Chapter 2

Related Work

In this section we will explain the SDN paradigm and how future networks may benefit from this novel architecture. We give an overview on load balancing algorithms, and conclude with a summary of the evaluation platforms that are used in our work.

2.1 Software-Defined Networking

Software-Defined Networking (SDN) [19, 27] is a new paradigm in networks. The main idea of SDN consists in the centralization of network control in a logically centralized program – the SDN controller – which controls and monitors the behavior of the network. The goal is to separate the control plane from the data plane. This separation is possible by means of an Application Programming Interface (API) between the switches and the controller, such as OpenFlow [30]. Networks thus become programmable, allowing the definition of the behavior of the entire network from the controller and the possibility to create advanced network policies such as load balancing, routing and security.

Figure 2.1 illustrates the various layers that constitute an SDN. The first layer, called application layer, consists of the applications that define the behavior of the network, commonly using the Representational State Transfer API [18]. This API uses the Hypertext Transfer Protocol (HTTP) to allow remote applications to send instructions to the controller or retrieve information from the controller. In the control layer we have a logically centralized software-based SDN controller, responsible for handling the control plane and maintain a global view of the network. This controller has the job of translating the applications instructions to the data layer by means of the OpenFlow API. It is also responsible to give applications an up-to-date view of the network state. The data layer is composed of the network devices responsible for packet forwarding, such as switches and routers. The communication between the data layer and the control layer is made by OpenFlow.

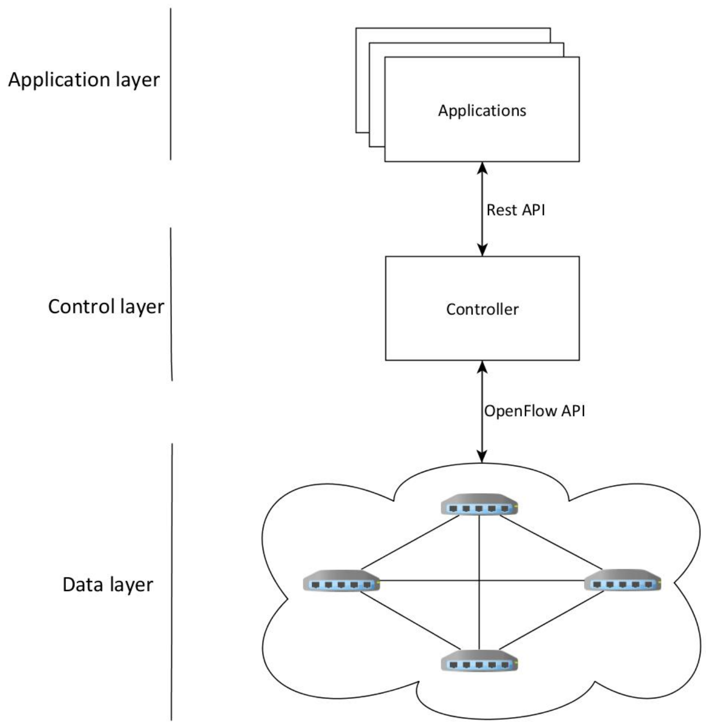

Figure 2.1: Network design using SDN

2.1.1 OpenFlow

OpenFlow [30] is a protocol that enables the communication between the switches and the SDN controller. OpenFlow started as a mechanism for researchers to realistically evaluate their experiments, as it enables the separation of experimental traffic from production traffic. This allows the use of a network switch for experiments without interfering with normal traffic. OpenFlow allows the modification of the flow tables of the switches, using a well-defined interface, by issuing commands from the controller. An OpenFlow-enabled switch (OF switch) can match packets against the different headers which enable more dynamic and flexible forwarding instructions than common network devices.

In Table 2.1 we illustrate a flow table that supports OpenFlow. The table shows the flow rules used to evaluate what action the switch should take when a packet for that particular flow arrives. The first 5 columns represent the packet headers that can be matched (this is what defines a flow). The column “Action” represents the action, defined by the controller, that the switch must perform when it matches on that row. Finally the last column represents the number of packets received by the switch that matched that flow. For example, we can see that all packages with Transmission Control Protocol (TCP) destination port 25 will be discarded and that the switch has already discarded 100 of those packets. The unknown packets (all the first 5 columns have only an *) are forwarded to the controller, which is the default behavior, the controller can then decide what action to perform to those packets.

MAC src	MAC dest	IP src	IP dest	TCP dport	Action	Count
*	10:20:*	*	*	*	port 11	235
*	*	*	123.8.2.1	*	port 2	300
*	*	*	*	25	drop	100
*	*	*	*	*	Controller	455

Table 2.1: Example of an OF-enabled switch flow table

2.2 SDN controllers

Controllers are the core component of an SDN. They oversee the behavior of the entire network and implement the decisions to achieve the desired state for the network. They are a logically centralized program that offers services and applications for controlling the network. It is important to emphasize that a logically centralized program does not mean that we have a centralized system. Actually, the controller can be distributed and replicated for fault tolerance and/or better performance [26]. In any case, applications are written as if the network view was stored on a single machine [21].

2.2.1 Nox

NOX [21] was the first SDN controller and was written in C++ and Python. As shown in Figure 2.2, a NOX-based network consists of a set of switches and one server, running the NOX controller software and the management applications over it. The NOX programing model is event-driven, meaning that, the data plane triggers events, like a *Packet In* event, and applications are notified of the event. NOX has core applications to discover and observe the network components. These applications are responsible for creating and updating a single database containing all network observations and data (network view), providing observation granularity at the switch-level topology, showing the locations of users, hosts, middleboxes, and other network elements [21]. Like any other centralized controller, Nox has to handle all flows in the network making it a possible bottleneck. Anyway this controller is able to handle around 100000 flows per second [21], which is considered enough for a good range of networks [15].

2.2.2 Onix

Onix [26] is a distributed SDN control platform that runs on a cluster of one or more physical servers, each of which may run multiple Onix instances. Onix uses a database

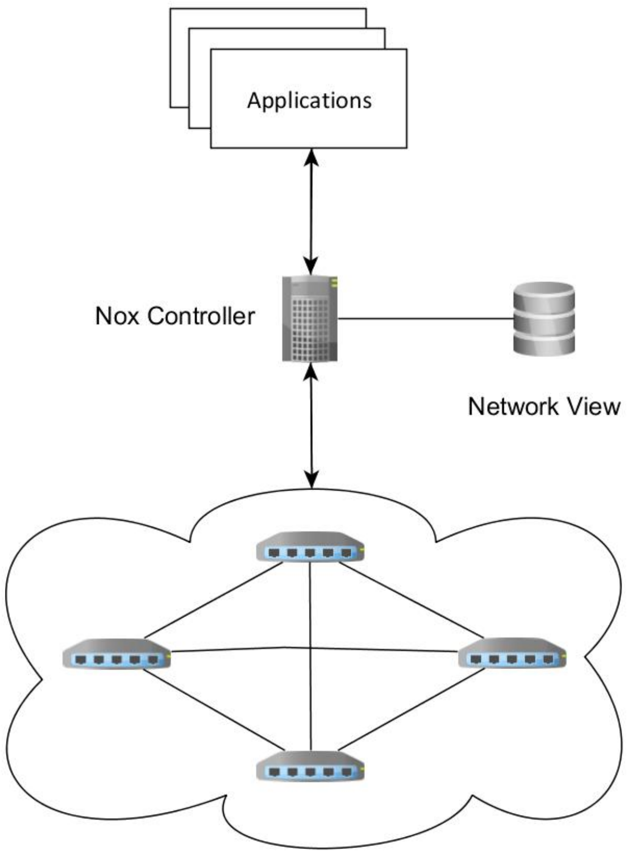

Figure 2.2: NOX-based network

called Network Information Base (NIB) that stores the current state of the network. The state in the NIB is distributed and replicated among all Onix instances using basic state distribution primitives. Onix also provides a general API which allows, depending on the desired implementation, to make trade-offs among consistency, durability, and scalability.

Contrary to other controller designs, the NIB sits between the management plane and the control plane, and it is through this database that the applications interact indirectly with the data plane. The management plane modifies the NIB and the controller reads those modifications and translates them in commands to the data plane. In the other way around, the controller updates the NIB according to the events triggered by the data plane, and notifies the applications about the updates made in the NIB. Every time a NIB is modified, the NIBs of the other Onix instances must be updated, for the sake of consistency. Onix provides the possibility of choosing between strong or eventual consistency for this purpose. For strong consistency it offers a transactional persistent database, and for eventual consistency it has a memory based Distributed Hash Table (DHT) available.

2.2.3 Floodlight

Floodlight [8] is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. This is the one we have chosen to use in our project because it is designed to offer high-performance and scales well with the number of network components [8]. The fact that it is implemented in Java also contributed to this decision.

The Floodlight controller is based on another controller called Beacon [17]. Java was the chosen programming language because it offers the best balance between performance and user friendliness. It is also portable, which means it can run on a variety of operative systems. In addition, Beacon (and Floodlight) has a good and simple API and comes with useful applications:

- Device manager: tracks devices seen in the network including information on their addresses, last seen date, and the switch and port last seen on;
- Topology: discovers links between connected OpenFlow switches;
- Routing: provides shortest path layer-2 routing between devices in the network;
- Web: provides a Web user interface.

One advantage of Beacon and Floodlight is the runtime modularity, the capability of not only starting and stoping applications while it is running, but to also add and remove them, without shutting down the controller process. Applications are fully multithreaded having blocking (Shared Queue) and non-blocking (Run-to-completion) algorithms for reading OpenFlow Messages. The evaluation presented in [17] concluded that Beacon was the controller with best performance when compared to NOX [21], Pox [5] and Maestro [32].

2.3 Load Balancing

Web applications scale by running on multiple servers to be able to service an increasing number of users that demand Web content. To achieve the desired performance, load balancers are used to distribute the request by the replicas. This results in important benefits such as scalability, availability, manageability, and security of Web sites [20]. The Load balancer job is to choose which server should handle the next request, using algorithms such as Round-Robin. After receiving a request from the client, it applies the load balancing algorithm and forwards the request to the chosen server.

Load Balancers today consist of expensive specialized hardware, the dispatcher, located at the entrance of the network [37]. This dispatcher is a special component used only for load balancing so it can handle many requests with good performance. The dispatcher may become a bottleneck and it is therefore necessary to replicate the load