UNIVERSITY OF ALBERTA

SDN USE CASES AND IT'S APPLICATIONS

Oleksii Ignatov, MINT

ABSTRACT

Since the invention and adoption of server virtualization, multitenant datacenters and cloud computing have made significant impact on networking industry. Virtual servers became mobile and easily movable from one physical location to another. As a result, the way data is stored and software is used was completely reshaped. Traditional networking designs are not able to fulfill today's business needs. This challenges are mostly related to static nature of networks and it's inflexibility. This study explores the structure and capabilities of SDN framework in perspective of solving the raised problems in today's networks.

TABLE OF CONTENTS

Abstract 2
Introduction 5
    Current networks 5
        Complexity 5
        Network is Inconsistent 6
        Scaling factor 6
        Vendor specifics 7
    Need of change 7
        New traffic behavior 7
        The rise of cloud services 8
Part 1. Background SDN concepts and other related topics. 8
    Engineering Need for software defined networks. 8
    SDN Framework 9
        Infrastructure layer 12
        Control layer 13
        Application layer 14
    SDN Operation 15
        Existing SDN Device Implementations 17
    SDN controller architechture 18
        SDN Controller essential functions 19
    SDN programmability 20
        SDN via legacy API 21
        REST-based API 21
    Network Applications 22
    Overview of available controllers and their capabilities 23
        Open Source Controllers 24
SDN Applications 34
    SDN Use cases 34
        SDN in Data centers 34
        SDN in Enterprise Networks 38

Hybrid WAN 43

High-level Design 44

Future activities in SDN community 53

Conclusion 57

References 58

INTRODUCTION

The invention and adoption (wide spread use) of server virtualization, multitenant datacenters and cloud computing have made significant impact on networking industry. Virtual servers became mobile and easily movable from one physical location to another. As a result, the way data is stored and software is used was completely reshaped.

Content access underwent considerable transformation with prevalence of east-west direction since databases scattered across physically separated datacenters. Applications access different servers, creating a high volume of machine-to-machine traffic before sending data back to the client.

Traditional networks were not ready for such rapid evolution in servers and turned out to be very static in nature. Today's networks are complex and hard to configure along with implementation of different policies network-wide. Configuration of thousands of devices and a number of protocols makes it highly difficult to keep it consistent considering access, quality of service (QoS), security and other policies. Adding or moving of any device becomes very troublesome leading to reconfiguration of multiple network nodes that takes a lot of effort, time and a great risk of service disruption.

A rapid raise of mobile personal devices makes a significant impact on traffic behaviour and forms new requirements for corporate access: any type of device, any location, at any time. Employment of private clouds and adoption of BYOD policies creates a lots of obstacles for IT departments to make it secure, easy to control and convenient to manage. Networks utilize a great amount of components to make it work as desired. All this components must be tightly integrated and correlated, which is hard to achieve.

CURRENT NETWORKS

Traditional networking designs are not able to fulfill today's business needs. Proliferation of mobile devices and numerous applications that require different types of access to data creates a great deal of troubles for network specialists to challenge. This challenges are mostly related to static nature of networks and it's inflexibility:

COMPLEXITY

Current network design comes to a collection of networking nodes that "think" independently according certain rules – protocols. These protocols are discrete and exist separately from each other. Furthermore, they have no visibility/integration with applications they provide service to.

This distributed model of thinking and making decisions has led to increasing complexity [1] in terms of configuration at the first place. Enforcing of desired network policies requires individual reconfiguration of switches, routers, firewalls, etc. using vendor-specific CLI. It is especially critical in virtualized computing environments where virtual machines can be moved or added in a few mouse clicks.

Nowadays, applications are scattered across virtual machines (VM) as opposed to traditional design where they were tied to specific physical servers. Such distribution of applications imposes a lots of data exchange between VMs that are mobile units by their nature. In order to optimize computing resources utilization they were designed to be movable and easy to migrate from one location to another. These features of virtualization do not comply with capabilities of current networks in aspects of addressing and consistent policies across networks [2].

As a result, adding changes to configuration is a time-consuming and challenging process. It requires a great deal of planning and manual reconfiguration in spite of all advancements in network management systems [3]. Furthermore, there are no mechanisms that could allow networks dynamically react to instant changes or faults in the topology.

In addition to adopting virtualization technologies, many enterprises today operate an IP converged network for voice, data, and video traffic. While existing networks can provide differentiated QoS levels for different applications, the provisioning of those resources is highly manual. IT must configure each vendor's equipment separately, and adjust parameters such as network bandwidth and QoS on a per-session, per-application basis. Because of its static nature, the network cannot dynamically adapt to changing traffic, application, and user demands.

NETWORK IS INCONSISTENT

Network is inconsistent. Network has been much smaller at inception. It grows in order to keep up with demands of market. Thousands of applications that encompass data, voice and video are utilizing networks. Each group of application has its requirements for communication and demands individual treatment based on its nature, type of devices, connection types. From the network prospective, it means provisioning of access, security, QoS and other policies that are very problematic to keep consistent, especially in large deployments.

SCALING FACTOR

As demands on the data center rapidly grow, so too must the network grow. However, the network becomes vastly more complex with the addition of hundreds or thousands of network devices that must be configured and managed. IT has also relied on link oversubscription to scale the network, based on predictable traffic patterns; however, in today's virtualized data centers, traffic patterns are incredibly dynamic and therefore unpredictable.

Explosion and constant development of applications result in new demands for data canters. Dynamic nature of the virtualized environment has made traffic behaviour highly unpredictable.

These scalability issues [4] become exceptionally noticeable in huge service providers (Google, Amazon etc.) Millions of users access these services at a time and they all expect a real-time experience from their applications. Computing resources are multiplying to provide required service, VMs and create myriads of data flows between them. Dynamic nature of virtualized environment leads to constant changes in the network connections. At the scale of huge networks with thousands of nodes, it is practically impossible to keep network consistent using manual configurations. Multi-tenant deployments which are virtually the case for every carrier make things even worth, as networks must satisfy the requirements of all different clients with different needs. Simple configurations of policy based routing become hardly manageable at a large scale.

VENDOR SPECIFICS

Carriers and enterprises seek to deploy new capabilities and services in rapid response to changing business needs or user demands. However, their ability to respond is hindered by vendors' equipment product cycles, which can range to three years or more. Lack of standard, open interfaces limits the ability of network operators to tailor the network to their individual environments.

User demands form actual capabilities and features that have to be implemented in carriers' networks. In order to satisfy clients' needs, constant upgrades have to be performed on the network side. It becomes a problem because it all depends from vendors' resources and their internal update cycles. Vendors implement many software features on hardware, limiting chances of timely updates as needed. This inflexibility and lack of programmability is dragging industry back from the real business needs.

Providing programmatic interface provides the means for innovation to take place in network infrastructure.

All abovementioned is important, as it created core motivations for what has evolved into Software-Defined Networking.

NEED OF CHANGE

Conventional networking design implies a three tired model of implementation. This approach was relevant for client-server architecture but it is absolutely inapplicable to modern application requirements. The rise of cloud computing and burst of mobile devices stimulates networking industry to reconsider network designs significantly [5].

Therefore, what exactly is driving these changes in network architecture:

NEW TRAFFIC BEHAVIOR

Finally, many enterprise data centers managers are contemplating a utility computing model, which might include a private cloud, public cloud, or some mix of both, resulting in additional traffic across the wide area network.

Interesting phenomena is observed in modern datacenters as traffic behavior has changed drastically. Virtualization technologies and modern application designs have become main factors that majority of traffic is exchanged between VMs. This machine to machine communication is very different from what we get used to see in classical datacenters with client-server architecture. That kind of change requires redesign in datacenters architecture.

BYOD trends affect the way traffic is generated and how it must be handled. This solutions demand enormous flexibility from network layer as users need access from any device, at any time, from anywhere.

THE RISE OF CLOUD SERVICES

With the ongoing spread of public and private cloud services, deployment of new services and applications have become significantly easier [6]. However, cloud services themselves pose new challenges for relatively static network infrastructure. Users expect to have access to applications in an agile fashion. As a result, cloud deployments add significant complexity to networks in order to provide required level of service to users. Ideally, to overcome this challenges, both computing and network resources should be equally resilient, scalable and controlled with common suite of tools.

Handling large amounts of data requires enormous number of servers that are capable of highly scalable parallel processing. As it was already mentioned, it causes extensive server-to-server also known as East-West, traffic that could be efficiently handled by high-speed and fully interconnected switching-fabric in the datacenter. Close networks is an old concept, but becomes extremely relevant for current challenges in network design and extensively used by many vendors.

PART 1. BACKGROUND SDN CONCEPTS AND OTHER RELATED TOPICS.

ENGINEERING NEED FOR SOFTWARE DEFINED NETWORKS.

Many pieces of work were introduced that preceded SDN and eventually led to development of it [7]. In a computing world if there are needs for application the systems can be programmed to satisfy those requirements. Networking never had this concept. This inflexibility and rudimentary way of configuring became a problem, especially on a large scale networks. People needed a programmatic interface - a way to program the behaviour of networks tailored to their specific requirements.

This pressing need for a change in a way the networking is done, originated from a business need, pushed by the people who operate large networks [8]. Networks have grown substantially right now with constant addition of more services and capabilities, and attempts to make them more reliable. The only tools network designers had in their disposal are individual boxes build for them by vendors – closed, proprietary and build on technology that is way behind leading edge technologies in computer systems. Network equipment didn’t have a flexibility or programmability. Configuration of network devices in a conventional way boils down to CLI and is done individually for every node in the network. The most advanced form of control was writing scripts to download configurations to network devices. And is usually performed by network management systems utilizing conventional CLI that serves more like a configuration work automation system while SDN provides means to program network behaviour not considering any CLI-based limitations.

Key objectives:

- Network is critical. Network is inconsistent. Network has been much smaller at inception. It grows.
- Network makes use of powerful and feature rich equipment. Vendors compete in making that equipment better. Vendors are oriented to financial gain.
- Devices are configured by engineers. Engineers have to keep up with vendor-specific features in big, inconsistent, networks. Engineers are prone to mistake. Nonstandard configurations may cause complications in multi-vendor networks.

SDN promise is a promise of a common, programmable, adaptable, interface to manage the network.

SDN FRAMEWORK

Software Defined Networking (SDN) is one of the most significant changes the networking industry has seen in recent years. Mainly it has been motivated by increasing requirements for dynamic, programmable high-speed network systems from emerging trends of virtualized cloud computing, mobility and big data applications. SDN is expected to promote network innovations [9] and to largely simplify management of large networks. However, most principles behind SDN are not entirely new - network programmability was experimented with Active Networking in the 1990s [10].

Original definition of SDN comes from works on OpenFlow [11] and mainly based on design where data plane is a separate entity of network abstraction being controlled by remote control plane. This separation of concerns provides an abstraction of underlying networking hardware and makes it accessible for applications to impose specific requirements and have certain level of control of it [12].

SDN framework can be characterized as based on such basic principles:

1. Decoupling of controller and data planes [13]. Control functions are moved from network nodes to a separate control unit, called controller. SDN controller keeps track of network states and becomes a main orchestration unit network-wide. Network elements are left with a role of basic packet forwarding units.

2. Control centralization. Controller plays a role of logical supervisory entity and performs programmatic functions based on abstract view of network resources. Decision making process improves because of a holistic network view controller gains in this model. It also becomes beneficial in prospective of scalability that has become a major issue for today's large-scale networks [14]. Centralization simplifies network management and operation making it less susceptible to configuration errors as opposed to device independent CLI configuration.

3. Programmability of the network allows applications to utilize APIs and adjust network layer behavior based on current conditions or requirements of particular application.

Following SDN concept defined in [15] by ONF, an SDN architecture (illustrated in Figure 1) can be logically represented as three main layers that perform specific functions:

- Infrastructure layer or forwarding hardware consists of network devices, which utilize control-data plane interface (e.g. Openflow) to communicate with controller and expose their capabilities.
- Control layer or SDN controller. Controller establishes communication channels with network devices and formulates flow entries based on applications' requirements and extracting status, state and event information from forwarding devices back to applications via application-controller API.
- SDN Application layer, comprise network services, utilities and applications which communicate with control level to specify their network requirements via the application-controller interface.

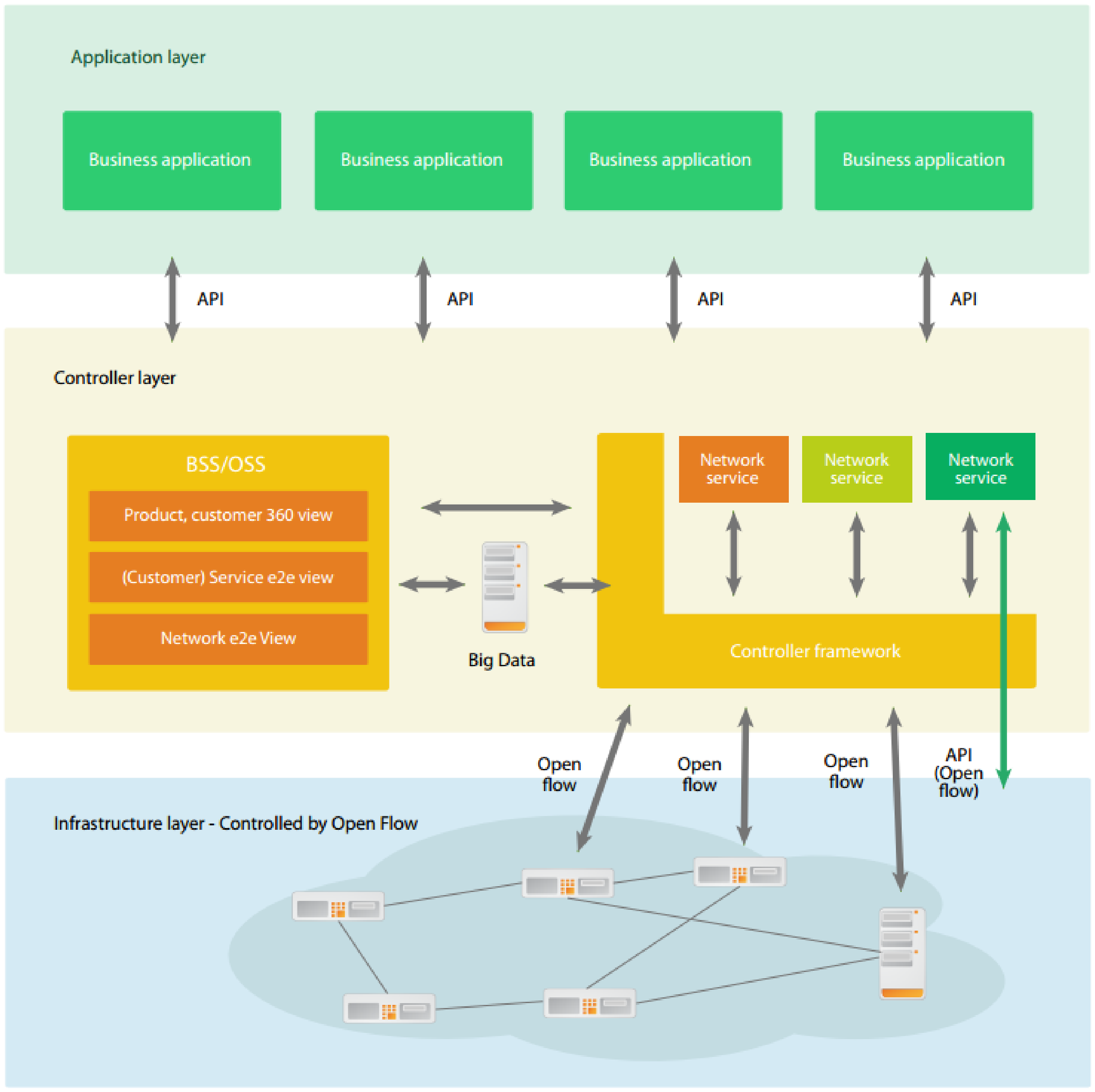


Figure 1. Software Defined Network Architecture

As the result of this layer abstraction, network is presented as one logical entity that can be easily accessible by applications via predefined APIs [16].

The layer separation provides several benefits:

- Development and implementation of applications becomes simpler since network is represented as abstraction via control plane that directly interacts with network devices.

- Centralized approach for policy implementation across the network. Applications use well-defined APIs to translate their requirements to underlying hardware. Helps to avoid inconsistency comparing to traditional discrete device configuration;
- Simplification of feature implementation and their management. Specific network services (firewalling, traffic optimization, load balancing etc.) represent software modules that can be applied in a predefined order or conditionally.

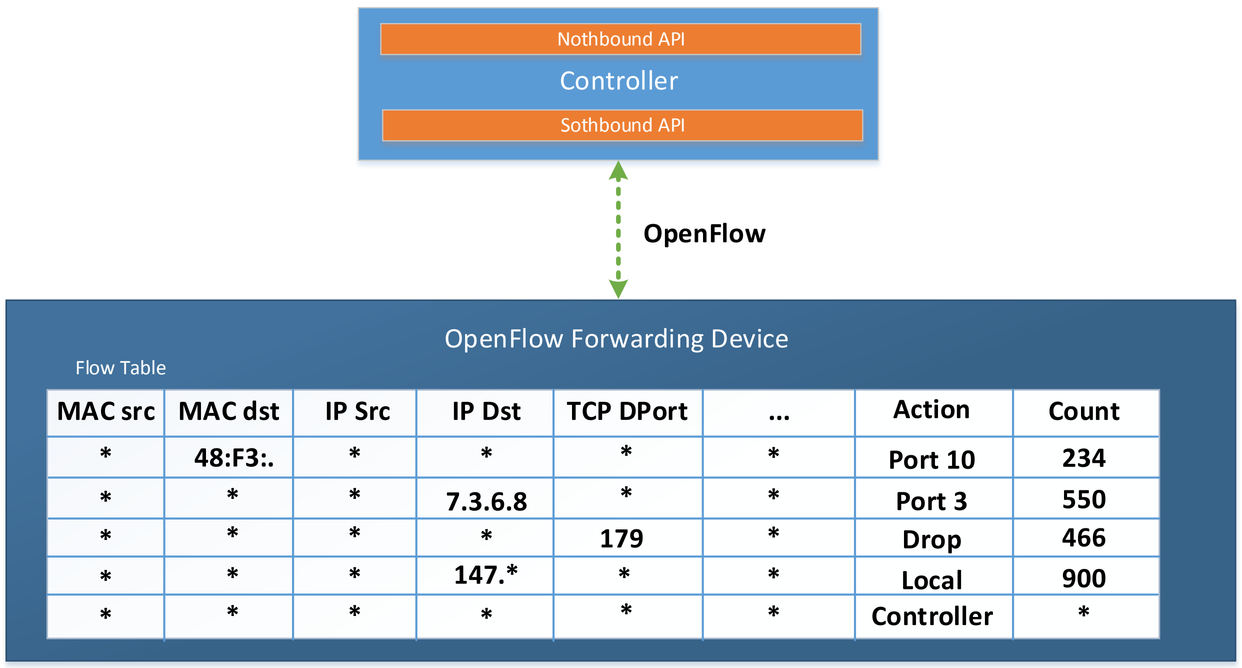
INFRASTRUCTURE LAYER

Very much in the same way as in traditional networks, SDN data plane represents a collection of networking hardware equipment that carries user traffic and enables data to and from clients.

The key difference is that network devices are now simple packet movers and don't have to make any decisions individually. Every protocol related activity (switching and routing path selection) is performed at logically centralized network engine – SDN control software. Additional processing functions can be implemented on SDN controller or by distinct system that may work under supervision of controller.

All network devices are controlled via standardized interface (e.g., OpenFlow protocol [17]) that enables modifications of forwarding tables according to instructions from control layer. Availability of well-defined interface allows to instantly impose policies, modify forwarding paths across thousands devices in the network avoiding compatibility issues and possible configuration inconsistency. Something that is extremely painful in traditional networking and characterizes them as static.

OpenFlow network devices could be represented as hardware or software units of network whose main and only task is forwarding of packets. Based on OpenFlow specification [18] the forwarding process relies on a pipeline of tables. Each table consist of set of instructions: 1) traffic define rules – match function; 2) rules of how to treat the traffic – action function; 3) statistic information – counters that are updated when packets are matched.



MAC src	MAC dst	IP Src	IP Dst	TCP DPort	...	Action	Count
*	48:F3:.	*	*	*	*	Port 10	234
*	*	*	7.3.6.8	*	*	Port 3	550
*	*	*	*	179	*	Drop	466
*	*	*	147.*	*	*	Local	900
*	*	*	*	*	*	Controller	*

Figure 2. OpenFlow Instruction Set

During the evolution of OpenFlow protocol it has obtained new features and improvements, such as multi flow tables group and meter tables [19]. This series of tables in the network element defines the actual forwarding path. Each flow table contains multiple flow entries. Every packet received, goes through the lookup process starting from the first table and depending from matching flow entry may be forwarded to destination port or to other table in the pipeline. In case packet doesn’t match any entry of the tables it automatically checked with table miss entry in the table. The default behaviour is to send this packet to the control layer for further processing, or to the normal switch pipeline in case of OpenFlow-hybrid switches. This default behaviour can be modified according to specific requirements.

Each table entry has a priority and listed sequentially in a table. First rule that incoming packet matches will be chosen to execute actions on that packet. Packet could be forwarded to predefined outgoing port, dropped, redirected to the control layer, or other flow tables.

Every new version of OpenFlow introduces new features and improvements. However, only a subset of those matching fields are mandatory to be compliant to a given protocol version. Similarly, many actions and port types are optional features. Flow match rules can be based on almost arbitrary combinations of bits of the different packet headers using bit masks for each field.

CONTROL LAYER

Control layer is designed to be a single logical entity that controls all network devices in the underlying infrastructure layer. In comparison to traditional networks that use distributed model of topology handling with device specific configuration languages and proprietary vendor operating systems. Control plane scattered across network elements creates a great deal of complications that are mostly troublesome during network convergence and topology changes.

Software Defined Networking approach is targeted to simplify network management by means of centralization. Main task of control layer is to provide abstraction of physical network resources (e.g. switches, routers) and to facilitate an easy access to those resources for upper application layer. Along with business applications, regular network services as routing protocols, switching loop prevention mechanisms and configuration propagation can run as separate modules on top of SDN controller. With controller-based approach, consistent policy deployment becomes simpler and there is no need for network operator to go deep in details of forwarding path that application traffic is following.

Controller is a master unit of SDN network as it dynamically modifies network equipment configurations based on application instant requirements or network administrator configuration. Network-wide management is handled via main instance of controller, while there are other instances that are being synchronized with the main one. This is a typical approach for modern SDN networks that is horizontally scalable and quiet resilient to failures. It serves as interpreter between infrastructure layer and application layer therefore hiding all details of forwarding devices interaction.

From the architectural prospective there are three main modules that can be defined in general SDN controller: 1) the application and services with northbound interfaces; 2) controller module with networking submodules; 3) forwarding layer interaction via southbound interfaces. We will get more in details of each module in further chapters.

APPLICATION LAYER

Application layer represents business applications that network applications that with the abstraction through controller obtain means of imposing real-time requirements to the network infrastructure.

Software Define Network by design provides methods of requesting specific network behavior as it required for business applications to work effectively. Logical interface responsible for interaction with applications is called as Northbound Interface (NBI) supported by ONF [20]. It lies between controller and actual applications and can be defined as a collection of APIs for communication.

Communication across NBI can be described as requests about the current condition of the network layer, or specific instructions from applications to modify network configuration. Instructions may contain commands for connectivity changes between network elements or for specific policy enforcement (ACLs, QoS etc.) Since controller layer has network-wide view of the physical and virtual infrastructure, network service applications may modify traffic flows in order to implement a service chains through specific services in the network.

As a result, this kind of abstraction provides an easy way for applications to make use of networks more efficiently through specific interfaces. It sets applications free from the detailed awareness of protocol implementations and functioning. Integrating applications with the network leads to more efficient and optimized use of it, because of inherent network awareness.

SDN OPERATION

As a concept, Software Defined Network functioning is relatively easy to grasp. Figure 3 depicts fundamental SDN topology containing all main components: Controller, network devices and applications. As it was mentioned before, network devices implement forwarding capabilities of SDN architecture and logically composed of data and forwarding components. All packets that come into the device represent data that is processed according to preprogrammed flows. Forwarding component is basically hardware elements that perform data movement form interface to interface.

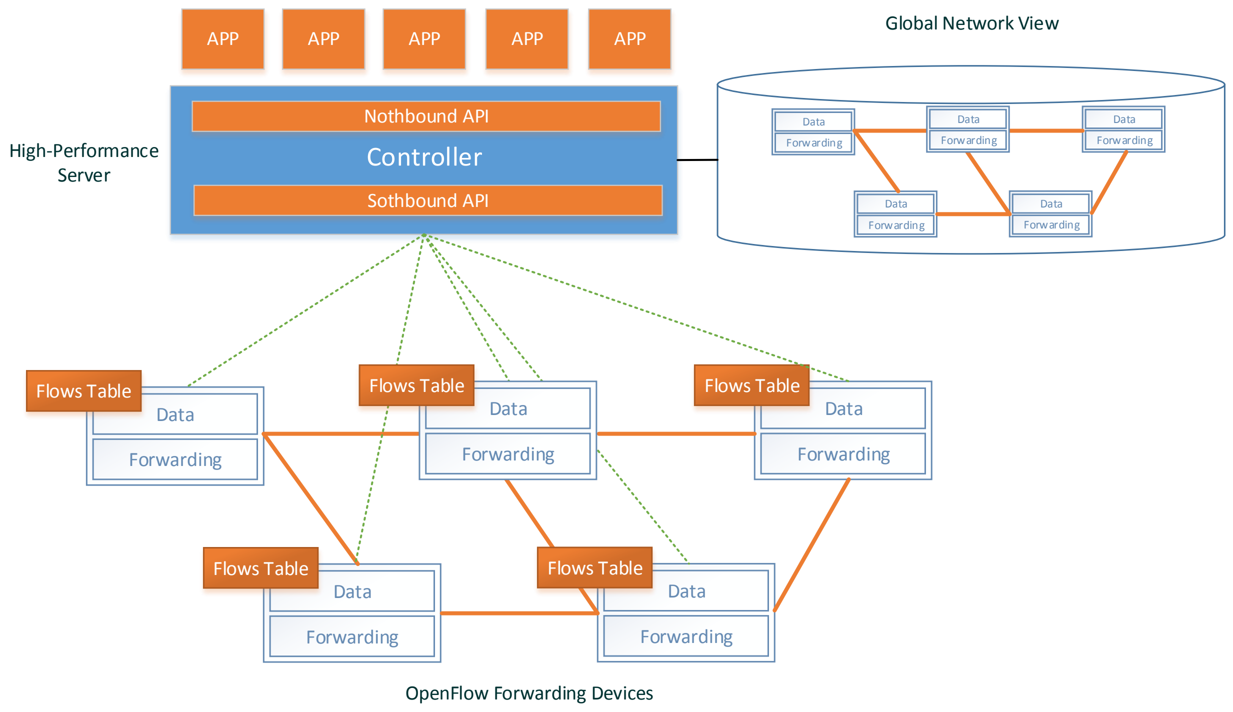


Figure 3. SDN operation

A group of packets that have common network parameters (source IP, destination IP, TCP/UDP ports) forms a flow. Each flow in the SDN network will have a calculated flow entry that describes actions which have to be performed with every packet form particular flow. It is worth mentioning that flows are unidirectional and one pair of endpoints will have different flow entries for each direction of communication.

Every SDN network device has a flow table that contains a list flow entries with corresponding actions to be executed for matching packets. When network device connects to the controller it receives a set of flow entries calculated by the controller. For every incoming a packet network device goes through the list of flow entries in the flow table looking for match. In case of match, a preconfigured action that usually means forwarding it to corresponding port leading to destination endpoint. If there is no flow entry that matches parameters of incoming packet it can be either dropped or forwarded to the controller for further processing.

One of the main tasks of the SDN controller is to provide an abstracted view of network resources to application layer. In this way applications get means of controlling network devices installing and changing flow entries in the flow tables of network devices. SDN controller serves as intermediate interpreter between applications and network devices. Centralized view of the entire network allows effectively perform forwarding with very flexible capabilities of path optimization.

SDN network applications reside on top of controller. All network logic is now migrated to controller and instead of distributed applications (control plane) on each network device, controller has set of applications that are used to control traffic forwarding at layer two and three of OSI model. These applications utilize northbound interface of the controller for communication with network layer: flows installation and modification, receiving data packets that did not have match in the flow table. There are two types of flows that can be distinguished by time of installation: proactive and reactive. Proactive flows are installed into networking devices upon application start up. Reactive flows are those that installed or modified based on some changes in the application network requirements or other circumstances: links failures, traffic load changes. These flows could be constructed upon receipt of packets passed to the controller from network devices. Appropriate application will receive an incoming packet and will instruct controller as what actions have to be taken. The most often case is generation of corresponding reactive flow that is immediately installed to the forwarding table of the network device. Service applications as firewalls, load balancers, traffic optimizers may also initiate flow entries changes in response to network behaviour alterations or external alerts.

Figure 4 shows controller to network device communication via OpenFlow protocol as the only standardized southbound protocol for network devices management. There are also other standards, that may have some vendor proprietary features. More detailed overview of available northbound and southbound interfaces will be discussed in later chapters.

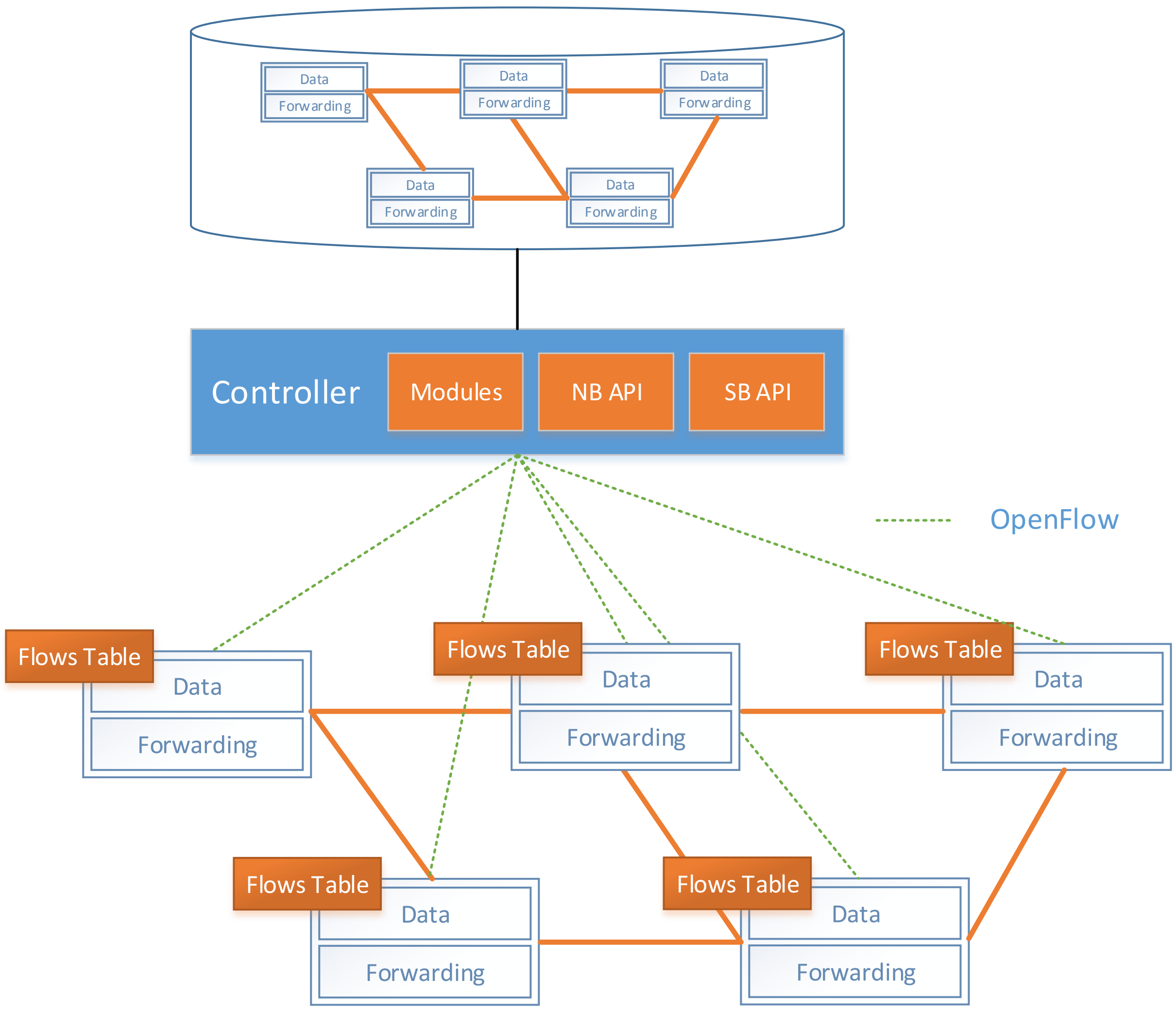


Figure 4. OpenFlow communication

EXISTING SDN DEVICE IMPLEMENTATIONS

Great variety of SDN exist and available for use today. Provided by well-known vendors as Cisco, HP, NEC, IBM, Juniper and others. Open source SDN solutions that are largely supported come from Nicira and Big Switch: Open vSwitch [21] and Indigo [22] respectively. As first steps in SDN adoption, main network vendors have added OpenFlow support to their existing series of switches. Hybrid network devices that can operate in both modes seem to be the most logical way for SDN appearance on market.

As a promise of lessen of cost of networks a new class of devices has emerged, which is called white-box switches. Main idea of these products is related to combination of merchant silicon switching chips produced by low-cost device manufacturers and free software solutions like OVS or Indigo. As a result, solutions based on white-box switches will require less expense on network infrastructure. Architecturally, most of network functions will reside on the controller side hence it will undoubtedly reflect on processing requirements of the controller hardware. However, total cost effectiveness is case-dependant.

SDN CONTROLLER ARCHITECHTURE

DESCRIPTION OF MAIN CONTROLLER FUNCTIONS.

A logically centralized control plane is a main concept of Software Defined Networking comparing to traditional networks that are mostly build in a distributed fashion. However, rapid evolution of applications poses new network requirements. From past experience [23] traffic engineering problems like load balancing could be better solved with a global view the network. Centralized approach could be also extremely applicable to maintain uniform policies across network domain [24].

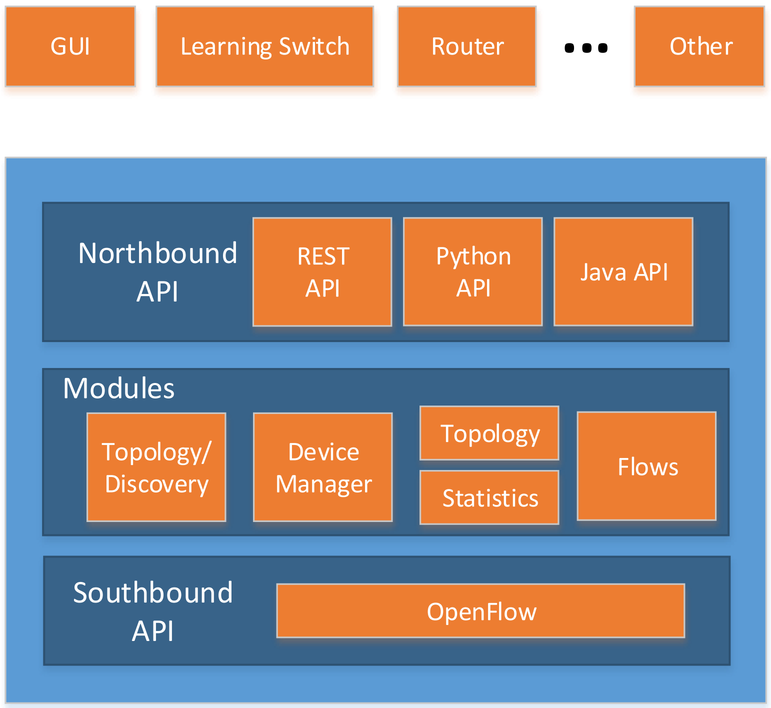


Figure 5. SDN controller components

Although, every individual controller has its own application and design specifics they all have a common set of functionality. Functions such as topology, statistics, notifications and device management, together with shortest path forwarding and security mechanisms are essential network control functionalities that applications may use for network control.

By design, one of the main controller tasks is maintaining a holistic view of the network altogether, mange network layer devices and provide means to program them. To clarify, the decision making process of traffic forwarding or other specific means of treatment are based on joint operation of controller and SDN applications. These applications usually come as modules of the controller that implement specific switching, security or other functions in contrast to business applications that utilize network a transport service.

Figure 5 depicts general structure of an SDN controller, including basic modules that from core controller's functionality, communication interfaces (API) for integration, and typical business applications. Southbound API provides communication means to control SDN network devices. As it was mentioned before, Openflow is the most common API protocol to facilitate these functions, but there are other possibly proprietary APIs. Some controller implementations may support several Northbound APIs on the same controller. Because of OpenFlow was the first implementation of Northbound API it received broad support and well development up to standardization. OpenFlow's companion protocol, OF-Config [25], and Nicira's Open vSwitch Database Management Protocol (OVSDB) [26] are both open protocols for the southbound interface, though these are limited to configuration roles.

Unfortunately, there is no corresponding northbound standard that puts some limitations on flexibility of SDN implementations. That leads to separate forms of implementations of Northbound interface that differ from controller to controller. For example, the Floodlight controller [27] includes a Java API and a *Representational State Transfer* (RESTful) [28] API. The OpenDaylight controller [29] provides a RESTful API for applications running on separate machines. The northbound API represents an outstanding opportunity for innovation and collaboration among vendors and the open source community.

SDN CONTROLLER ESSENTIAL FUNCTIONS

As a main unit of abstraction, controller provides application acceptable means of interactions with the network leaving network configuration details aside of application development. Essential features that every controller should provide can be summarized as:

- **Device discovery.** Discovery of end-user devices (laptops, desktops, printers, mobile devices) and network devices that encompass network forwarding plane (switches, routers etc.)
- **Topology management.** Gather and sustain information about the device connections along with topology changes
- **Flow management.** Track changes in flow database and perform modifications according to topology changes.

The controller is a central control unit in the network with all logic functions emplaced. Consequently it is responsible for collection of all information about network devices in the network and tracking network changes. Every controller architecture has a modular structure with independent modules that provide specific network functions listed above. Since all network operation is centrally orchestrated, the controller maintains flow information copy from every device keeps those synchronised.

SDN PROGRAMMABILITY

Network development has been ongoing process for years and through this timeframe, there were many solutions to adapt network infrastructure to new requirements and improve scalability, reliability and performance. Maintaining large networks as they grow in dynamic application environment have become a crucial concern for the last decade. In attempts to improve situation centralized network management appeared to be the only logical solution.

Similar to SDN but utilizing different technique, such as conventional API methods, a new class of network solutions has appeared, SDN via APIs.

Legacy APIs program control plane resides on the network device in contrast to OpenFlow that is designated to directly control the data plane. This forms a crucial distinction between new and legacy APIs. SDN via APIs makes a significant difference as it brings certain degree of programmability to traditional networks and allows SDN controllers to utilize old-style management APIs for network management. Nevertheless, this approach resembles hybrid solution, it is important to emphasize that control plane becomes a software defined as well as programmable centralizing network logic.

Network APIs that are currently in use:

- The Command-Line Interface (CLI) is the oldest and is reputed to be the most flexible in terms of configuration capabilities. It is vendor dependant and human oriented. Any management software that attempts to use specific CLI must behave human-like logging into CLI, putting commands and read the results. It can also be characterized as not application friendly as it causes very noticeable time delays between configuring the device and collecting statistics to make further changes or input new instructions;
- Simple Network Management Protocol (SNMP) has been present in networking since 90s and was originally developed for management network devices. Unfortunately, in spite its original purpose of configuring (SET function) and monitoring (GET function) SNMP was primarily used for network monitoring instead of configuration. Majority of network operators continued to use CLI as it was text-based and included specific features that were not available through SNMP;
- NETCONF is an XML-based management protocol that was developed to substitute SNMP. It provides means to configure, change, delete configurations of networking devices and other handy functionality (asynchronous event notifications, partial locking of the running configuration, stand-alone entity etc.) to make it convenient and scalable in use.

All these methods have been in use for a long time and have been advancing to improve usability. However, control plane remains distributed even though it may have a central programmable logic to perform management. This approach does not scale well in data center environment, as configuration for each individual device is still complex and vendor-specific.

SDN VIA LEGACY API

Most of the vendors have SDN solutions implemented according to their vision of SDN. Some vendors base their SDN solutions on legacy APIs as means to control networking devices with the goal to utilize existing network equipment while applying SDN designs. Cisco introduced proprietary SDN solution based on Cisco onePK [30] that is basically new set of APIs that are tighter integrated with network equipment control software and provide a richer feature set to manipulate switches and routers in SDN fashion. The control plane remain separated although it has software defined controller orchestrating network equipment in centralized fashion. The same path was chosen by another network equipment provider Arista [31]. Arista's vison of SDN falls into SDN over APIs category and is oriented on scaling traditional network architecture with implementation of appropriate APIs.

OpenDaylight is one of the representatives of open source implementations that may be called a hybrid controller. Due to support of other southbound interfaces apart from OpenFlow, OpenDaylight controller is capable of fitting in both design approaches original SDN and SDN via APIs.

To summarize, SDN via APIs is plausible approach that has commercial implementations and beneficial on early stages of SDN development. It helps to promote SDN in production because of utilization of existing network equipment while meeting some goals of SDN.

REST-BASED API

RESTful interfaces have become a preferable solution for Software Defined Network architecture. REST is not a new technology and was used extensively to request information from different services via web interface (HTTP). REST approach prevails because of number of advantages it introduces:

• **Simplicity.** REST interfaces use simple and well-known HTTP instructions in order to request data from services behind web service.

• **Flexibility.** Configurations are exposed to other entities via separate URLs that is less complex than SNMP MIB modules that has to be pre-computed. Access to new resources can be easily provided through specific URLs.

• **Security.** REST API may utilize HTTPS as widely used and commonly acceptable way of secure communication that is easily routable through numerous security devices.

REST-based APIs are flexible and extensible and found its application in Northbound interfaces as a common mechanism to program SDN networks. Variety of controller implementations use REST-based methods to expose SDN application functionality to developers.

NETWORK APPLICATIONS

SDN network applications are related to the network management software that performs general network functions as in traditional network and could be dedicated to high-level traffic processing functions (firewall, IPS, load balancing etc.). SDN network applications interface with the controller via its API and program flow tables of network devices. Essential functions that are provided by the applications are:

1. Flows configuration;
2. Flows modifications in respond to topology changes;
3. Traffic steering according to network load;
4. Traffic redirection for security checks, user authentication etc.

These examples represent some typical SDN applications that have been developed by researchers and vendors today. Applications such as these demonstrate the promise of SDN: being able to take complex functionality that formerly resided in each individual network device or appliance and allowing it to operate in an Open SDN environment.

SDN network applications represent centralized control-logic that is translated via controller's API to network devices. Each application is dedicated to specific function it has been developed for. Most of current applications preform traditional network functions such as switching, routing, load balancing, security policies. Application behaviour depends from event messages it receives from the controller and other applications that might be integrated in order to consider external conditions in control logic. The basic principle of application-to-controller interaction is depicted in Figure 6.

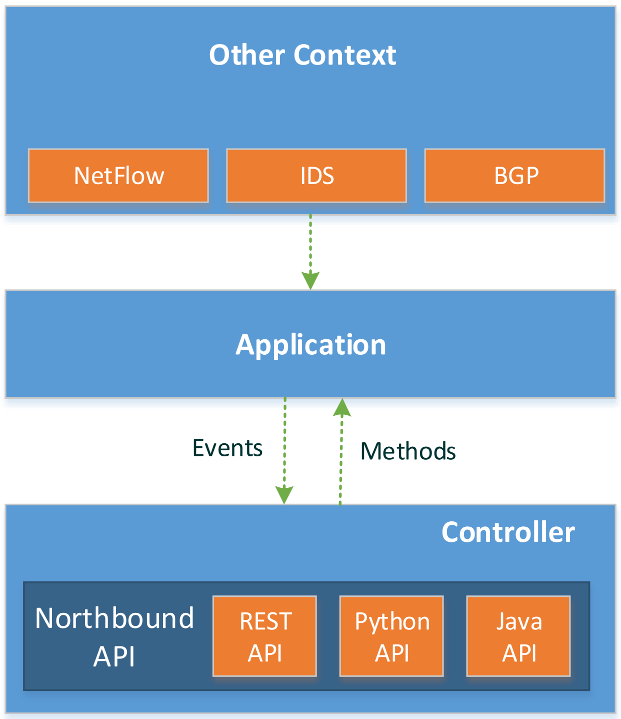


Figure 6. Northbound API

Controller preprogrammed to generate a variety of notification messages “events” in response to state changes, network device registration and many others. Once certain event occurs and it has a loaded application listener to it, the controller will execute application’s method to handle it.

To summarise, main components of application-to-controller interaction are events, listeners and methods

Application responses to events raised by the controller totally depend from desired implementation: starting from basic injections of default flows upon network device connection, to complex decisions based on conditions from other applications. The conditions may rely on current network state, security policies, business application requests.

Based on the purpose all applications can be grouped as:

- Traffic engineering applications;
- Wireless management applications;
- Management and monitoring applications;
- Security applications.

OVERVIEW OF AVAILABLE CONTROLLERS AND THEIR CAPABILITIES.

Given the broad discussions and current ‘noise’ in the market, numerous options of SDN controllers have become available for testing and production deployments, including commercial and open source alternatives. Implementation and internal design of available controllers varies depending from programming language used: C-based NOX [32] controller or Java versions of Beacon [33] and Floodlight [27]. Interface APIs options is another concern that impacts deployment flexibility and controller selection. Most of the controllers provide RESTfull type of interfaces for communication. Well-known OpenDaylight controller is an collaborative project aimed to accelerate SDN adoption was developed by partnership of vendors. Practically every vendor has engaged in SDN development race and produced its own, vendor-specific SDN controller.

Nevertheless proprietary SDN controllers do not conform to one of the main promises of SDN and represent closed systems from the prospective of vendor dependence these solutions have some benefits in use. The most valuable advantage consists in possibility of utilization of legacy network equipment as most of the vendors provide operating system upgrades to support SDN functionality. This allows faster and less costly SDN migration in production networks.

OPEN SOURCE CONTROLLERS

Open Source controllers play a huge role in SDN promotion and allow to perform extensive testing of applications and network infrastructure. For that reason, it is important to examine the most promising open source SDN controllers and their capabilities. Table below summarizes some of the most relevant open source SDN controllers providing brief comparison of main characteristics.

Core elements	OpenDaylight	OpenContrail	Floodlight	Ryu	Beacon
Network services	Topology, statistic, switch management, SPF algorithm, Host tracking	Routing, Tenant Isolation	Topology Manager/Routing, device manager	Endpoint/ topology management, L2 switching	Topolgy management, routing
Northboud API	REST, JAVA API	REST	REST (JSON)	REST	Openflow-based API
Southbound API	OpenFlow, SNMP, NETCONF, OVSDB	XMPP and NETCONF	Openflow	Openflow, OVSDB, OFCONFIG, NETCONFIG, XFLOW	Openflow
Management interfaces	GUI, REST	GUI	GUI (Java)	CLI	GUI

OPENDAYLIGHT

THE ONLY ALL-PURPOSE SDN CONTROLLER IN THE MARKET

The OpenDaylight consortium was formed in February 2013 as a Linux Foundation project. Numerous networking and software vendors participated in this project, including Cisco, Microsoft, Citrix and Big Switch. Main purpose of which was creating of an open source SDN framework with well-defined northbound API, while providing extensive support of variety of southbound protocols. Consequently Open Daylight Project is expected to form a northbound API “standard” to be used to control network devices via separate southbound protocols: OpenFlow, NETCONF, OVSDB etc.

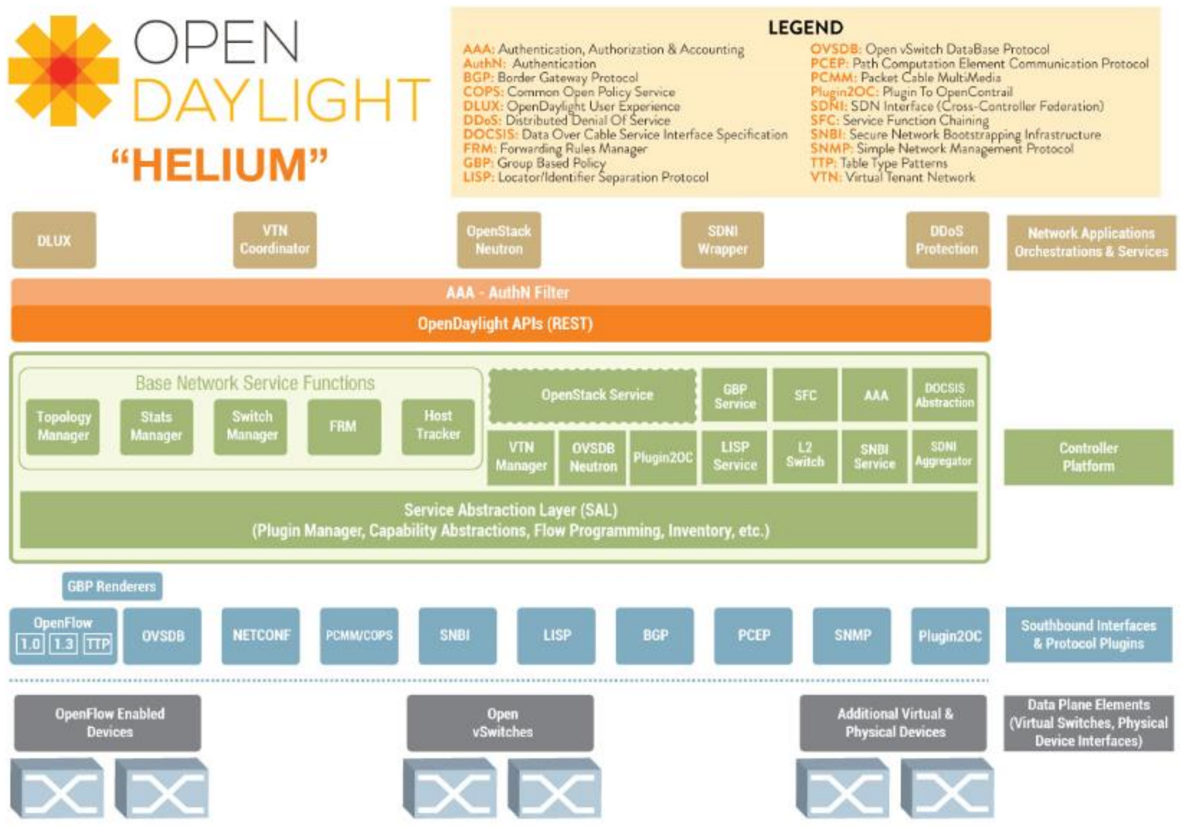

OPEN DAYLIGHT
"HELIUM"
LEGEND
AAA: Authentication, Authorization & Accounting
AuthN: Authentication
BGP: Border Gateway Protocol
COPS: Common Open Policy Service
DLUX: OpenDaylight User Experience
DDoS: Distributed Denial Of Service
DOCSIS: Data Over Cable Service Interface Specification
FRM: Forwarding Rules Manager
GBP: Group Based Policy
LISP: Locator/Identifier Separation Protocol
OVSDB: Open vSwitch DataBase Protocol
PCEP: Path Computation Element Communication Protocol
PCMM: Packet Cable MultiMedia
Plugin2OC: Plugin To OpenContrail
SDNI: SDN Interface (Cross-Controller Federation)
SFC: Service Function Chaining
SNBI: Secure Network Bootstrapping Infrastructure
SNMP: Simple Network Management Protocol
TTP: Table Type Patterns
VTN: Virtual Tenant Network
DLUX
VTN Coordinator
OpenStack Neutron
SDNI Wrapper
DDoS Protection
Network Applications Orchestrations & Services
AAA - AuthN Filter
OpenDaylight APIs (REST)
Base Network Service Functions
Topology Manager
Stats Manager
Switch Manager
FRM
Host Tracker
OpenStack Service
GBP Service
SFC
AAA
DOCSIS Abstraction
VTN Manager
OVSDB Neutron
Plugin2OC
LISP Service
L2 Switch
SNBI Service
SDNI Aggregator
Controller Platform
Service Abstraction Layer (SAL)
(Plugin Manager, Capability Abstractions, Flow Programming, Inventory, etc.)
GBP Renderers
OpenFlow
1.0
1.3
TTP
OVSDB
NETCONF
PCMM/COPS
SNBI
LISP
BGP
PCEP
SNMP
Plugin2OC
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)

Figure 7. OpenDaylight Controller Architecture

Java API in the controller makes possible to use both OpenFlow and non-OpenFlow protocols for management of the network devices.

The supported collection of Southbound protocols has a modular structure and installed as plugins. These modules are linked into Service Abstraction Layer (SAL) that serves an additional abstraction layer between Northbound and Southbound APIs. SAL translates API requests into any supported Southbound API that that is used to control network devices. This implementation gives an advantage of development applications that will be able to work with both OpenFlow and non-Openflow switches with all details of Southbound interface handled by the controller.

Northbound API include OSGi framework [34] and bidirectional REST. OSGi framework is used for applications that will run in the same address space as the Controller while the REST (web based) API is used for Apps that do not run in the same address space or external to the Controller.

The RESTful APIs functions of OpenDaylight can be summarized in following groups:

RESTful API	Description
Topology	Collects and holds topology information that can be used by applications
Flow programmer	Flow tables interpreter that allows to program and change flows on network devices
Host tracker	Holds information about connected hosts
Switch manager	Collects and stores detailed information about active switches in the network
Statistics	Provides an interface to obtain statistics about flows from network devices

OPENCONTRAIL

Along with the announcement of Juniper commercial SDN controller Contrail, an open source alternative called OpenContrail. OpenContrail is an Apache 2.0-licensed project that uses standards-based protocols and provides network virtualization via overlays between virtual nodes in data centers. Similar architectural approach that is used in WMware SDN solutions.

Primary use cases for overlay networking in OpenContrail controller are:

- Cloud Networking that includes private clouds and Infrastructure as a Service (IaaS) that involves multitenancy and virtualization in data centers;
- Network Function Virtualization (NFV) that addresses service provider’s operational challenges and high costs of maintaining vendor-specific hardware deployed in ISP networks.

In every mentioned case data center virtualization is involved that as it was described in first chapter, poses significant problems for relatively static networks. Overlay networking is one of SDN approaches to tackle this problems and it gains popularity among vendors. OpenContrail is design around this concept and uses virtual tunnels (overlay networks) to connect virtual machines across multi-tenant datacenters. Overlay tunneling implies that every virtual network is isolated and uses physical network devices as pure data forwarding fabric with no distinct preferences for underplaying routing architecture.

An architecture of OpenContrail controlled SDN network is depicted in Figure 7. The system consists of two core components: OpenContrail Controller and OpenContrail vRouter.

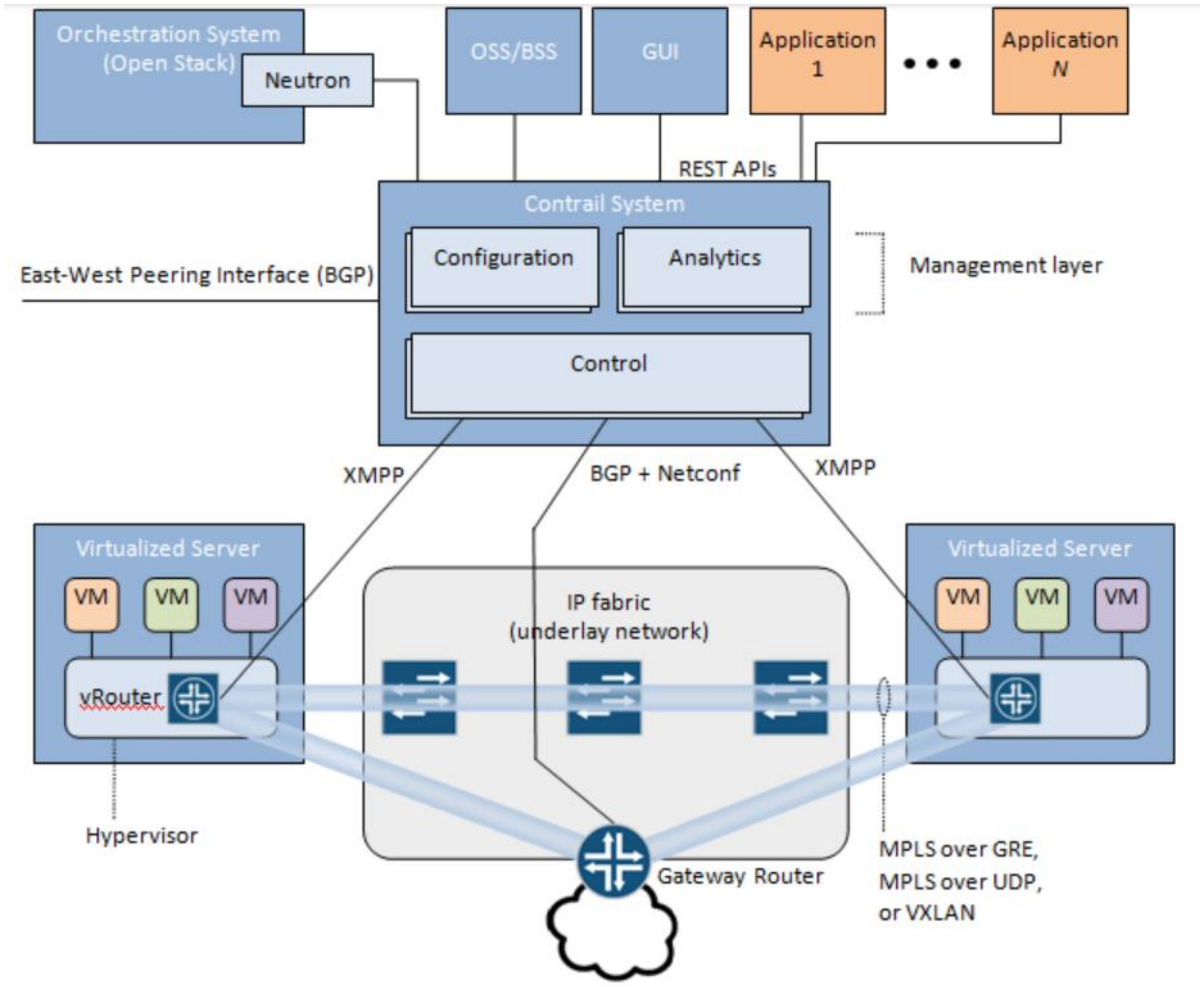


Figure 7. OpenContrail Architecture

The OpenContrail Controller is a logically centralized but may be implemented in a distributed multi-node design. The controller that is responsible for providing the management, control, and analytics functions of the virtualized network.

OpenContrail vRouter represents a tunnel agent that resides in the hypervisor of physical server and contains forwarding information (IP prefixes and MAC addresses) of all VMs in particular server. Basic task of this agent is to encapsulate and decapsulate MPLS/GRE packets received from the overlay network and delivering those to the appropriate VM. Conceptually this agent is similar to OpenFlow switches in classical SDN architecture, although it is controlled via XMPP protocol. For configuration and management of control plane nodes and physical gateways BGP jointly with NETCONF is used.

The RESTful APIs functions of OpenContrail can be summarized in following groups:

RESTful API	Description

Analytics	Provides an API interface to store statistics and counters from network devices
Configuration	Provides an API that basically serves as an interpreter converting API requests into network level commands for network devices configuration
Control	Main orchestration unit of network agents (via XMPP) that utilizes BGP for east/west communication with other logical nodes in distributed design and generates actual configuration commands for network layer devices.

FLOODLIGHT

A WIDELY USED AND WELL SUPPORTED JAVA-BASED CONTROLLER FOR RESEARCH PURPOSES AND OPERATOR NETWORKS

Floodlight controller have become an admired and popular Java controller that has been provided by Big Switch Networks. Floodlight is an open source Apache-licensed OpenFlow controller with wide support community as many core packages are from OpenFlow community.

Floodlight include considerable list of modules and provides both Java and RESTful APIs (depicted in Figure 8). Java APIs are utilized by modular applications that can register listeners and subscribe for specific events in the network to realize different features for traffic processing. This set of APIs is very similar to OpenDaylight and Beacon controllers.

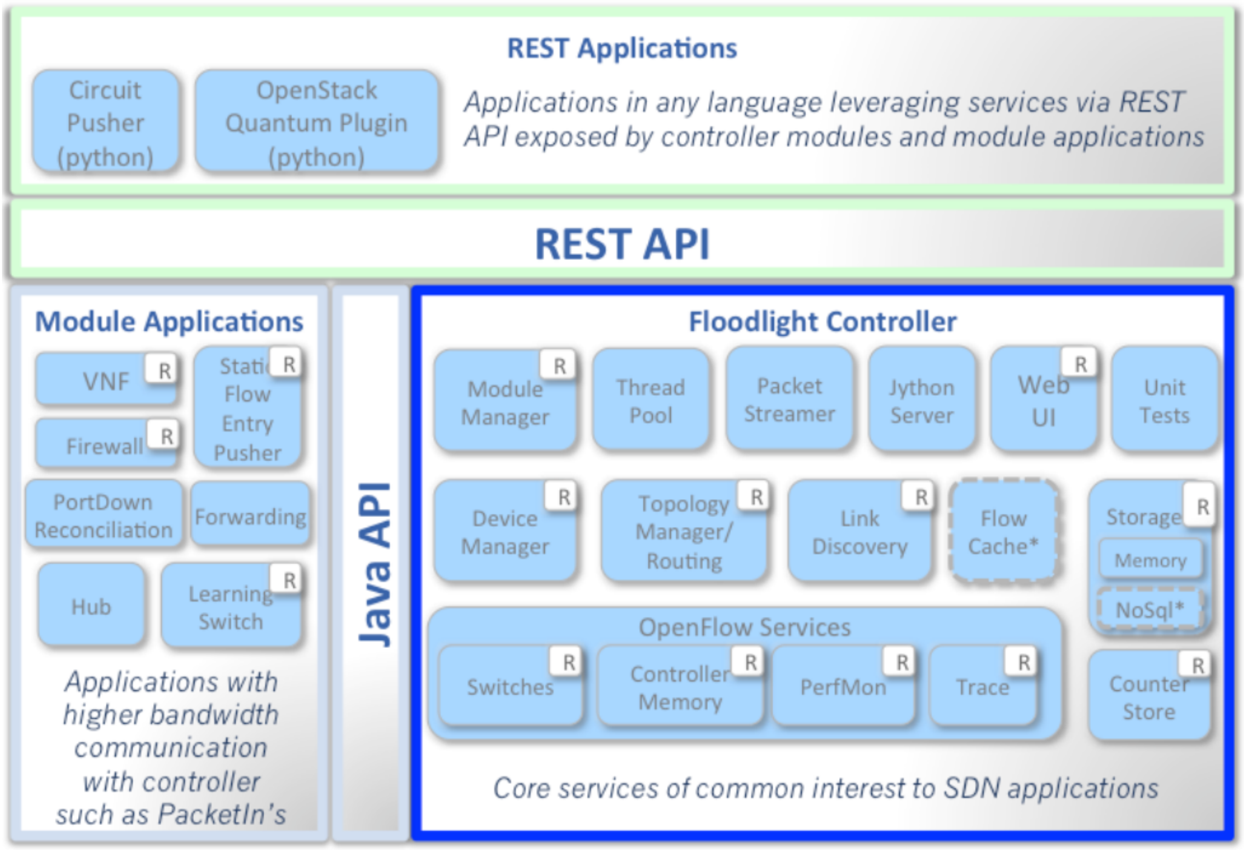


Figure 8. FloodLight Controller Architecture

Internal architecture of Floodlight has modular structure and includes following building blocks:

RESTful API	Description
OF Switches Manager	Manages and interacts with all switches connected to the controller and collects statistic for individual switches
Controller Memory	The Memory Storage Source is an in memory NoSQL style storage source
Device manager	Tracks devices as they move around a network and provides destination device for a flow installation
Topology Manager	Topology Service maintains the topology information for the controller, as well as to find routing in the network via LLDP.
Link Discovery	The link discovery service is responsible for discovering and maintaining the status of links in the OpenFlow network.

RYU

AN OPENSTACK INTEGRATED PYTHON-BASED CONTROLLER WITH SUPPORT OF DIFFERENT SOUTHBOUND PROTOCOLS SUCH AS NETCONF AND OF-CONFIG.

Ryu is open-source networking framework that is often called Network Operating System implying that it is more than just a controller. Ryu is one hundred percent Pyhton-based and supported by NTT labs. Similar to other controllers, Ryu provides APIs exposed to program applications. One of the distinct features of Ryu is variety of supported Southbound protocols: OpenFlow, NetConf, OF-Config.

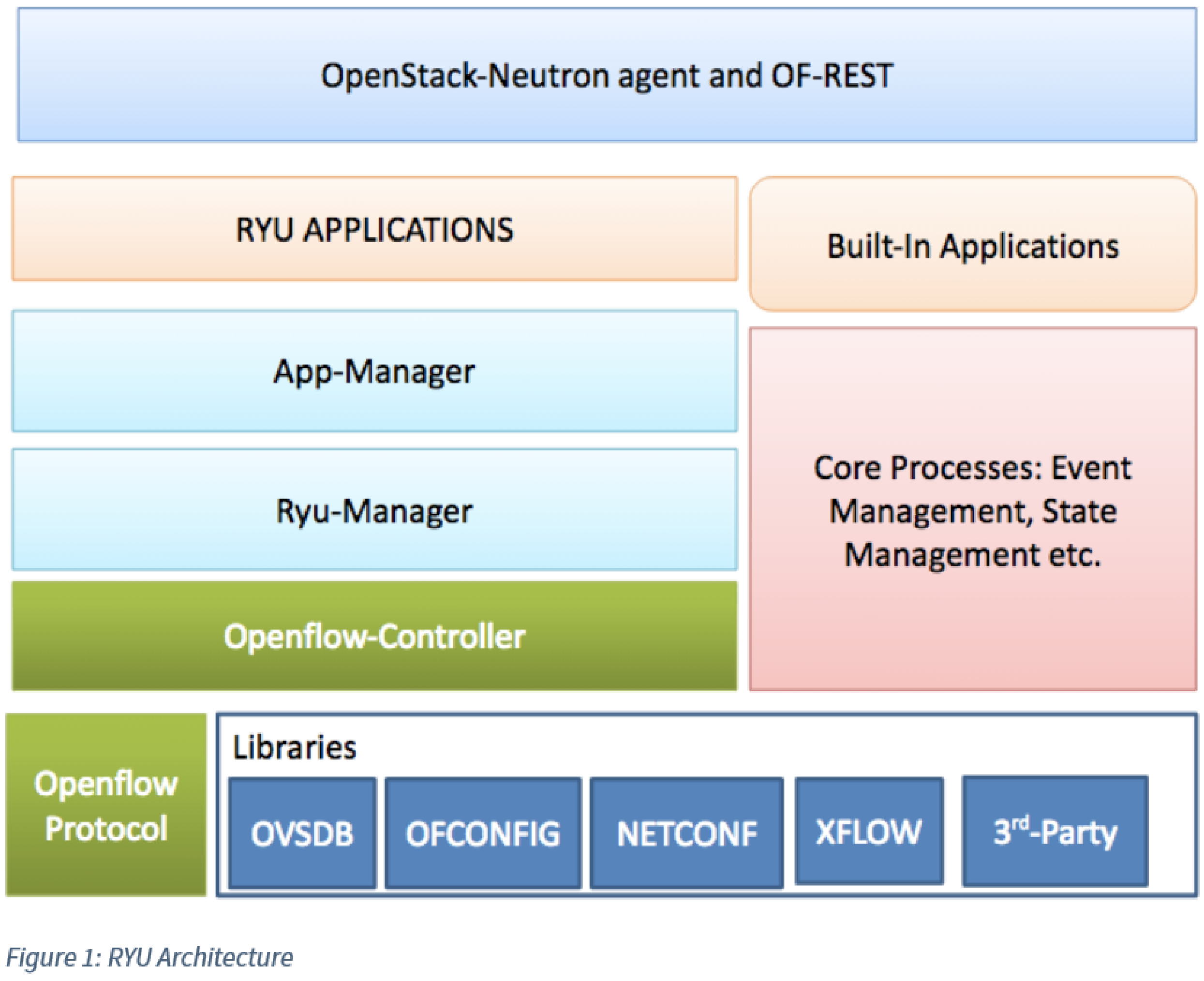


Figure 9. RYU Controller Architecture

As most of the controllers on market, Ryu is modular-based with number of elements (depicted in Figure 9), including infrastructure applications, collection of libraries for Southbound communication, event and application managers etc. Core of the Ryu controller is composed of Ryu manager that handles connections of OpenFlow switches and application manager that is a fundamental class for all Ryu applications.

Ryu APIs represented with Openstack Quantum plug-in [35] that provides GRE and VLAN functionality and REST interface to handle OpenFlow communications with network devices. The latest OpenFlow version 1.4 is supported that one of the great advantages of Ryu framework.

Single-threaded Ryu applications accomplish all high-level features, such as routing, firewalling (Snort-based), GRE and VLAN network abstractions, VRRP etc. Messages exchanged between applications are asynchronous events. Separate event handler thread manages all events for specific application pulling events from application event queue (FIFO type). Synchronous events are generated for inter-application exchange in order to consider other apps states in event management. From the performance standpoint multi-threading gives some performance advantage to Ryu controller, however developers strongly discourage multi-thread implementations for Ryu applications.

BEACON

Beacon is Java-based OpenFlow controller with modular architecture. It can operate in events- and thread-based fashion. Multithreading provides significant advantages comparing to single-threaded controllers (as Ryu). Originated in Stanford University as one of the first experimental implementations for early OpenFlow research and was used in several research projects. It provides impressive performance and stability [33] that makes it extremely preferable choice for research testing. Java-based code provides great cross-platform capabilities as Beacon can run on many different platforms, starting from multi-core Linux servers to even Android phones.

Modular code architecture makes possible independent starts and stops of functional applications with no interruption of other modules (bundles in Beacon terminology). Such flexibility improves research and testing work. Beacon had a huge impact on SDN industry development as significant amount of researches were conducted using Beacon and its code serves as basis for Floodlight controller.

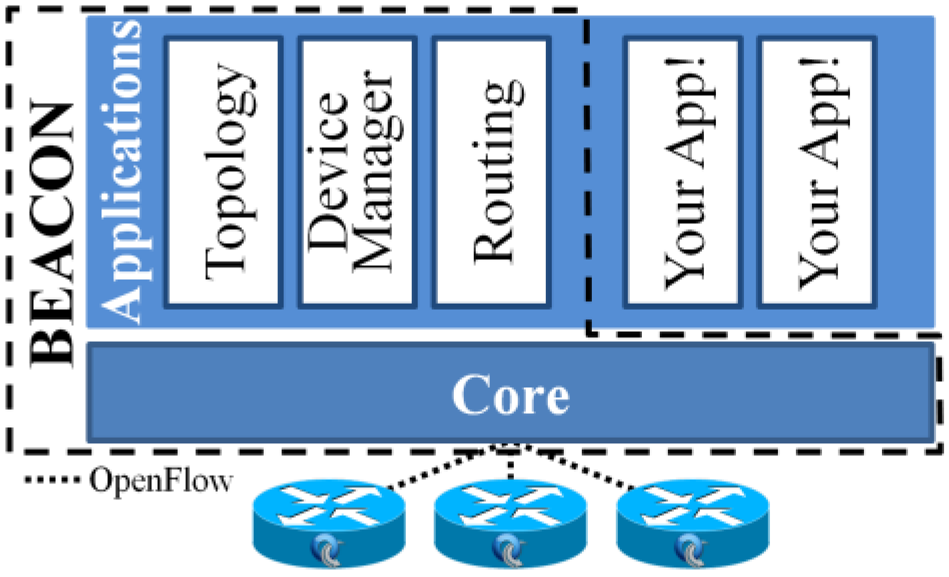


Figure 9. Beacon Controller Architecture

Beacon’s core is main bundle that tights all other bundles together and connects switches. Core publishes IBeaconProvider that operates with OpenFlow messages and applications (Device manager, Topology manager, Routing bundle etc.) register listeners to get notifications about changes in current topology or receive OpenFlow messages. Beacon applications are build upon the core and provide additional APIs:

APIs	Description
Device Manager	Keeps track of network devices and information about their addresses. Provides an interface to receive events about changes in devices states.
Topology	Determines links and interconnections between network devices. Provides an interface for event regarding links state changes.
Routing	Interacts with Device and Topology managers to build shortest L2 paths across the network.
Web	Provides Web interface for Beacon controller

Beacon's API is built to be simple and provide freedom for developers to use any Java libraries and structures. OpenFlow library has been adopted to Java environment and called OpenFlowJ (based on OpenFlow 1.0). It serves as OpenFlow message interpreter and handles message exchanges between controller and network devices.

Employment of Spring library in Beacon controller and its major components: Inversion of Control (IoC) and Web framework provides efficient use of code in controller and application development process. IoC framework improves applications development and helps manage application's interconnections. Web framework improves REST interactions interpreting requests and responses addressed to and from Java objects.

In conclusion, it is worth noting that controller implementation is one of the critical points for the success of SDN. Lots of things need to be evaluated in order to determine controller implementations that are more suitable to specific cases.

- **Internal architecture and modularity** directly impacts ease of development and testing new applications as they can be added and removed from the controller with no affect to other modules of the controller. It also gives additional benefit in performance and flexibility of the platform as instantly required modules can be added for specific time and removed if there is no need in them anymore. It is also worth noting that lack of modularity in the controller architecture may force developers to implement simple network services in each new application.
- **Ease of development.** The programming language and development environment have a significant impact on the productivity of developers, and can also be a limiting factor in application performance. Supported libraries and APIs is another major point that affects development environment. One of the main issues that needs to be addressed in this respect is interoperability and Southbound OpenFlow API was the first to help improve this situation.

- **Performance and stability**. Performance of an OpenFlow controller is typically measured as the number of packet_in events a controller can process and respond to per second and the average time the controller takes to process. Multithreading or multi-instance functionality significantly improves performance, although most of the open-source controllers miss this functionality as opposed to commercial implementations.
- **Scalability** is a major concern for huge deployments. Most of open source controllers are single-threaded while commercial alternatives support distributed implementations and provide distribution mechanisms to run separate instances for fault tolerance. Some of the public controllers in this space include OpenContrail, Big Network Controller and Programmable Flow. Other issues that require to be researched for this purpose are the East/westbound APIs, that can provide hierarchical designs in order to reach scalability, modularity, and security. Coordination between different controllers becomes very important in diverse networking implementations, so APIs standardization would be a huge improvement in order to make it real.

SDN APPLICATIONS

SDN USE CASES

SDN is a complementary technology that brings new ways to implement todays networks. It helps to solve current network problems that have become a limiting factor for IT infrastructure development. Although SDN holds a great promise to improve network efficiency and resolve many management issues SDN production implementations are relatively rare. Mainly, only particular and usually large-scale service providers, like Google or Amazon find it feasible to implement SDN in production deployments. Nevertheless, more and more SDN comes under consideration as a future path for network development in many companies.

SDN IN DATA CENTERS

Applications have become virtualized and complex, demanding networks to be more proactive and responsive to application behaviour. Today's applications are no more attached to physical machine. They are virtualized, dynamic in nature and distributed across the servers. Network infrastructure must adjust to emerging needs of modern applications. Data center networks must be dynamic and easy to manage. Software Defined Network becomes a logical step to satisfy all mentioned requirements.

CHALLENGES IN MODERN DATA CENTERS

Rapid adoption of compute and storage virtualization have had a significant impact on state of operation in today's data centers and networks. Virtual machines can be created or migrated to other physical locations instantly, straining abilities of traditional network infrastructure, as it has to handle these changes according to virtual server environment. Networking protocols are tied to physical ports on devices and can't deal with dynamic nature of computing environment. All this placed additional requirements on network devices:

- **MAC address table sizes.** Network virtualization is the main reason of MAC address explosion. Number of server nodes in one layer two domain increased radically. Each physical machine can potentially serve tens of virtual hosts with separate virtual MAC addresses assigned to every VM;
- **VLAN limitations.** With the advancements of virtualization and introduction of multitenancy, data canters started to expand significantly. As a result 4096 VLANs have become not enough and can be exceeded with no effort;
- **Dynamic computing world.** Virtualization provides a considerable freedom in computing management. Virtual machines can be created, migrated or deleted in few mouse clicks. It is very important that network keep in pace with changes in virtual server infrastructure. Conventional network technologies are fairly complex and slow to make changes in. These processes need to be optimized and automated to be synchronized with server world.

- **Multitenancy.** With virtualization separate data center clients share the same physical resources pool. It is a huge challenge to provide flexible client isolation and to efficiently use network infrastructure. It becomes particularly difficult with large-scale distributed data centers. Large amounts of East-West traffic require a very granular control of bandwidth utilization inside of data-canter and between them.

TUNNELING/OVERLAYS USE CASE

In order to overcome modern data center network problems a specific mechanism had to be developed. The easiest way from implementation prospective would be to create an additional level of abstraction via network virtualization. Tunneling method addresses most of issues that modern data center faces. SDN overlays are application centric and build on top of existing physical infrastructure, hiding forwarding details from applications. Current physical network doesn't require any changes and used as data forwarding plane.

Combination of overlay concept and centralized control by means of controller, gives unprecedented level of flexibility, dynamics and management improvement in network infrastructure. Because all traffic is encapsulated into virtual tunnel and only MAC address of the tunnel (created in hypervisor) will be exposed to network. This reduces number of MAC addresses in the network. New tunneling technique substitutes current VLAN network virtualization method. In this way, VLAN number limitation is not a concern anymore as new MAC-to-IP tunneling protocols such as VXLAN [36] or NVGRE [37] can provide up to 16 millions segments inside the tunnel. High-level design of tunneling architecture is depicted in Figure 10.

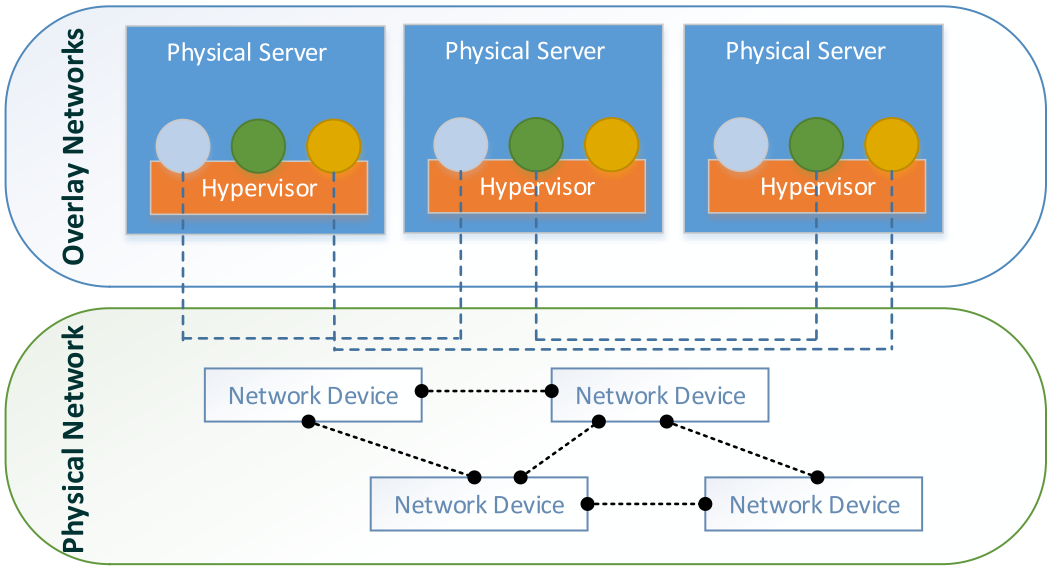
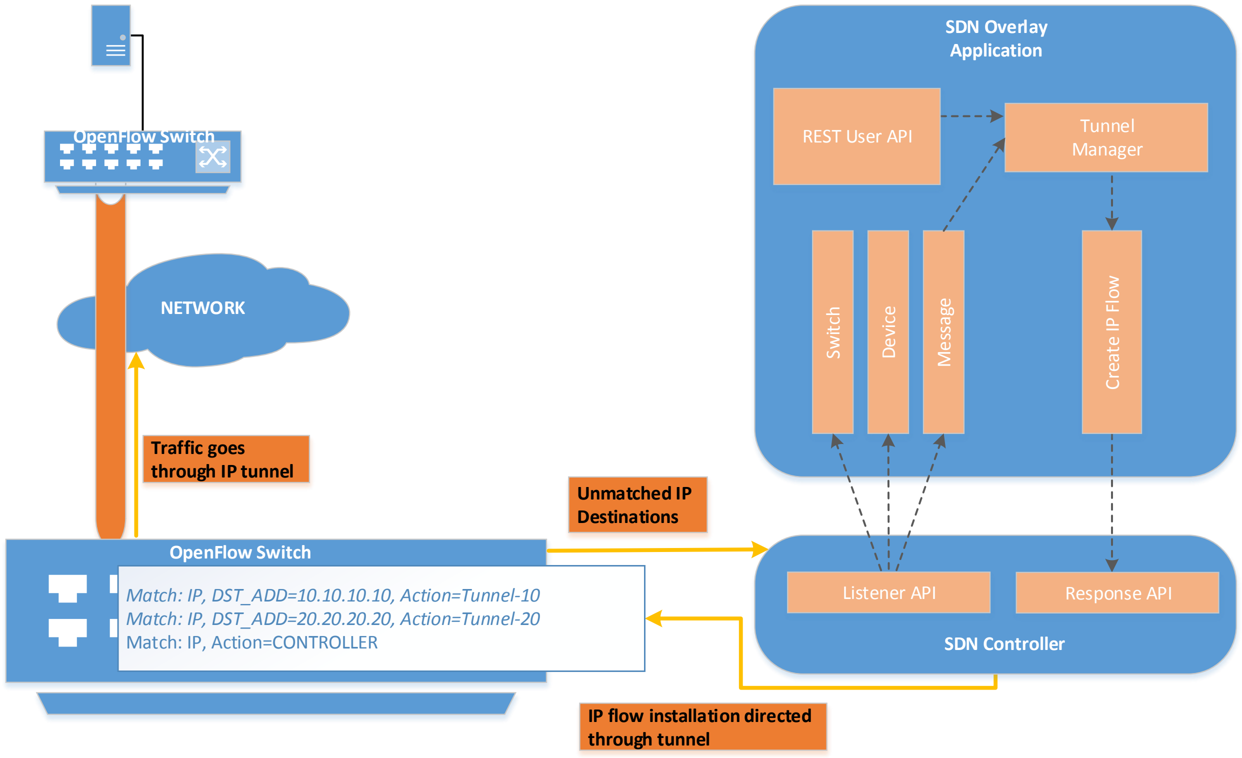


Figure 10. Network Virtualization

Actual network traffic is forwarded inside of the tunnel and logically abstracted form physical infrastructure. Server hypervisor encapsulates virtual machine traffic and emulates a physical connection between different VMs on separate servers. All the details of underlying infrastructure are hidden form hosts. Tunnel terminating entities (usually hypervisors) exercise full control of the tunnels and called virtual tunnel endpoints (VTEP). All originating packets are encapsulated based on rules predefined by the controller.



Match: IP, DST_ADD=10.10.10.10, Action=Tunnel-10
Match: IP, DST_ADD=20.20.20.20, Action=Tunnel-20
Match: IP, Action=CONTROLLER

Figure 11. SDN Tunnels application design

A high level implementation design is depicted at Figure 11. Overlay application runs on the controller and responsible for construction and control of overlay tunnels. It maintains hosts and tunnel databases in order to provide tunnel-to-host associations. Every incoming packet that has no flow entry in the switch is forwarded to the controller and handled by overlay application that listen for incoming packet events. Since controller has a global view of the network it (overlay application) determines the switch with destination endpoint connected. OpenFlow networking device creates a tunnel and installs a flow entry from the controller, that maps a particular flow to the specific tunnel towards destination.

DEPLOYMENT CONSIDERATIONS

Implementation of overlay network is based on several main design principles:

Forwarding method. Classical overlay approach relies on existing networking mechanisms to forward packets. Thus, overlay network implementation will be as reliable as underlying infrastructure. Throughput and performance of virtual overlay network is also depends from undelaying forwarding plane, for the most part;

Mapping database synchronization. As a central logical unit, controller collects all the information about network elements, including hosts. Overlay control application that resides on the controller maintains and correlates hosts and tunnels databases to keep live mapping. This mapping should be updated instantly and be kept as accurate as possible. This aspect should be well considered in case of physically distributed data centers. Controllers should be placed in strict hierarchy and with precisely tuned update mechanism in order to avoid database discrepancies. Database inconsistencies will likely cause incorrect packet forwarding and as a result packet drops.

Encapsulation protocols. Traffic encapsulation places an additional workload on the server hypervisor and must be considered in overlay networking design. Software encapsulation takes server CPU cycles and may decrease throughput characteristics as well as adding latency. On the other hand, this approach distributes encapsulation load across physical servers in the data center and does not load network equipment that is significantly limited in performance comparing to modern servers. There are three major network virtualization protocols: VXLAN, NVGRE and STT. All of them are fairly similar architecturally but each has individual benefits:

- VXLAN is widely supported across vendors and will be an advantageous option in multi-vendor environment. It is worth mentioning, that this relates to tunnel terminating software/hardware, not networking equipment that is simply forwarding encapsulated traffic;
- NVGRE makes use of 24 bit of the GRE key in order to separate tenants inside of the tunnel. Comparing to VXLAN it promises to be more scalable as there is no flooding MAC address learning. As regards performance, initial lack of hardware offload on NICs have diminished performance and popularity of this protocol;
- STT is the most recent encapsulation method and was designed to be compatible with the most of today’s network interface cards (NICs). This compatibility allows to offload encapsulation process to hardware of NIC. That gives a great advantage for STT and given general absence of multi-vendor issues makes it a most preferable choice.

KEY BENEFITS

To summarize, SDN using overlay tunnels is mainly designed for virtualized data center environments. It helps to overcome current data center networking problems:

- **MAC address table issues.** Tunnels hide real machine MAC addresses;

- **VLAN limitations**. Tunneling mechanisms such as VXLAN and NVGRE are specifically designed for MAC-to-IP encapsulation with VLAN limitations in mind and can perfectly substitute VLANs;
- **Scalability and flexibility of the network.** Since tunnels can be build instantly and based on changes in computing environment, it solves the problem of relatively static traditional network.
- **Performance.** Comparing to traditional methods of encapsulation when one network node encapsulates traffic for number of sources, overlay encapsulation load is distributed and is handled individually by the endpoint (server hypervisor, hardware OpenFlow switch etc.)
- **Deployment.** Deployment and migration is less disruptive than complete SDN solutions. It can be considered as a first step towards SDN implementation in the network.

DRAWBACKS

Nevertheless network overlaying addresses a variety of data center network issues, it does not show itself as a complete SDN solution that promises to make network to be completely programmable. To specify these issues:

- **Path visibility.** Encapsulation of endpoint traffic at the very edge decrease overall visibility of the network and brings an additional layer of complexity as end-to-end communication is hidden from physical path.
- **Management issues.** Network elements still require traditional methods of configuration and control;
- **Troubleshooting complexity.** Virtualization of networks increase troubleshooting complexity because of intricate mapping database that needs to be examined in details;
- **Network agility.** Since overlay virtual networks do not deal with physical infrastructure, problems of convergence in case of links failure and efficient bandwidth utilization remain unaddressed;
- **Innovation limitations.** With implementation of overlays, control plane remains untouched and thus network data plane is still confined with all of the limitations of traditional networks.

SDN IN ENTERPRISE NETWORKS

Enterprise networks are groups of geographically distributed local networks that serve for data exchange and for provisioning of different services in order to improve business efficiency. Users access the network via wired or wireless mediums, as well as with number of different devices such as desktops, notebooks or mobile devices. These endpoints may be shared, issued by enterprise or personal. Furthermore, present mobile devices functionality imposes new demands to the network infrastructure that must provide access to services and data from any location and at any time.

ENTERPRISE NETWORK NEEDS

Increased user mobility needs as well as inherent desire to employ personal mobile devices for work purposes (BYOD) have raised new challenges to enterprise networks:

- Transparent deployment of new applications, as well as network reconfiguration for new services;
- Consistent contextual policy application across the network based on number of criteria (user profile, device type, requested application, time and location);
- Simple and convenient methods of control and monitoring across wired and wireless networks.

Additionally to new mobility demands, a number of fundamental issues has been present in enterprise networks:

- **Operational complexity.** Large scale enterprise networks face serious challenges in configuration consistency and prompt configuration adjustments for new services deployments or links fails;
- **Vendor dependency.** Features and configuration tools rollout completely depends on vendors and their software update cycle;
- **End-to-end visibility.** Traditional three-tier network design provide limited network visibility that complicates troubleshooting and monitoring, especially in case of wireless access that is typically deployed in overlay fashion;
- **Network merging.** Significant operational and technical challenges in case of organizations merging [38].

Software Defined Network flow-based approach promises to simplify network management, traffic isolation and bring holistic network visibility.

TRAFFIC ISOLATION USE CASE

Large scale organisations have a great number of departments with diverse functions, as well as temporary contractors and alliances with other enterprises. This variety of business units have different data access requirements, security policies and possibly overlapping addressing schemes, in case of alliances or enterprise acquisitions. Every case requires a logical partitioning of the physical network in addition to well-defined and consistent security policies across organisation’s network. Traditional solutions as VLANs, VRFs and MPLS VPNs address this challenges providing isolation on its own scale. Deployment and especially management is error-prone and mostly manual hence not flexible and time-consuming.

In today’s networks a centralized policy management system, similar to today’s network access control solutions (NAC) addresses these goals. Combining this idea with SDN’s flexibility to control flows makes it a very suitable and beneficial solution for traffic isolation based on policies. In general, policy implementation can be described as follows:

- As user connects the access will be limited and all initial requests are sent to the controller;
- Controller runs a specific application that maintains preprogrammed policy database. This application makes decision regarding every new users and provides access policies to the controller;
- As appropriate policy was found, the controller installs corresponding flow entries to the edge device.
- All consequent user traffic will be handled based on user-specific flow rules. In case of new requests that are not covered by these flows, the whole process will be repeated resulting in installation of new flows or modification of existing ones.

OpenFlow edge devices serve as policy enforcement elements. These devices implement logical user separation and control over logical networks separation. Policies define allowed applications granularly limiting access to network resources. For this purpose, it is critical to consider table sizes of edge devices as each user will have specific flow entry enforcing policy-based access. In some cases, more than one flow entry will be required.

Policy enforcement at the distribution layer is based on broader definitions as traffic classes or some aggregated values. It is not practical from architectural prospective to implement end-user policy flow rules at higher layers (distribution, core).

A high-level access control design is depicted in Figure 12. Initially, as user connects to the network only very limited access will be provided: ARP and DNS requests. Default specific flow entries will be installed to network devices in order to forward DHCP request to the controller. DHCP message exchange is used to build MAC-to-IP address mapping for all connected devices.

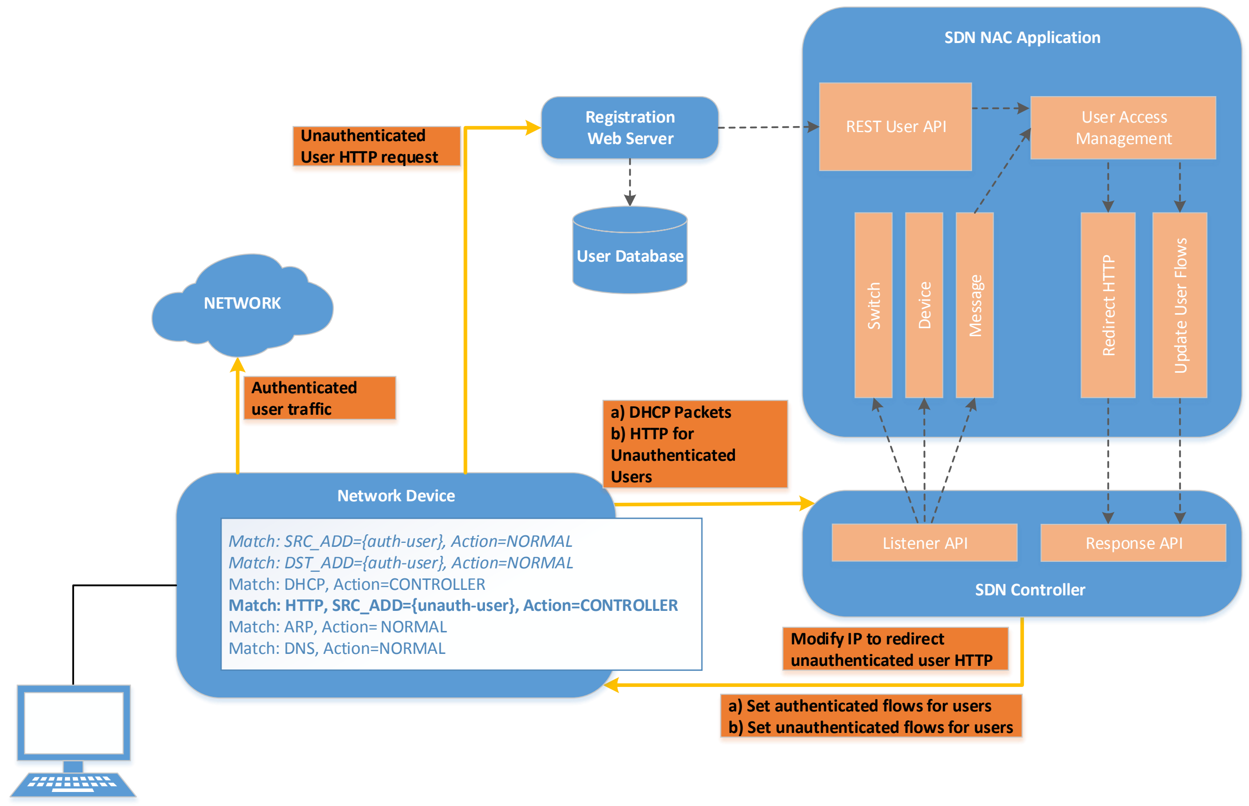

SDN NAC Application
REST User API
User Access Management
Switch
Device
Message
Redirect HTTP
Update User Flows
Registration Web Server
User Database
Unauthenticated User HTTP request
NETWORK
Authenticated user traffic
a) DHCP Packets
b) HTTP for Unauthenticated Users
Network Device
Match: SRC_ADD={auth-user}, Action=NORMAL
Match: DST_ADD={auth-user}, Action=NORMAL
Match: DHCP, Action=CONTROLLER
Match: HTTP, SRC_ADD={unauth-user}, Action=CONTROLLER
Match: ARP, Action= NORMAL
Match: DNS, Action=NORMAL
Listener API
Response API
SDN Controller
Modify IP to redirect unauthenticated user HTTP
a) Set authenticated flows for users
b) Set unauthenticated flows for users

Figure 12. SDN Network Access Control Design

Based on DHCP reply, the user state will be determined. In case of new user, the appropriate record will be added to the database indicating that it is unauthenticated user, along with the appropriate flow entry on the edge switch.

In order to process user authentication the most relevant solution is to build a captive web-portal that will handle authentication, registration and other administrative procedures to properly define user policies. To make it work, all HTTP requests from unauthenticated users should be redirected to captive portal. The controller installs an individual flow entry (based on user source MAC address) to redirect HTTP requests to itself. It is done for IP/MAC address modification purposes, as all HTTP traffic should be forwarded to the portal and not to the controller. The controller serves as HTTP forwarding device, modifying destination fields of initial packets.

No other traffic is allowed to flow, apart from HTTP before the user is authenticated. Thus, the user can’t get access to any part of the network unless all required information (user credentials, registration information) is provided vis captive portal. When all checks complete the captive portal notifies the NAC application about results of authentication. In case of successful registration, all temporary redirection flow entries are deleted or modified to precise rules based on policy assigned to the user. These rules will remain valid till preconfigured timeout, thus the registered user will have access to the network in case of later reconnection.

The same approach can be used to address BYOD trends [39]. In the same way more advanced capabilities can be implemented such as endpoint posture assessment (local machine firewall policies, anti-viruse updates etc.) before allowing access to the network. Captive portal might be used for employees to provision their own devices for use in corporate network. All security checks and policy application might be automated, thus IT personnel will not be involved in this procedure.

Traditional BYOD implementation involves a complex deployment and configuration process. Edge devices software must support redirection features in order to forward initial user requests to captive portal for registration and device assessment. In case of multi-vendor environment or limited functionality of deployed switches, installation of specialized network nodes was required for policy enforcement. In large scale networks with thousands of access switches reconfiguration of all access devices is required.

DEPLOYMENT CONSIDERATIONS

Controller performance. Since all registrations are performed through controller application it might experience moments of significant load, especially in the mornings when all employee turn on they work stations. It is very important estimate approximate load peaks and consider connections per second rate that has to be processed in particular environment. Load distribution strategies as load balancers, distributed deployment designs, role separations etc. are highly recommended for reliable deployments.

Distributed deployments. High availability, geographically dispersed networks are things that have to be taken into account in deployment scenarios. Database synchronization over WAN connections including latency and possibility of link outages have to studied in detail for successful deployments. Well planned role distribution (policy enforcement, configuration, monitoring) will improve effectiveness and reliability of final solution.

Authentication mechanism. OpenFlow tunneling vs redirection. Redirection might be preferable for migrations. There are at least two ways to design authentication mechanism. First is based on OpenFlow capability to send a packet from the controller to a specific port of a switch. In this way all responses to 802.1x, DHCP and other requests from the clients will be crafted by applications on the controller. It is a convenient way because all functionality is centralized. On the other hand, this method may be troublesome in implementation and application development. An alternative way is to have all authentication and captive portal functionality on separate machines and just to redirect all traffic form endpoints to provide specific service and obtain results via RESTful API of the controller. In that way a chain of service can be implemented, that reside on physically separated processing units. This approach is convenient on early stages of migration to SDN-based networks. But might turn out difficult in troubleshooting.

BENEFITS

SDN-based BYOD solution greatly simplifies deployment and configuration:

- All edge devices are under control of the controller via OpenFlow. Any additional functionality such as HTTP redirections, can be implemented as additional controller application.
- No need in specific software updates as all advanced features are implemented on the controller.
- New functionality can be added in centralized manner developing specific controller applications. Since policy enforcement function is handled by OpenFlow edge switches and no intermediate devices required. It minimizes changes in the network topology.
- SDN-based network access minimizes vendor dependence;
- Proactive flows installations help to minimize flow modification rate, making solution more efficient;

DRAWBACKS

Nevertheless SDN-based network access solution seems to overcome issues in traditional approaches it may draw some problematics:

- Large scale network implementations require a very intense design thinking;
- The edge device flow tables increase in size limiting an edge device platform choice;
- Advanced BYOD features implementation requires a significant programming effort.

HYBRID WAN

Enterprise connectivity requirements have evolved and single MPLS WAN connections are not enough to connect branch sites to the data center:

- Although traditional WAN connections to data centers are still in need, today’s branch sites now require connections to number of external cloud services via Internet connection. These cloud services may be permanent to improve business effectiveness or temporary dedicated for specific project activity;
- Enterprise network access means are shifting to mobile systems and primarily remote access (mobile phones, tablets with VPN access via Internet).

As a result, enterprise WAN connectivity has evolved into a collection of individual access solutions that might have a temporary nature, per specific projects. Internet, MPLS, mobile VPN edge units, as well as public cloud services. Figure 13 illustrates these elements.

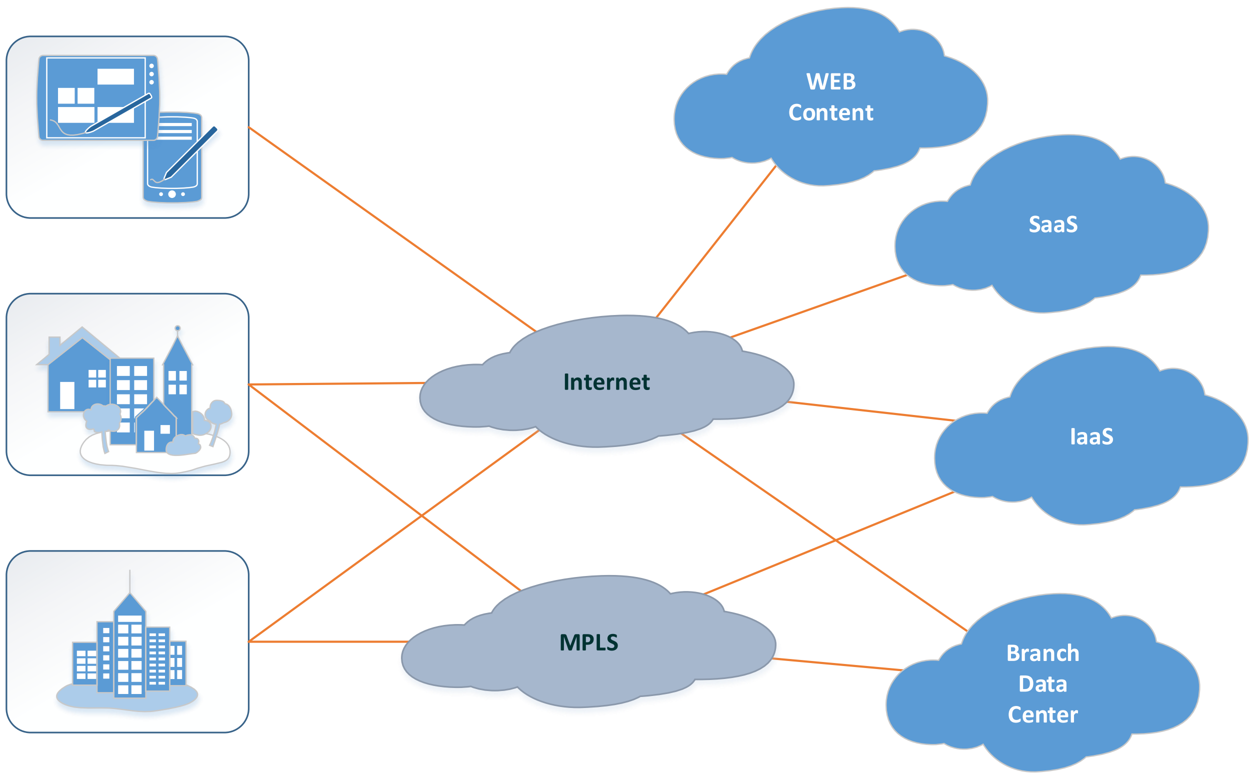


Figure 13. Traditional WAN Architecture

This solutions separation creates a problem that enterprise has multiple individual solutions with no holistic approach to manage the infrastructure. It causes difficulties to ensure consistent end-to-end performance for applications.

In order to avoid these issues, network WAN architecture designing process should focus more on application performance and their flows in the WAN. Otherwise, incomplete WAN designs will cause poor application performance and high latencies. In the long run it causes employees dissatisfaction and productivity drop.

Form technical perspective, access methods separation leads to limited application performance visibility and to troubleshooting difficulties.

HIGH-LEVEL DESIGN

ARCHITECTURE

Hybrid WAN design presumes that in the distributed enterprise WAN network, traffic can freely traverse a hybrid network and is forwarded in a way to optimize performance and total costs.

Enterprise sites will be connected to both MPLS and Internet services and a central control unit decides the actual traffic path, based on preconfigured priorities. End-to-end application communication can go through MPLS network, Internet or both, depending from current network conditions and application requirements.

Proposed hybrid WAN architecture based on SDN will allow to integrate all network access elements together and to create an abstraction of WAN backbone, hiding details about actual transport links. It shares some similarity with Google production implementation of SDN [40] with additional high-level modifications for enterprise environment. Custom designed SDN applications will monitor traffic flows and steer them according to required performance levels or required advanced services (firewalling, IPS, monitoring etc.) defined by IT personnel. Software defined forwarding engine is intended to improve application performance, provide centralized means of WAN control, optimize link utilization, elastic service provisioning and control. This is illustrated in Figure 14.

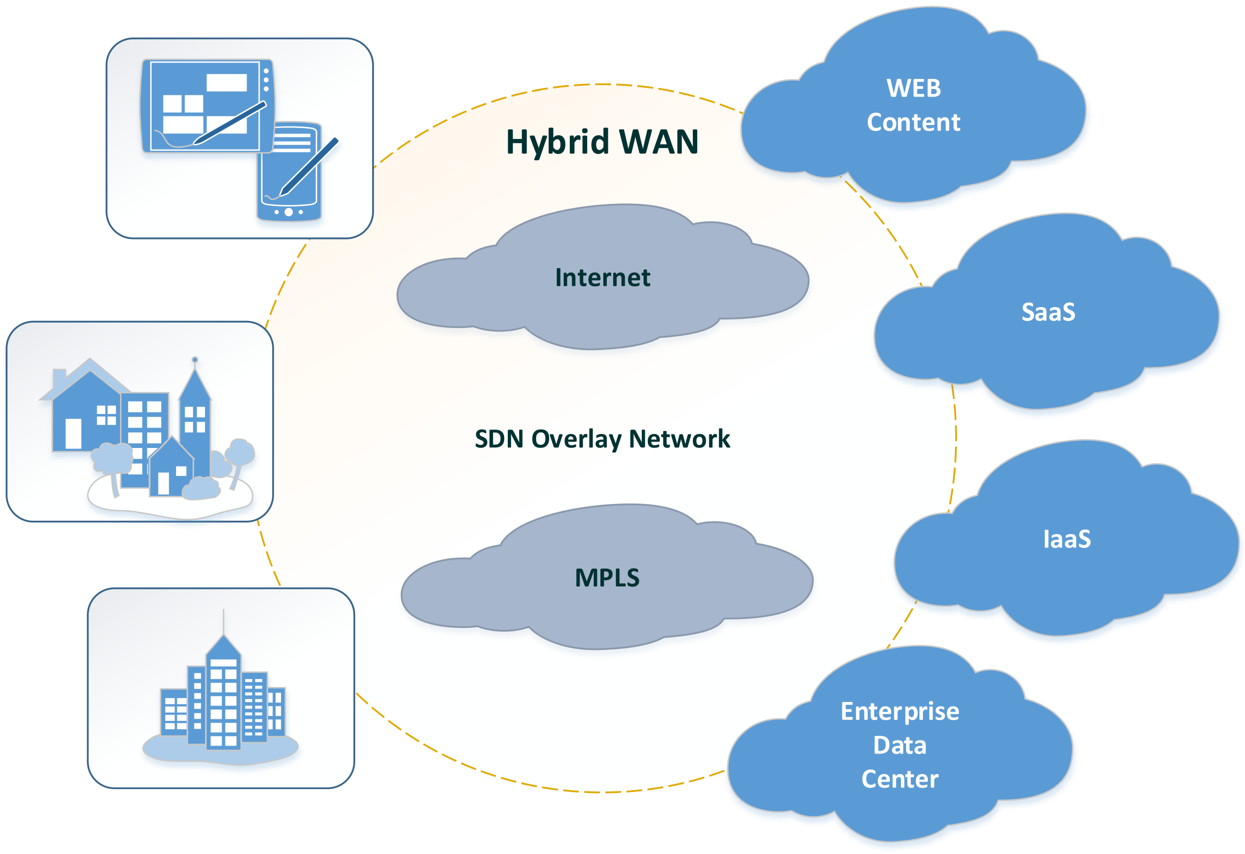


Figure 14. SDN Hybrid-WAN Overlay

Ultimate goal of this architecture is to improve application end-to-end performance, centralize the control of WAN network as vendor-independent backbone, that can dynamically steer the traffic according to administrative policies and network conditions. Only open-source components will be recommended for architecture implementation.

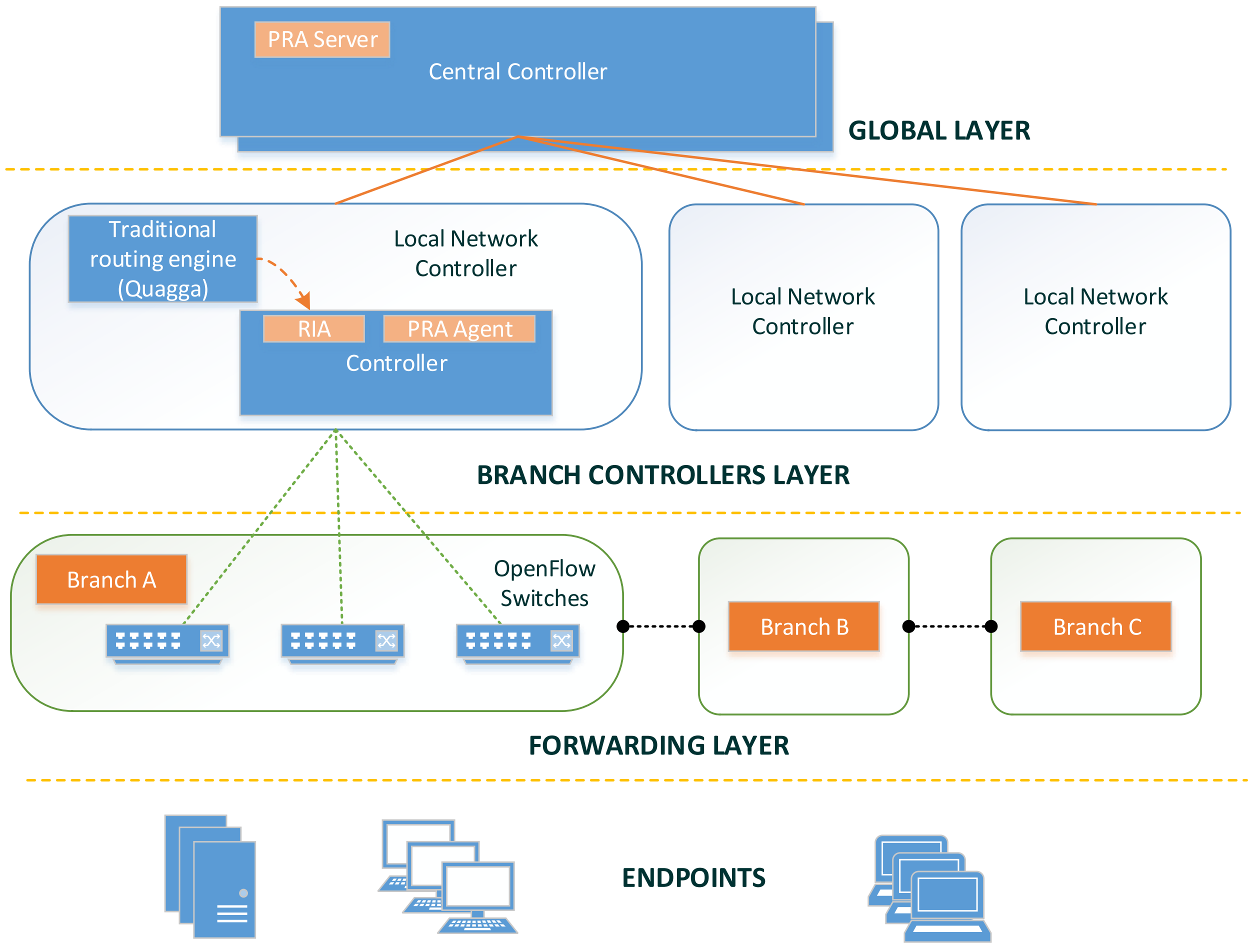


Figure 15. Hybrid WAN control architecture

Hybrid WAN architecture is logically divided in three layers (Figure 15). Global layer consist of central controller with global SDN applications that collect link metrics from branch sites, perform central monitoring and replicate centralized application requirement policies. It creates a topology abstraction and calculates optimal paths. For redundancy purposes, this layer may be mirrored across several geographical separate sites. In that case a specific priority mechanism must be implemented.

Branch controllers layer consist of local network controller (LNC) with SDN performance routing application (PRA) that is responsible for routing decisions at the corresponding WAN site. Forwarding layer is a collection of OpenFlow controlled devices that forward locally originated traffic from users based on local controller defined flow entries.

Local network controllers maintain network state, providing distributed routing and centralized performance routing as an additional constrain for forwarding decisions. PRA will maintain a current network state and modifies LNC forwarding logic based on that.

For migration purposes each branch has to support existing set of routing protocols (OSPF, IS-IS, BGP). Migrating this functionality to the central location with remote OpenFlow control may cause reliability issues as well as integration problems with existing routing protocols. Each branch site will use a desired routing protocol over MPLS or Internet connections.

Performance routing application operates as an overlay independent service on top of existing routing protocols. This separation allows to make independent troubleshooting and services updates. In addition, it provides flexibility to disable performance routing functionality as desired with no impact on shortest path routing protocols.

Forwarding layer consists of OpenFlow forwarding devices implemented in traditional SDN fashion. The main purpose of these switches is to hold flow tables and to forward traffic accordingly.

All routing decisions for every particular flow will be based on predefined policies of particular enterprise. Main policies criteria:

- Type of application (video, voice, databases) and it’s delay sensitivity, data security concerns;
- Applications bandwidth requirements will impact the path of the flow (MPLS or Internet). Administrator may wish to define a lower and upper boundaries of allowed bandwidth per specific application flow. According to these characteristics, traffic can be dynamically rerouted via appropriate WAN connection;
- Time of the day may be taken into account to adjust forwarding decisions.

In contrast to Internet, MPLS connections are more reliable and SLA-driven. Nevertheless, Internet is a cheaper communication and most of the time can provide predictable connection characteristics. Because of that offloading of expensive MPLS WAN traffic is one of the goals of hybrid-wan design. The design presumes overlay tunneling over the Internet as a valid and flexible solution to traverse the traffic from one branch to another. As it was mentioned before VXLAN, NVGRE or STT are possible options for tunneling mechanism. Every flow that has been approved by central controller to be offloaded will be put in the appropriate tunnel terminated by endpoint attached OpenFlow forwarding devices (Figure 16).

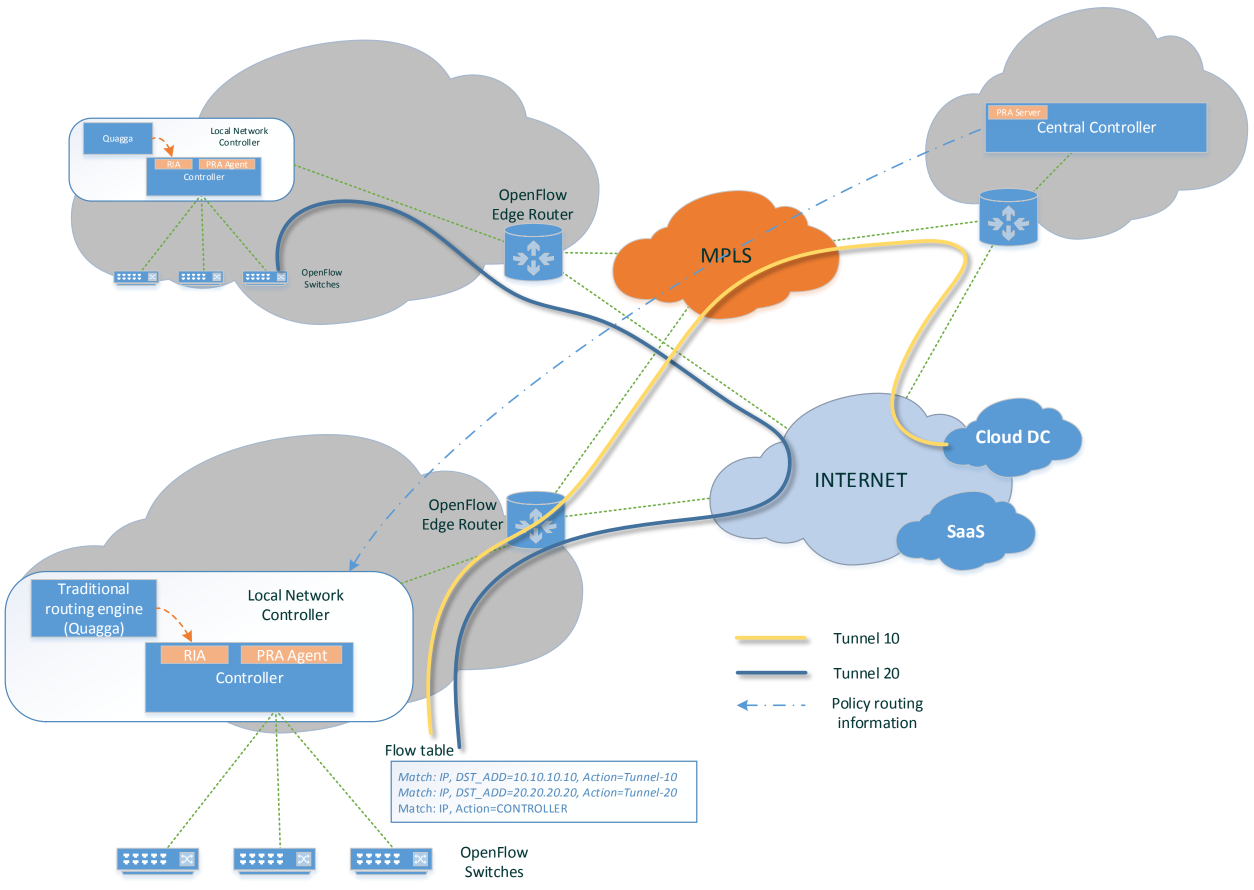


Figure 16. Policy tunnel rerouting

Service chaining will be well suited for hybrid wan solution for offloaded traffic flows. As this traffic runs through unprotected medium, basic security measures have to be taken to secure the traffic. Using overlaying with centralized controller, every flow in the network can be steered to specific service devices (firewalls, IPSs, application optimization appliances etc.) This approach is an extremely easy way to implement service chaining, not confined to specific site – traffic can pass through (and processed by corresponding VMs that run required services) several geographically separated data centers on the its way to destination.

With the adoption of numbers of cloud services, all enterprises are heavily rely on Internet access quality and reliability. With centralized software defined WAN architecture an individual branch SaaS service access can be rerouted over the MPLS WAN to the other branch with better connection characteristics in order to increase performance of application.

APPLICATIONS

Network Control Function

Local network controller implements most functionality on branch sites modifying flow tables of forwarding layer devices (OpenFlow switches). A Linux machine with proper technical characteristic will serve for this purpose.

Internally it consists of separate modules with specific functions:

- SDN Controller maintains a current network state with respect to load performance information, link utilization and optimized application paths. OpenFlow access devices are registered to the controller and receive instant updates about topology changes in form of OpenFlow flow entries for required destinations;
- Routing protocols module. A control application with a collection of supported routing protocols for traditional routing mechanisms (OSPF, IS-IS, BGP);
- Performance routing application. SDN application that runs on the controller and collects WAN links metric information with administratively defined application traffic policies from global layer (controller);
- Routing information adapter (RIA). A specialized instruction translator to implement interoperability between routing protocol module and controller.

OpenFlow serves as Southbound control protocol that is preferably implemented in out-of-band fashion.

Routing

For the purpose of integration with existing protocols, a local network controller must support traditional distributed routing protocols. It can be implemented as a separate application on the local network controller. Nowadays, the tool most used for dynamic routing in Linux is Quagga. It allows system administrators to implement, with a relatively low-cost Linux server, the same functionality that is provided by powerful (and costly) Cisco routers. The tool itself does not handle the routing, but rather modifies the kernel routing table as it learns new best routes to handle packets.

To integrate quagga with the local SDN controller module a separate application has to be developed to build connectivity between Quagga and OpenFlow devices: routing information adapter (RIA). Since all routing and interface status updates will be received on OpenFlow switches, routing interface adaptor will serve as a bride between data plane and routing application (Quagga).

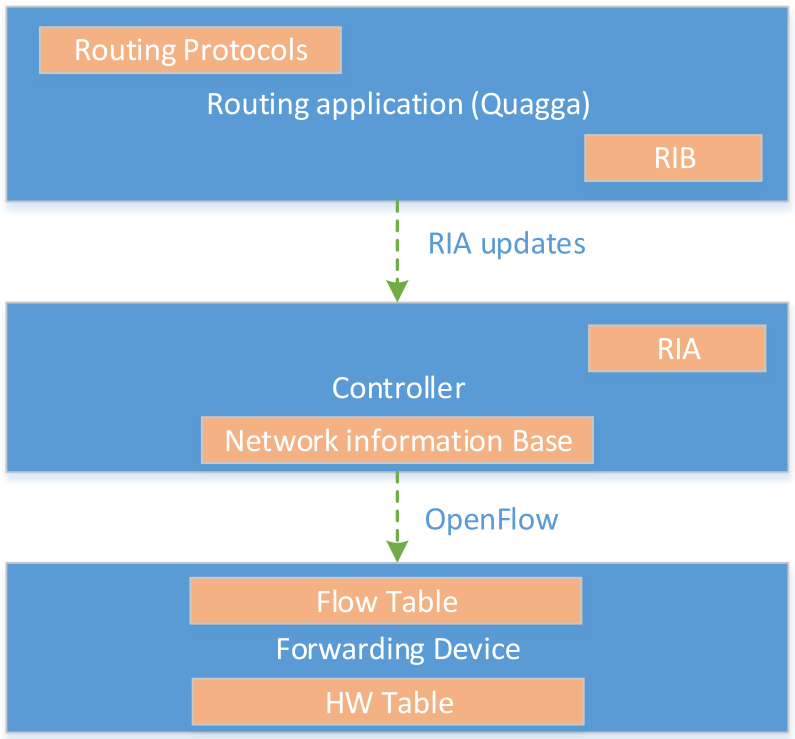


Figure 17. Routing updates installation

RIA catches updates generated from Quagga component and translates them into acceptable instructions for the controller. The controller in one's turn will update own routing process with new interface mappings and generates corresponding flow tables updates for the connected OpenFlow devices. The communication process is illustrated in Figure 17.

Performance Routing Application.

The core of forwarding optimization relies on this application that maintains network metrics (latency, jitter, drops), application priorities defined by administrator. The application consists of main application on central controller with global view of the network and performance routing agents located on branch local network controller. Global application orchestrates all routing decisions with respect to policies and current network conditions and updates local controllers with global changes in the network.

The very first packet form every new flow that is received on OpenFlow device will be forwarded to the local controller. PRA will subscribe to incoming packets function in the controller and makes a final decision about new flow entry installation. Based on actual destination of the packet PRA will request a global controller about current Internet link utilization at the destination and will also start probing the Internet connection to the destination. If the current connection metrics, link bandwidth quotas or application priority settings satisfy all the conditions, a dynamic tunnel (VXLAN, NVGRE) to the requested destination will be created.

Local controller will update OpenFlow devices table entries for this specific flow. The priority between MPLS and Internet connections has to be preconfigured by administrator based on type of application, time, sensitivity of data and network performance requirements.

Performance routing flow entries will have a higher priority than regular shortest path flows. Thus, they will be hit first and traffic will follow an optimized path

SOFTWARE

Software defines features and capabilities of the solution as a whole. SDN software is developed by opensource communities as well as by number of vendors that implement their own proprietary features. The main focus of this solution is to be based on open source software in order to be vendor-independent and fully customized.

SWITCH IMPLEMENTATIONS

There are two of the most mature open source SDN switch implementations available:

- **Open vSwitch (OVS).** A multilayer SDN switch developed with variety of programming capabilities for SDN-based implementations. It is a production-grade switch that supports OpenFlow and conventional management APIs;
- **Indigo**. Is a lightweight and high performance Linux compatible virtual switch that enables support of OpenFlow on physical ASIC-based switches.

Open source SDN switches can be implemented in the hardware and software-based environments. This characteristic makes them a versatile and flexible solution for number of architectures.

OVS is probably the most complete OpenFlow switch implementation. This software has been used as a switching logic in production hardware switches as well as virtual switching entities between virtual machines. Open vSwitch supports OpenFlow for flows manipulation and OVSDB to apply configurations. OVS software has been used for number of switching platforms and is adapted to different switch ASICs.

Indigo switch software is an open source project in order to promote OpenFlow implementation on physical switches, but it can also work in virtual environments. Big Switch Networks have released it to open source community with the public license. Indigo switch software can turn a regular switch to OpenFlow switching device. The performance of Indigo switch is basically line rate because of low-level integration with hardware of the switch (ASIC). Comparing to OVS, Indigo switching software is limited to only OpenFlow support. One additional thing that plays in favor of OVS is that it has better community support and broader set of virtualization features supported.

Because of more advanced feature set and more implementation cases OVS would be a preferable choice for open source based SDN switching solution.

CONTROLLER IMPLEMENTATIONS

A summary of available open source controller options that could be used for development is described in this part. A more detailed overview of controller implementations was presented earlier in the document.

Controller	Description	License	Language
Beacon	A very efficient and fast multi-threaded OpenFlow controller with modular architecture	GPL	Java
Floodlight	Beacon-based OpenFlow controller with high performance processing capabilities. Is well supported by the community.	Apache	Java
Ryu	A network operating system that supports NETCONF and OF-Config in addition to OpenFlow and capable to integrate with OpenStack.	Apache	Python
OpenDaylight	A universal SDN controller that supports number of southbound interfaces with network abstraction capabilities.	EPL	Java
NOX	The first high-performance OpenFlow controller used in numerous implementations and researches.	GPL	C++
POX	A Python-based NOX controller that is mostly used for research.	GPL	Python

Every controller implementation has its own benefits different applications. There are numerous factors that can impact the controller choice. Some researches [45] in attempt to find a multi-purpose controller conclude that Ryu is the best choice. It is true that Ryu is sometimes referred to a network operating system, not just a controller because of the support of wide variety of protocols and feature and it would probably be the first choice for any enterprise implementation.

OpenDaylight is a good choice for as it originally intended to be a basis for SDN deployments in enterprise. Because OpenDaylight is based on Beacon controller code, it provide impressive performance characteristics. Network abstraction layer in OpenDaylight simplifies application development for OpenFlow and non-OpenFlow solutions.

Nevertheless, every network deployment is different and has individual requirements and deployment scenarios.

APPLICATIONS

Apart from custom SDN applications that tailor network behaviour to individual organization requirements there is a number of most important open source SDN applications. Every application has specific purpose and there is no way to list all categories. There are four main categories that find ones application in every enterprise: routing, security, network function virtualization and network management.

Application	Description
Quagga	IP routing protocol application (OSPF, IS-IS, BGP)
The BIRD	IP routing protocol application (OSPF, IS-IS, BGP)
Avior	FloodLight network management suite
FlowScale	Traffic load balancer utilizing OpenFlow protocol
Frenetic	An application that creates an abstraction layer in order to hide low-level network details of controller to forwarding device communication
FortNOX	A security platform that eliminates flow rules conflicts that can raise in multi-application SDN environment.
FRESCO	A security scripting application to easily develop security detection applications

The BIRD and Quagga are general routing application that provide traditional routing protocols support in SDN networks. Avior is Floodlight management application that runs independently from the controller and provides a graphical illustration of the network with capability to manually add, delete or modify flows. SDN has also means to solve network security issues in an efficient and manageable manner: FRESCO and FortNOX are two promising applications that enable security in OpenFlow networks. Network function virtualization (NFV) applications bring services virtualization feature to SDN framework. FlowScale is an NFV application provides load balancing capability to SDN solutions grounded on OpenFlow protocol.

FUTURE ACTIVITIES IN SDN COMMUNITY

Software becomes a major component of current network control and will keep to be determining element in future. Because of software development flexibility it will help to improve innovation and customization in today’s networks. Software defined networking is expected to change today’s static network topologies into intelligent programmable solutions that are extremely scalable with high degree of automation.

Universities all over the world have been conducting researches related to SDN. Stanford University and U.C. Berkley [41] are the most famous contributors to the SDN development. SDN ideas practically originated from these institutions and were extensively advanced by students and professors. The idea of control and data plane separation found it’s inception on Sanford campus, that basically led to creation of OpenFlow protocol. Number of researches were produced by joint efforts of Stanford and Berkley that were receiving funds from major vendors:

Cisco, VMware, and Google. ONRC research center [42] has a number of projects going at this moment that are mostly focused on increasingly popular wireless 4G and LTE networks. These type of networks are drawing attention of research teams because of exponential increase in mobile traffic nowadays and network scalability issues because of that. In addition to wireless technologies ONRC researchers are trying to improve SDN network management via SDN Version Control project [43], network virtualization by developing OpenVirteX platform that will be managing virtual Software Defined Networks (vSDNs) [44].

In summary, universities and research organizations play a very important role in SDN adoption and advancement cycle and we can expect a significant contributions to SDN community originated from these institutions.

Well-known enterprises and vendors usually organize research laboratories to conduct researches for their own purpose, but not for a specific product as final papers are usually published or presented at conferences. SIGCOMM is a professional forum that serves as a knowledge sharing community regarding communications and architectural questions of it. Some of the vendor research labs that took part in the conference presentations: Microsoft, NEC, IBM, HP and others. These organizations bring innovation to SDN technology by means of research of hardware network infrastructure, deployment best practises, protocol development and other SDN matters.

Network equipment vendors quickly joined SDN development initiative, even before official standards were released. Different network vendors chose different approaches to master SDN technologies. Some vendor achieved it through acquisitions of small start-ups or started funding SDN standardization bodies and alliances.

Practically every vendor has an SDN solution in its portfolio as market demand is very distinct nowadays. Some vendors have had a substantial influence on SDN development because of their market profile or commitment.

Cisco jumped into SDN technology quiet late comparing to others, but plays a major role in SDN community right now, in OpenDaylight project specifically. The other vector of Cisco efforts is proprietary APIs (onePK) in order to achieve SDN goals and extend functionality beyond OpenFlow standard. As a leading network equipment manufacturer, it is almost certain that Cisco activity on SDN market will have a considerable influence as Cisco APIs, that will force other vendors to develop same style APIs to compete with Cisco solutions on market. This makes Cisco a very similar to standardization body that everybody needs to follow if they want to stay competitive.

Brocade is one of the main developers and contributors of OpenDaylight project, as well a provider of OpenFlow devices for researchers. Brocade, is one of the vendors that offer network devices with both approaches, an SDN in its classical way (OpenFlow-based) along with API-based that helps to provide a less disruptive way to transition to full-blown SDN design.

Giants as IBM and NEC united at the inception of OpenFlow project and have been providing OpenFlow-supporting equipment to the community. As a contribution to OpenDaylight project vendors adapted virtualization applications form their own controller platforms to community OpenDaylight controller.

Hewlett-Packard one of the most enthusiastic contributors to SDN community since early years of OpenFlow standardization. HP is radically promotes and supports OpenFlow protocol producing a significant market value of OpenFlow switches and commercial SDN controllers. Because of large server's division, HP plays a significant role in OpenStack development and will probably introduce a virtualization-type solution based on that project.

Network vendor clients that run their business using equipment form Cisco, Juniper and others stimulate the development of SDN technologies as it is driven by their own interest to advance current networks and to get more optimized network gear in future. Giant companies as Google that provide cloud-based services all over the world have a particular interest in SDN adoption. Large-scale cloud environments require a vast virtualization capabilities for servers, storage and networking. Thus, there is a strong interest in optimization of networking performance and lowering overall cost. In Google scenario SDN-based approach have shown to be extremely effective, and this experience keeps moving community to new researches and development works.

Standardization is prerequisite for new technologies to be implemented in production networks. In case of SDN the standardization process is actively supported by industry alliances, opposed to IETF and IEEE. Alliances as ONF, OpenDaylight and OpenStack are major standardization bodies in SDN community.

ONF and OpenDaylight look at SDN from different angles. The ONFs board members are represented mostly by enterprise sector: Facebook, Google, Verizon etc. OpenDaylight alliance is solely driven by network equipment vendors as Cisco, Juniper, Citrix, Brocade and others. Vendors perceive SDN as opportunity to generate business cases and to develop new products. There is a great possibility that equipment vendors will be protecting their own business interests through innovation that is in favour to participating vendors only. On the other hand, because ONF is mostly controlled by enterprises, it may cause inconsistency between ONF standards and network device capabilities. As a result, ONF may define protocol versions that industry demands instead of what equipment vendors can actually implement. ONF has different workgroups that are active and work on protocols standardization. OpenFlow is one of them and is its standardization process is completely controlled by ONF.

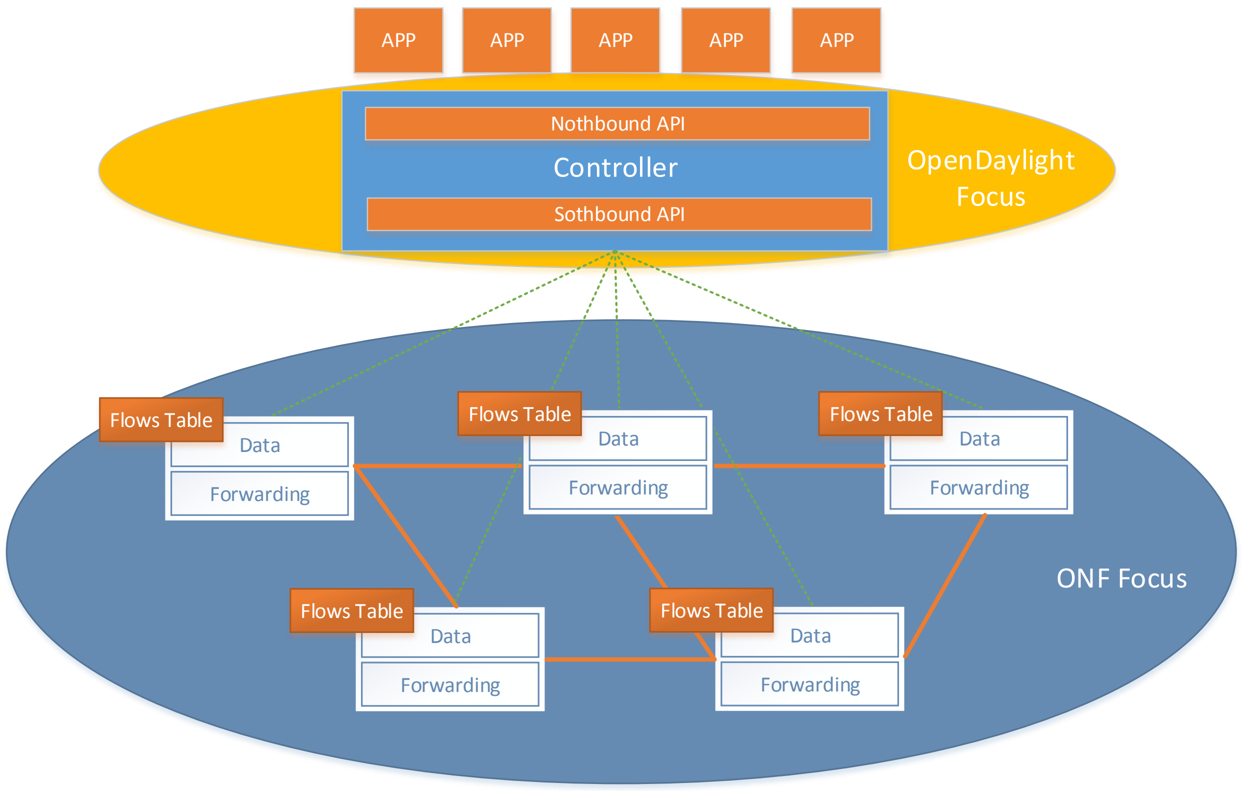


Figure 18. SDN development focuses

OpenDaylight, on the other hand is aimed to provide an open source network operating system that should serve as a base framework to develop and execute SDN applications (similar to Linux in computing). Open source approach provides flexibility to develop new applications and it just a matter of time to populate this niche with high-quality SDN applications. Certain concerns remain because of Cisco major position in OpenDaylight community, as it may untimely lead to Cisco-centric standards. And there is still some uncertainty regarding OpenDaylight real vision of SDN future either by providing truly open source products versatile as possible or to incline changes towards vendors needs while claiming it to be a part of SDN advancement.

The IETF is investigating models of SDN for practical technical realizations. However, the IETF standardization efforts are currently limited to SDN-related technologies as VXLAN, NVGRE, and STT as SDN overlay implementation. There are other areas of work for different related topics:

- The Interface to Routing System (I2RS) for intelligent programmability of routing devices;
- Stateful-PCE extension for MPLS that should make MPLS an SDN-driven;
- The Network Virtualization Overlays (NVO3) work group that is actively discussing data plane and control separation in data center environment.

CONCLUSION

It becomes clear from study of current network architectures that with appearance of virtualization, multitenant datacenters and cloud computing networking industry has faced new challenges. SDN holds a promise of changing network technologies towards more programmable, scalable and highly automated in order to overcome today's networks limitations.

SDN-based solutions are found to be exceptionally applicable in solving current network problems and network automation. Hybrid WAN idea is mainly a combination of various SDN technologies that will optimize traffic flows between the Internet and the MPLS, as well as control end-to-end path forwarding. Although hybrid WAN architecture is an inevitable step in network evolution and enterprises will keep experimenting with partial deployments to solve particular problems, complete SDN deployments will take time to be implemented in production networks.

REFERENCES

[1] D. Meyer, “Macro Trends, Complexity, and Software Defined Networking”, Advanced Technology Center, University of Oregon, Mar. 2013

[2] R. Jain and S. Paul, “Network virtualization and software defined networking for cloud computing: a survey,” 2013

[3] H. Kim and N. Feamster, “Improving network management with software defined networking,” 2013.

[4] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, “On scalability of software-defined networking,” 2013.

[5] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, “Intelligent design enables architectural evolution,” New York, NY, USA, 2011.

[6] R. Jain and S. Paul, “Network virtualization and software defined networking for cloud computing: a survey,” 2013.

[7] N. Feamster, J. Rexford, and E. Zegura, “The road to SDN,” Dec. 2013.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks,” SIGCOMM, Mar. 2008.

[9] A. Lara, A. Kolasani, and B. Ramamurthy, “Network innovation using OpenFlow: A survey,” 2014.

[10] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, “A survey of active network research,” 1997.

[11] ONF, “Open networking foundation,” 2014. https://www.opennetworking.org/

[12] M. Casado, N. Foster, and A. Guha, “Abstractions for software-defined networks,” Sep. 2014.

[13] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, “Software-defined internet architecture: Decoupling architecture from infrastructure,” 2012.

[14] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart,¨ and A. Vahdat, “B4: experience with a globally-deployed software defined wan,” SIGCOMM, 2013.

[15] ONF, “Software-defined networking: The new norm for networks,” Open Networking Foundation, Tech. Rep., April 2012.

https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[16] C. Rothernberg, C. N. A. Correa, R. Raszuk ”Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking”, SIGCOMM, 2012

[17] ONF, “Open networking foundation,” 2014. https://www.opennetworking.org/

[18] ONF, “Technical Specifications” https://www.opennetworking.org/sdn-resources/technical-library

[19] Tiantian Ren, Yanwei Xu, “Analysis of the New Features of OpenFlow 1.4,” 2nd International Conference on Information, Electronics and Computer (ICIEAC 2014)

[20] ONF, “SDN Northbound Interfaces Community,” http://www.onfsdninterfaces.org/

[21] “Open vSwitch,” 2013. http://vswitch.org/

[22] “Indigo Project,” June, 2013. http://www.bigswitch.com/topics/introduction-of-indigo-virtual-switch-and-switch-light-beta

[23] S. Schenker, “The Future of Networking, and the Past of Protocols,” October 2011. http://www.youtube.com/watch?v=YHeyuD89n1Y

[24] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, “Design and implementation of a routing control platform,” 2nd conference on Symposium on Networked Systems Design & Implementation. Berkeley, CA, USA, 2005.

[25] ONF, “OpenFlow management and configuration protocol (OF-CONFIG) v1.2,” 2014. https://www.opennetworking.org/images/stories/downloads/sdnresources/onf-specifications/openflow-config/of-config-1.2.pdf

[26] B. Pfaff and B. Davie, “The Open vSwitch Database Management Protocol,” RFC 7047, Internet Engineering Task Force, Dec. 2013. http://www.ietf.org/rfc/rfc7047.txt

[27] Big Switch Networks, “Project Floodlight,” 2013. http://www.projectfloodlight.org/

[28] L. Richardson and S. Ruby, RESTful web services. O’Reilly Media, Inc., 2008.

[29] OpenDaylight, “OpenDaylight: A Linux Foundation Collaborative Project,” 2013. http://www.opendaylight.org

[30] Cisco, “Cisco's One Platform Kit (onePK)”, http://www.cisco.com/c/en/us/products/ios-nx-os-software/onepk.html

[31] Arista Networks, “Extensible Operating System,” www.aristanetworks.com

[32] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “NOX: towards an operating system for networks,” 2008.

[33] D. Erickson, “The Beacon OpenFlow controller,” SIGCOMM, 2013.

[34] OSGI Alliance, “The OSGi Architecture,” http://www.osgi.org/Technology/WhatIsOSGi

[35] Quantum Communicty, “OpenStack Networking (”Quantum”),” 2012

[36] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, “VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks,” Internet Engineering Task Force, August 2014. Link: https://datatracker.ietf.org/doc/rfc7348/

[37] P. Garg Ed., Y. Wang Ed., “Network Virtualization using Generic Routing Encapsulation,” Internet Engineering Task Force, November 2014. Link: https://datatracker.ietf.org/doc/draft-sridharan-virtualization-nvgre/

[38] Eric Murray, “Challenges in Enterprise Networking and How SDN Can Help,” presentation, Open Network Summit, April 2012. Link: http://opennetsummit.org/archives/apr12/murray-wed-enterprise.pdf

[39] AlHarthy, K., Shawkat, W., “Implement network security control solutions in BYOD environment”, Control System, Computing and Engineering (ICCSCE), 2013 IEEE International Conference, November 2013

[40] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, “B4: experience with a globally-deployed software defined wan,” SIGCOMM 2013 conference, NY, USA, 2013.

[41] UC Berkley Research Faculty, Link: https://www.sdxcentral.com/listings/uc-berkeley-extension/

[42] Open Networking Research Center, Link: http://onrc.net/

[43] Stanford ONRC, “SDN Version Control”, Link: http://onrc.stanford.edu/sdn-version-control.html

[44] Stanford ONRC, “SDN Version Control”, Link: http://onrc.stanford.edu/network-virtualization.html

[45] Rahamatullah Khondoker, Adel Zaalouk, Ronald Marx, Kpatcha Bayarou Fraunhofer, “Feature-based Comparison and Selection of Software Defined Networking (SDN) Controllers” Institute for Secure Information Technology Rheinstr. 75, Darmstadt, Germany