACM (Association for Computing Machinery) to design a new protocol (OpenFlow) that can program network devices from a network controller.

In 2011, ONF (Open Networking Foundation) was created to promote SDN Architecture and OpenFlow protocols.

## SDN startups acquired by major networks or virtualization vendors

The first companies focusing on SDN were founded around 2010. (Most of them have now been acquired by main networks or virtualization solution vendors.) In 2007, Martin Casado, who was working on Ethane project founded Nicira to provide solutions for network virtualization with SDN concepts. Nicira was aquired by vMware in 2012 to develop VMWare NSX. In 2016, VMWare also bought PLUMGrid, a SDN startup founded in 2013. In 2010, BigSwitch networks, which was proposing an SDN solution, was founded. In early 2020, BigSwitch was acquired by Arista Networks. In 2012, Cisco created Insieme Networks, a spin-off start-up company working on SDN. In 2013, Cisco took back control of Insieme in order to develop its own SDN solution called ACI (Application Centric Infrastructure). In early 2012, Contrail Systems was created and aquired at the end of the year by Juniper Networks. In 2013, Alcatel Lucent created Nuage Networks, a spin-off start-up company working on SDN, which is now an affiliate of Nokia.

The road of SDN development and its history is never straightforward and looks more nuanced than a single storyline might suggest. It’s actually far more complex to be described in a short section. Figure 2.1 from [[sdn-history]](#sdn-history) shows developments in programmable networking over the past 20 years, and their chronological relationship to advances in network virtualization.

![sdn-history](data:image/png;base64;base64,)

## So What is SDN?

The concept of SDN, and the term itself, is both broad and often confusing. There is no real accurate definition of SDN, and vendors usually take it very differently. Initially it was used to describe Stanford’s OpenFlow project, but today it has been extended to include a much wider swath of technologies. Discussion about each vendor’s exact SDN definition is beyond the scope of this book, but in general consider that a SDN solution has to provide one to several of following characteristics:

* A network control and configuration plane split from the network data plane.
* A centralized configuration and control plane (SDN controller)
* A simplified network node
* Network programmability to provide network automation
* Automatic provisioning (ZTP zero touch provisioning) of network nodes
* Virtualization support and openness

According to [[onf-sdn-definition]](#onf-sdn-definition), **Software-Defined Networking (SDN)** is:

The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices

**SDN layer**[**[onf-sdn-definition]**](#onf-sdn-definition)**.**

![image](data:image/png;base64;base64,)

Figure 1.3 What is SDN?

You can see in Figure 1.3, that SDN allows simple high-level policies in the application layer to modify the network, because the device level dependency is eliminated to some extent. The network administrator can operate the different vendor-specific devices in the infrastructure layer from a single software console – the control layer. The controller in the control layer is designed in such a way that it can globally view the whole network. This controller design helps to introduce functionalities or programs, since the applications just need to talk to the centralized controller without needing to know all the details communicating with each individual device. These details are hidden by the controller from the applications.

Several traits fit this new model:

* Openness: Communication between controller and network device uses standardized protocols like REST, OpenFlow, XMPP, NetConf, gRPC, etc. This eliminates traditional vendor lock-in, giving you freedom of choice in networking.
* Cost reduction: Due to the open model, users can pick any low-cost vendor for their infrastructure (hardware).
* Automation: The controller layer has a global view of whole network. With the APIs exposed by the control layer, automation of applications becomes much easier.

In Figure 3.1, OpenFlow is labeled as the protocol between the control and infrastructure layers. This is just an example showing the use of standard communication protocols. As of today more choices of communication protocols are available and the standardized in the SDN industry, which will be covered later in this chapter.

## Traditional Network Planes and SDN layer

Traditionally, a typical network device (for example, a router) has the following planes shown in Figure 1.4.

![image](data:image/emf;base64;base64,)

Figure 1.4 A typical modular router

The figure illustrates the following:

* Configuration (and management) plane: This is used for network node configuration and supervision. Examples of widely use protocols are CLI (Command Line Interface), SNMP (Simple Network Management Protocol) and NetConf.
* Control plane: This is used by network nodes to make packet forwarding decisions. In traditional networks there have been a wide range of different network control protocols: OSPF, ISIS, BGP, LDP, RSVP-TE, etc.
* Forwarding (or data) plane: This plane is responsible for performing data packet processing and forwarding. This forwarding plane is made of proprietary protocols and is specific to each network equipment vendor.

Configuration and Control planes are located in the device’s main processor card, often called the routing engine (RE) or routing switching engine. The forwarding plane is located in the device’s packet forwarding card, often called the line card.

SDN architecture typically has three layers as shown in Figure 1.5.

![image](data:image/emf;base64;base64,)

Figure 1.5 SDN Architecture

The figure illustrates SDN architecture layers:

* Application Layer: This layer contains all the applications provided by the SDN solution. Generally a Web GUI dashboard is the first application provided to SDN users. Other common applications are network infrastructure interconnection interfaces allowing the SDN solution to be plugged into a cloud infrastructure or a container orchestrator.
* Control Layer: The layer containing the SDN controller. This is the most intelligent part of a SDN solution and has a global view of the whole network. The SDN controller is made up of:
  + The SDN engine, which contains SDN control logic and databases to store the state and configuration of the network.
  + Southbound interfaces, which are used to communicate with the SDN network nodes. Some of the most commonly used southbound interface protocols are OpenFlow, XMPP, and OVSDB.
  + Northbound interfaces, which are used to expose services provided by the infrastructure layer ‘upwards’ to the SDN applications. The most commonly used northbound interface protocol is HTTP/REST.
* Infrastructure Layer: This layer consists of the SDN network nodes. This is the work horse of a SDN solution, and SDN network nodes can be either physical or virtual. Typically, each SDN node is composed of:
  + A SDN agent: which is handling the communication between each SDN network node and the SDN controller.
  + A flow/routing table built by the SDN Agent.
  + A forwarding plane engine

## Primary changes between SDN and traditional networking

In a traditional infrastructure, the route calculation is made on each individual router. Each router needs to run one or several routing protocols, through which it exchanges routes with the rest routers in the network, and eventually, based on the route information learned, each router assumes it gains enough knowledge about the network in order to make the forwarding decision. From the network perspective, the control plane is distributed in each individual router, and the end to end routing path is the result of all decisions made by the control plane located on each router.

The control plane on a router might look like Figure 1.6.

![image](data:image/emf;base64;base64,)

Figure 1.6 Components in a traditional router.

For example, a simplified Juniper MX Series control plane typical looks like the one illustrated in Figure 1.7.

![image](data:image/png;base64;base64,)

Figure 1.7 Typical Control Plane in a Sample MX Series Router

Running a control plane on each router makes it very hard to manage, because each individual network device needs to be carefully configured. It requires extensive, vendor-specific experiences and skills to configure the device. The high number of configuration points can make it challenging to build a robust network. Flexibility is also a recurring hurdle for traditional networks since most routers run proprietary hardware and software.

In contrast, in SDN networking, control and configuration functions are gathered into a SDN controller which is controlling network devices. The new architecture intends to provide a completely new way to configure the network. This new cloud infrastructure brings:

* simplified routers, without complex control planes in each router.
* a centralized control plane, which is a single configuration point

Let’s compare the two architectures as shown in Figure 1.8.

![image](data:image/emf;base64;base64,)

Figure 1.8 Comparison Between Tradition Network Devices and SDN Devices

You can see that the SDN infrastructure uses a centralized configuration and control point. Route calculation is done centrally in the controller and distributed into each SDN network node. While the idea looks simple, it requires a few fundamental protocols and infrastructures to be implemented before this model can work:

* A southbound network protocol: This is needed to allow routing information being exchanged between the SDN controller and each controlled element.
* A underlay network: This is a network infrastructure which allows the communication between SDN controller and SDN network nodes, and also the communication between SDN nodes themselves.

The underlay network infrastructure is playing the same role that the local switch fabric is doing inside a standalone router between the control processor card and lines cards. Based on it, an overlay network can be built by the controller, which basically hides underlay network infrastructure details from the applications so they will focus on the high level service implementations. We’ll discuss underlay and overlay more in the next section.

This model also makes the controller the weakest point. Think of what will happen if this SDN controller, serving as the brain, stops working. Everything will be frozen and nothing works as expected, or even worse, some part of the infrastructure continues to run but in an unexpected way, which will very likely trigger bigger issues to other parts of the network.

Lots of effort has been done by each SDN solution supplier to solve this weakness. A common and efficient practice is to use clustered architecture to build a highly resilient controller cluster. For example, three SDN controllers can load balance and/or backup each other. On failure of one or two, the other one can still make the whole cluster survive, giving the operator longer maintenance windows to fix the problem.

## Underlay Versus Overlay

In SDN architecture, each network node is connected to a physical network infrastructure. This physical network which is providing basic connectivity between network nodes is called the underlay network infrastructure. Sometimes it is also called fabric, and typically it’s a plain L3 IP network.

Very often, the underlay needs to separate between different administrative domains (called tenants), switch within the same L2 broadcast domain, route between L2 broadcast domains, provide IP separation via VRFs, and more. This is implemented in the form of an overlay network. The overlay network is a logical network that runs on top of the underlay network. The overlay is formed with tunnels to carry the traffic across the L3 fabric.

**Why the Need Overlay Networks?**

Today the industry is moving towards building L3 data centers and L3 infrastructures, mostly due to the rich features coming from L3 technologies, for example ECMP load balancing, flooding control, etc. But the L2 traffic does not disappear and most likely it never will. There is always the desire that a group of network users need to reside in the same L2 network - typically a VLAN. However, In today’s virtualization environment, a user’s VM can be spawned in any compute located anywhere in the L3 cluster. Even if two VMs are spawned in the same server, there is often a need to move them around between different servers without changing their networking attributes. These requirements to make a VM always belonging to the same VLAN calls for an overlay model over the L3 network. In other words, you need a new mechanism to allow you to tunnel L2 Ethernet domains with different encapsulations over an L3 network.

For example, let’s assume that in a SDN node, node1, you were running VM11 and VM12, both serving the same sales department and so they were located in the same VLAN. Because of some administrative requirement, VM12 needs to be moved to another physical SDN node2 which, may be physically located in another rack that is a few router hops away. Now, we need to ensure that not only data packets from VM11 in SDN node1 to be able to reach VM12 in SDN node2, but also they are talking to each other as if they are still in the same VLAN, exactly the same way as before as if VM12 has never moved. This ability to make the local (in same VLAN) traffic to traverse transparently across underlay network infrastructure calls for packet encapsulation, or tunneling mechanisms in SDN networks (see Figure 1.9).

![image](data:image/emf;base64;base64,)

Figure 1.9 Overlay Tunnels and Encapsulations

Indeed, without such an encapsulation mechanisms, traditional segmentation solutions (VLAN, VRF) would have to be provided by the physical infrastructure and implemented up to each SDN node in order to provide an isolated transportation channel for each customer network connected to the SDN infrastructure.

Tunneling protocols used in SDN networks have to provide at least the following capabilities:

* The ability to build several different network connectivity between two SDN network nodes. This is called network segmentation.
* The ability to carry Ethernet frames and IP packets transparently
* The ability to be carried over IP connectivity

Today, several tunneling protocols are used into SDN networks:

* VxLAN
* MPLS over GRE
* MPLS over UDP
* NVGRE
* Geneve
* STT

![image](data:image/emf;base64;base64,)

Figure 1.10 Overlay networks implemented by tunneling protocols

These tunneling protocols are providing Overlay connectivity which is required between customer workloads connected to the SDN infrastructure.

TIP In VxLAN, specifically, Each SDN node is called a VTEP (Virtual Tunnel End Point) as it is starting and terminating the overlay tunnels.

## Interfaces Between Layers

Let’s elaborate on the concepts of southbound and northbound interface and some available choices in today’s industry.

**Southbound interface**

The southbound interface resides between the controller in the control layer and network devices in the infrastructure layer. Basically what it does is to provide a mean of communication between the two layers. Based on demands and needs, a SDN Controller will dynamically change the configuration or routing information of network devices. For example, a new VM will advertise a new subnet or host routes when it is spawned in a server, and this advertisement will be delivered to an SDN controller via a southbound protocol. Accordingly, the SDN controller collects all the routing updates from the whole SDN cluster through the southbound interfaces, and decides the most current and best route entries, then, it may reflect this information to all the other network devices or VMs. This ensures all devices has the most up-to-date routing information in real time. Among others, examples of the most well-known southbound interfaces in the industry are openflow, OVSDB, gRPC and XMPP. Among them, openflow and OVSDB perhaps are the most well-known southbound interfaces. We’ll briefly introduce them.

**OpenFlow**

OpenFlow is a protocol that sends flow information into the virtual switch so the switch can forward the packets between the different ports. Flows are defined based on different criteria such as traffic between a source MAC address and a destination MAC address, source and destination IP addresses, TCP ports, VLANs, tunnels, and so on.

OpenFlow is one of the most widely deployed southbound standards from the open source community. It first made its appearance in 2008 by Martin Casado at Stanford University. The appearance of OpenFlow was one of the main factors which gave birth to Software Defined Networking.

OpenFlow provides various information for the controller. It generates the event-based messages in case of port or link changes. The protocol generates a flow-based statistic for the forwarding network device and passes it to the controller.

OpenFlow also provides a rich set of protocol specifications for effective communication at the controller and switching element side. Open Flow provides an open source platform for the research community.

Every physical or virtual OpenFlow-enabled network (data plane) device in the SDN domain needs to first register with the OpenFlow controller. The registration process is completed via an OpenFlow HELLO packet originating from the OpenFlow device to the SDN controller.

**OVSDB.**

An abbreviation for Open vSwitch Database, unlike openflow, OVSDB is a southbound API designed to provide additional management or configuration capabilities like networking functions. With OVSDB you can create the virtual switch instances, set the interfaces and connect them to the switches. you can also provide the QoS policy for the interfaces. OVSDB is a protocol written in the JavaScript Object Notation (JSON) that basically sends and receives commands via JSON RPCs.

**Northbound Interface**

The northbound interface provides connectivity between the controller and the network applications running in the management plane. As already discussed, the southbound interface has different available protocols, while northbound lacks such type of protocol standards. However with the advancement of technology we now have a wide range of northbound API support like ad-hoc API’s, RESTful APIs, etc. The selection of a northbound interface usually depends on the programming language used in application development.

## More Alphabet Soup of Terms

With the development of virtualization, SDN technologies, and their ecology in recent years, more and more terms and changing of these terms are emerging. A lot of confusion rises due to the context in which these terms are used. Sometimes the latest term the industry uses is a particular technology such as VNF or a concept such as NFV. Terms rise and fall out of favor as the industry evolves. In recent years, terms such as OpenStack, NVF/VNF have become the industry’s favorite buzzword. This raises the question - Just what is OpenStack, NVF/VNF and what are what is the relationships of these things with SDN?

**NFV: Networking Function Virtualization.**

NFV/VNF sounds like a new buzzword but it’s have around for years, see Figure 1.11.

![image](data:image/png;base64;base64,)

Figure 1.11 VNF/VNFI (contrail/NFX) vs NFV (vsrx) vs NMO (cso)

NFV means network function virtualization, it stands for an operation framework for orchestrating and automating VNFs.

And VNF means virtualized network function, such as virtualized routers, firewalls, load balancers, traffic optimizers, IDS or IPS, web application protectors, and so on.

In a nutshell, you can think of NVF as a concept or framework= to virtualize certain network functions, while VNF is the implementation of each individual network functions. Among others, firewalls and load balancers are the two most common VNFs in the industry, especially for deployments inside data centers. When you read today’s documents about virtualization technology, you will see the terms in such a pattern like vXXX (e.g., vSRX, vMX), or cXXX (e.g., cSRX) where the letter v indicates it is a virtualized product while the letter c equals containerized or its container version.

OpenStack

Jointly launched by NASA and Rackspace in 2010, OpenStack has rapidly gained popularity in many enterprise data centers. It is one of the most used open source cloud computing platforms to support software development and big data analytics. OpenStack comprises a set of software modules, for example compute, storage and networking modules, which work together to provide an open source choice for building private and public cloud environments. As an IaaS (Infrastructure as a Service) open source implementation, it provides a wide range of services, from basic services like computing service, storage service, networking service and more to advanced services like database, container orchestration, and others.

You can think of OpenStack as an abstraction layer providing a cloud environment on your promise. With OpenStack installed on your servers, you can spawn a VM, consume and recycle it when you are done, all in seconds (see Figure 1.12). Under that abstraction layer, OpenStack hides most complexities of automation and orchestration of diverse underlying resources like compute, storage, and networking. You could choose servers, storage, networking devices from your favorite vendors to build the underlying infrastructure, and OpenStack will consume all of them and expose them to the user as a pool of common resources like number of CPUs, RAMs, hard disk spaces, IP addresses, etc. The user does not (need to) care about vendor and brand details.

![image](data:image/png;base64;base64,)

Figure 1.12 OpenStack Launching a New Instance

If you compare OpenStack with SDN, it’s easy to see that the two models share some common features. Both provide a certain level of abstraction, hiding the low-level hardware details and exposing upper level user applications. The differences are somewhat subtle to describe in just a few words. First off, although there are various distributions from different vendors, they share common core components managed by the OpenStack Foundation. SDN is more of a framework or an approach to manage the network dynamically, which can be implemented with totally different software techniques.

Secondly, from the perspective of technical ecological coverage, the ecological aspects of OpenStack are much wider, because networking is just one of its services that is implemented by its Neutron component among its other various plugins. SDN, and its ecology, in contrast, mainly focus on the networking. There are also differences in the way that Neutron works comparing with how a typical SDN controller works. OpenStack Neutron focuses on providing network services for virtual machines, containers, physical servers, etc, and provides a unified northbound REST API to users. SDN focuses on configuration and management of forwarding control toward the underlaying network device, it not only provides user-oriented northbound API, but also provides standard southbound APIs for communicating with various hardware devices.

The comparison between OpenStack and SDN here is conceptual. In reality these two models can, and in fact are often coupled with each other in some way, loosely or tightly. One example is Tungsten Fabric(TF), which we’ll talk about later in this chapter.

# SDN Solutions Overview

## controllers

As mentioned previously, SDN is a networking solution which changes the traditional network architecture by bringing all control functionalities to a single location and making centralized decisions. SDN controllers are the brain of SDN architecture, which perform the control decision tasks while routing the packets. Centralized decision capability for routing enhances the network performance. As a result, an SDN controller is the core components of any SDN solutions.

While working with SDN architecture, one of the major points of concern is which controller and solution should be selected for deployment. There are quite a few SDN controller and solution implementations from various vendors, and every solution has its own pros and cons along with its working domain. In this section we’ll review some of the popular SDN controllers in the market, and the corresponding SDN solutions.

## OpenDaylight (ODL)

OpenDaylight, aften abbreviated as ODL, is a Java-based open source project started from 2013. It was originally led by IBM and Cisco but later hosted under the Linux Foundation. it was the first open source Controller that could support non-OpenFlow southbound protocols, which made it much easier to be integrated with multiple vendors.

ODL is a modular platform for SDN. It is not a single piece of software. It is a modular platform for integrating multiple plugins and modules under one umbrella. There are many plugins and modules built for OpenDaylight. Some are in production, while some are still under development.

![BoronDiagrams final](data:image/png;base64;base64,)

Figure 1.12 OpenDaylight Boron

Some of the initial SDN controllers had their southbound APIs tightly bound to OpenFlow, but as you can see from Figure 1.12, besides OpenFow, many other southbound protocols that are available in today’s market are also supported. Examples are NETCONF, OVSDB, SNMP, BGP, and more. Support for these protocols is done in a modular method in the form of different plugins, which are linked dynamically to a central component named the Service Abstraction Layer (SAL). SAL does translations between the SDN application and the underlaying network equipment. For instance, when it receives a service request from an SDN application, typically via high level API calls (northbound), it understands the API call and translates the request to a language that the underlying network equipment can also understand. That language is one of the southbound protocols.

While this translation is transparent to the SDN application, ODL itself needs to know all the details about how to talk to each one of the network devices it supports, their features, capabilities, etc. A topology manager module in ODL manages this type of information. It collects topology related information from various modules and protocols, such as ARP, host tracker, device manager, switch manager, OpenFlow, etc., and based on this info, it visualizes the network topology by drawing a diagram dynamically, showing all the managed devices and how they are connected together (see Figures 1.13 and 1.14).

![odl topo1](data:image/png;base64;base64,)

Figure 1.13 ODL topology.

Any topology changes, such as adding new devices, will be updated in the database and reflected immediately in the diagram.

![odl topo2](data:image/png;base64;base64,)

Figure 1.14 ODL topology update.

As an SDN controller, ODL has a global view of the whole network, therefore it has all the necessary visibility and knowledge of the network that can be used to draw the network diagram in real time.

## Open vSwitch (OVS)

OVS is one of the most popular and production quality open source implementations of a multilayer virtual switch. OVS was created by Nicira back in 2009, which was acquired by VMware. It is licensed under the Apache 2.0 license and provided by Linux Foundation. The virtual switch does most of the jobs you could expect a physical switch does but in a software method. OVS is typically run with Linux hypervisors like KVM and can be loaded on a Linux kernel. OVS supports most features supported in traditional physical switches, such as:

* 802.1Q and VLAN
* BFD
* NetFlow/sFlow
* port mirroring
* LACP
* VXLAN
* GENEVE GRE Overlays
* STP
* IPv6

Besides the functions of traditional switches, the bigger advantage of OVS is that it also has native support to the SDN solution via OVSDB and OpenFlow protocols. That means any SDN controller can integrate OVS via these two open standard protocols. Therefore OVS can work either as a standalone L2 switch within a hypervisor host, or it can be managed and programmed via an SDN controller, such as ODL. That is why it is used in so many open source and commercial virtualization projects.

## Calico

Quote from calico official website:

*Calico is an open source networking and network security solution for containers, virtual machines, and native host-based workloads. Calico supports a broad range of platforms including Kubernetes, OpenShift, Docker EE, OpenStack, and bare metal services.*

Calico has been an open-source project from day one. It was originally designed for today’s modern cloud-native world and runs on both public and private clouds. Its reputation mostly comes from its deplayment in Kubernetes and its ecosystem environments. Today Calico has become one of the most used Kubernetes Container Network Interface(CNI) and many enterprises are using it at scale.

Compared with other overlay network SDN solutions, Calico is special in the sense that it does not use any overlay networking design or tunneling protocols, nor does it require NAT. Instead it uses a plain IP networking fabric to enable host-to-host and pod-to-pod networking. The basic idea is to provide Layer 3 networking capabilities and associates a virtual router with each node, so that each node is behaving like a traditional router, or a virtual router. We know that a typical internet router relies on routing protocols like OSPF, or BGP to learn and advertise the routing information, and that is the way a node in calico networking works. It chooses BGP as its routing protocol because of its simplicity, the industry’s current best practice, and the only protocol that sufficiently scales.

Calico uses a policy engine to deliver high-level network policy management.

## VCP(nuage)

The SDN platform offered by Nuage Networks (now Nokia) is called Virtualized Cloud Platform (VCP). It provides a policy-based SDN platform that has a data plane built on top of the open source OVS, and a closed source SDN controller.

The Nuage platform uses overlays to provide policy-based networking between different clouding environment (Kubernetes pods or non-Kubernetes environments such as VMs and bare metal servers). It also has a real-time analytics engine to monitor Kubernetes applications.

All components can be installed in containers. There are no special hardware requirements.

# Overview of Tungsten Fabric (TF)

The Tungsten Fabric (TF), is an open-standard based, proactive overlay SDN solution. It works with existing physical network devices and help address the networking challenges for self-service, automated, and vertically integrated cloud architecture. It also improves scalability through a proactive overlay virtual network technique.

The TF controller integrates with most popular cloud management systems such as OpenStack, VMWare, and Kubernetes. TF’s focus is to provide networking connectivity and functionalities, and enforce user-defined network and security policies to the various workloads based on different platforms and orchestrators.

One other major advantage of Tungsten Fabric is that it is multi-cloud and multi-stack. It is made up of open standards for easier interoperability with other networking hardware like routers or switches. Today it supports:

* Multiple compute types - Baremetal, VMs and containers
* Multiple cloud stack types - VMware, OpenStack, Kubernetes (via CNI), OpenShift
* Multiple performance modes - Kernel native, DPDK accelerated, and several SmartNICs from different vendors
* Multiple overlay models - VxLAN, MPLSoUDP, MPLSoGRE tunnels or direct, non-overlay mode (no tunneling)

TF fits seamlessly into Linux Foundation Networking’s (LFN) mission to foster open source innovation in the networking space.

The TF system is implemented as a set of nodes running on general-purpose x86 servers. Each node can be implemented as a separate physical server, or VM.

**Open Source Version**

Initially, Contrail was a product of a startup company Contrail System, which was acquired by Juniper Networks in Dec. 2012. It was open sourced in 2013 with a new name OpenContrail under the Apache 2.0 license, which means that anyone can use and modify the code of the OpenContrail system without any obligation to publish or release the modifications. In early 2018, it was rebranded to Tungsten Fabric as it transitioned into a fully-fledged Linux Foundation project. Currently TF is still managed by the Linux Foundation.

**Commercial Version**

Juniper also maintains a commercial version of the Contrail System, and provides commercial support to licensed users. Both the open source version and commercial version of Contrail provides the same full functionalities, features and performances.

Throughout this book, we use these terms Contrail, OpenContrail, Tungsten Fabric and TF interchangeably.

## TF architecture

TF consists of two main components:

* Tungsten Fabric Controller: This is the SDN controller in the SDN architecture.
* Tungsten Fabric vRouter: This is the forwarding plane that runs in each compute node performing packet forwarding and enforcing network and security policies.

The communication between the controller and vRouters is via XMPP, which is a widely used messaging protocol.

A high level Tungsten Fabric architecture is shown in Figure 1.15

![TF arch](data:image/png;base64;base64,)

Figure 1.15 TF Architecture

### The TF SDN Controller Node

The TF SDN controller integrates with an orchestrator’s networking module in the form of a plugin, for instance:

* In OpenStack environment, TF interfaces with the Neutron server as a Neutron plugin
* In Kubernetes environment, TF interfaces with k8s API server as a kube-network-manager process and a CNI plugin that is watching the events from the k8s API.

TF SDN Controller is described as a logically centralized but physically distributed SDN controller. It is physically distributed because the same exact controllers can be running in multiple (typically three) nodes in a cluster for high availability (HA) purpose. However, all controllers work together to behave as a single logical unit that is responsible for providing the management, control, and analytics functions of the whole cluster.

This physically distributed nature of the Contrail SDN Controller is a distinguishing feature. Because there can be multiple redundant instances of the controller, operating in an active/active mode (as opposed to an active-standby mode). When everything works, two controllers can share the workload and load balance the control tasks. When a node becomes overloaded, additional instances of that node type can be instantiated after which the load is automatically redistributed. On the failure of any active node, the system as a whole can continue to operate without any interruption. This prevents any single node from becoming a bottleneck and allows the system to manage a very large-scale system. In production, a typical High-Availability (HA) deployment is to run three controller nodes in an active-active mode, as single point failure is eliminated.

As any SDN controller, the TF controller has a global view of all routes in the cluster. it implements this by collecting the route information from all computes (where the TF vRouters resides) and distributes this information throughout the cluster.

### TF vRouter: compute node

Compute nodes are general-purpose virtualized servers that host VMs. These VMs can be tenants running general applications, or service VMs running network services such as a virtual load balancer or virtual firewall. Each compute node contains a TF vRouter that implements the forwarding plane.

The TF vRouter is conceptually similar to other existing virtualized switches such as the Open vSwitch (OVS), but it also provides routing and higher layer services. It replaces traditional Linux bridge and IP tables, or Open vSwitch networking on the compute hosts. Configured by the TF controller, the TF vRouter implements the desired networking and security policies. While workloads in same network can communicate with each other by default, an explicit network policy is required to communicate with VMs in different networks.

As with other overlay SDN solutions, TF vRouter extends the network from the physical routers and switches in a data center into a virtual overlay network hosted in the virtualized servers. Overlay tunnels are established between all computes, communication between VMs on different nodes are carried out in these tunnels and behaves as if they are on the same compute. Currently VxLAN, MPLSoUDP, and MPLSoGRE tunnels are supported.

### TF controller components

In each TF SDN Controller there are three main components as shown in Figure 1.16.

![contrail arch](data:image/png;base64;base64,)

### Figure 1.16 TF controller components

The figure shows:

* Configuration nodes - These nodes keep a persistent copy of the intended configuration states and store them in a Cassandra database. They are also responsible for translating the high-level data model into a lower-level form suitable for interacting with control nodes.
* Control nodes - These nodes are responsible for propagating the low-level state data it received from configuration node to the network devices and peer systems in an eventually consistent way. They implement a logically centralized control plane that is responsible for maintaining network state. Control nodes run XMPP with network devices, and run BGP with each other.
* Analytics nodes - These nodes are mostly about statistics and logging. They are responsible for capturing real-time data from network elements, abstracting it, and presenting it in a form suitable for applications to consume. It collects, stores, correlates, and analyzes information from network elements.

### TF vRouter components

The TF vRouter is the main forwarding module running in each compute node. The compute node is a general-purpose x86 server that hosts tenant VMs running customer applications.

The TF vRouter consists of two components:

* The vRouter agent, which is the local control plane
* The vRouter forwarding plane

In a typical configuration, Linux is the host OS and KVM is the hypervisor. The Contrail vRouter forwarding plane can sit either in the Linux kernel space or in the user space while running on DPDK mode. More details about this will be covered in later chapters.

The vRouter agent is a user space process running inside Linux. It acts as the local, lightweight control plane in the compute, in a way similar to what a routing engine does in a physical router (see Figure 1.17). For example, vRouter agents establish XMPP neighborships with two controller nodes, then exchanges the routing information with them. The vRouter agent also dynamically generates flow entries and injects them into the vRouter forwarding plane. This gives instructions to the vRouter about how to forward packets.

![contrail vrouter1](data:image/png;base64;base64,)

Figure 1.17 vRouter Agent.

The vRouter forwarding plane works like a line card of a traditional router (see Figure 1.18). It looks up its local FIB and determines the next hop of a packet. It also encapsulates packets properly before sending them to the underlay network and decapsulates packets to be received from the underlay network.

We’ll cover more details of TF vrouter in the later chapters.

![contrail vrouter2](data:image/png;base64;base64,)

Figure 1.18 vRouter Forwarding Plane.

# References

* <https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf>
* <https://www.opennetworking.org/sdn-definition/>
* <https://www.openvswitch.org/>