Formal models of software-defined networks

Feng, H.

Chapter 2

Software-defined Networks

In this chapter, we briefly introduce software-defined networks (SDNs) starting from the idea behind SDNs and their benefits over traditional computer networks. We also discuss a few protocols and concentrate on OpenFlow, a protocol that enables the centralized control of network switches and allows for programmable networking capabilities.

2.1 Traditional networking vs. SDN

Traditional computer network architectures are based on the seven layers of the Open Systems Interconnection (OSI) model: the physical layer, the data link layer, the network layer, the transport layer, the session layer, the presentation layer, and the application layer. More specifically these layers describe how data is transferred and received over a network. Each layer is accountable for carrying out particular tasks related to sending and receiving data, whereas, for a message to be delivered and received, all of the layers must be utilized.

The seven layers above provide a conceptual framework for understanding network communication[20]. When considering the functional components and responsibilities we then classify network components depending on the control logic, the forwarding functionality, and where applications are executed. In general, the primary function of a network is to ensure the transport of packets via routing decisions and packet forwarding. The actual forwarding part of the network is called the data plane, whereas the part controlling how this has to be executed is called the control plane. In a traditional network, the control and

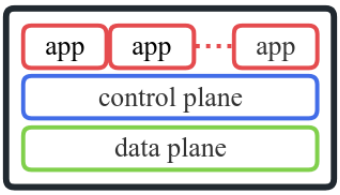

Figure 2.1: Structure of the switch in a traditional network

data planes are closely tied together with the application plane, see Figure 2.1, as they are in charge of packet forwarding, building, and maintaining the routing tables. In fact, within the OSI model[102], switches are typically in the data link layer, and are responsible for forwarding packets by using physical addresses that identify devices on a network, the so-called MAC addresses. But switches can also have routing functionality and are instead positioned in the network layer. As such, if the behavior of this network needs to be changed the switches have to update their local control planes accordingly. This is even more difficult when more and more network devices are employed since the control plane is highly distributed without a global view of the network. As a result, it is very difficult to program network-wide decisions and even more difficult to verify their correspondence against global requirements.

The solution of an SDN architecture is to separate the data plane from the control plane and the application plane[62], moving the latter two from the switches to the network as shown in Figure 2.2. The advantage is a much simpler way to regulate the network behaviour allowing for more flexible and dynamic network management, as well as improved scalability and security. Since the data plane is only responsible for packet forwarding, modifying the networks can now be done at the application level, using software, thus explaining the name "software-defined" networking.

In an SDN, the control plane consists of a few controllers that are responsible for managing a large part of a network. Each controller can be configured to decide how to transfer packets over the network based on a variety of factors, such as traffic patterns, security policies, and service-level agreements.

One of the key benefits of an SDN is that it allows for more efficient use of network resources because each controller can make routing decisions in real-time, without the need to stop the forwarding process of the switches, for example by reconfiguring the network logical structure in response to changing conditions,

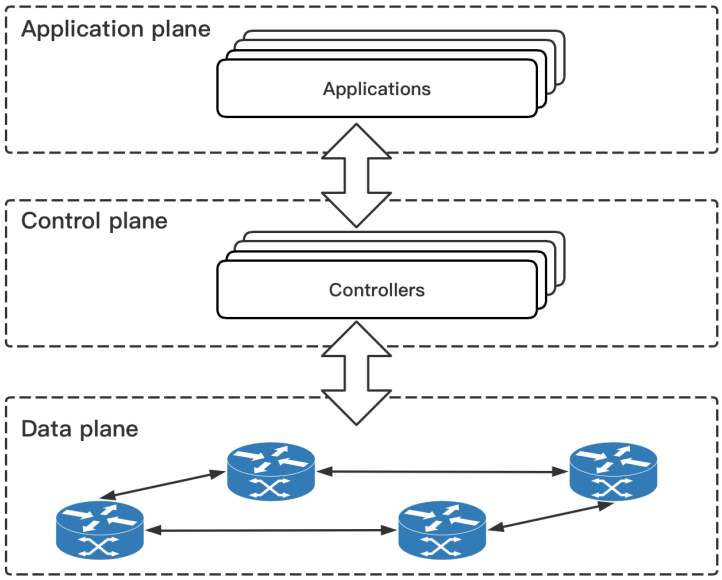

Figure 2.2: SDN architecture

such as nodes or hardware link failures.

2.2 Protocols for SDNs

The architecture offered by SDNs simplifies the design and deployment of network management tasks: the control plane is a logically centralized controller that gathers information from the data plane and provides a global view to applications running on top of the controller. These applications make packet routing decisions based on the global view and distribute these decisions to the data plane via the control plane. There is a northbound interface (NBI) between the application plane and the controller for permitting a specific entity of a network to communicate with a higher-level entity, the interface is on the upper side of the controller called the "northbound" interface. Different from the southbound interfaces (SBIs) between controllers and the switches, NBIs are offered by the controllers themselves and are vendor-dependent as there is no open standard NBI.

On the contrary, the SBI supports communication between the control planes and data plane and is typically subject to OpenFlow [82], a specific communications protocol developed by the Open Networking Foundation (ONF). While

OpenFlow is the most famous communications protocol for SDNs, it is not the only one. For instance, other popular protocols for the SBI of an SDN include:

- the Simple Network Management Protocol (SNMP), an Internet standard protocol used for managing and monitoring network devices, allowing for the collection of information and the control of network elements in a managed network [115] [79];
- the Forwarding and Control Element Separation (ForCES) protocol that separates the control plane from the data plane in network devices beyond switches using a master-slave mechanism, and allowing for centralized control and management of forwarding elements [45];
- the Path Computation Element (PCE) protocol, a communication protocol used to enable centralized path computation and optimization in multi-domain or multi-layer networks, moving this functionality from the routes operating systems to the control plane [105];
- the Border Gateway Protocol (BGP), used to exchange routing and reachability information between autonomous systems (AS) on the Internet, enabling the interconnection and dynamic routing decisions among different networks [91, 73].
- the NETwork CONFiguration Protocol (NETCONF), based on the Yet Another Next Generation (YANG), used for remote network device management and configuration, providing a programmable interface for network automation and orchestration [31].

Each of these protocols has its own set of features, capabilities, and use cases and all can be used in an SDN architecture without competing with each other. In this thesis, we only concentrate on the OpenFlow protocol.

2.3 The OpenFlow protocol

Next, we present an informal introduction to the basics of the OpenFlow protocol, following the specifications released in [39]. We first recall the definition of an SDN packet. Then we describe the three types of OpenFlow messages, from where they flow, and how they are used to change the network behaviors.

OpenFlow is a communications protocol that is used to control the flow of data in SDN. It provides a standard way for SDN controllers to communicate with switches and viceversa [97, 82, 52, 64, 17, 51].

2.3.1 SDN packets

A packet is a unit of data that carries information across a network, encapsulating the control information necessary for its transmission. Besides data, it consists of a header containing fields storing, for example, source and destination addresses, and protocol information. The example below shows two packets, both with a header containing information about the `tcp` and `ethernet` destination address of the packet:

`tcp_dst`:22, `eth_dst`:11	`data`: d1

`tcp_dst`:23, `eth_dst`:11	`data`: d2

2.3.2 OpenFlow flow tables and flow entries

In an SDN architecture, each switch in the data plane has a number of ports that are used to receive and forward packets. Switches are connected to at least one controller, from which they may receive or which they may send OpenFlow messages to. The main task of a switch is to forward packets to other switches.

When a packet arrives at a port of a switch, the SDN controller, which has a global view of the network, determines how the packet should be handled based on its header information. The process of forwarding a packet involves forwarding rules stored in a flow table. The table consists of an ordered set of pairs (b, a), where b is a Boolean condition on the values in the header fields of a packet (the so-called matching criteria) and a is the corresponding action to be executed on the matching packet. The order of the matching-action pairs gives priority to the application of the matching condition. There are basically three types of actions: *forwarding* a packet to one or more ports of the switch, *dropping* a packet, and *updating* a field of a packet with some value.

For example, the leftmost packet above matches the first rule of the table below, and it is forwarded to the output ports 3 and 4. The rightmost packet however matches only the last rule and it is forwarded to port 1 after its field `tcp_dst` is updated to 22.

Matching Condition	Action
`tcp_dst`:22	`Forward`[3, 4]
`tcp_dst`:23, `eth_dst`:12	`drop`
`true`	`tcp_dst` := 22; `Forward`[1]

2.3.3 OpenFlow messages

Controllers and switches communicate through three types of messages. A `PktIn` message is from a switch to a controller. It includes a packet with all its header fields and data and some additional information such as the port from where the packet entered into the switch. Typically a `PktIn` message would be processed by the controller to trigger an update of the flow tables.

A `PktOut` message is from a controller to a switch. It consists of a packet together with a flow table action to be executed by the switch. This way a packet need not pass through the flow table but is, for example, immediately forwarded to other switches. The `PktOut` message is used to implement a one-time redirection of a packet to a different path after a decision of the controller.

The flow table of a switch is updated by a `FlowMod` message, and it is sent by a controller to the switch. Each `FlowMod` message consists of a `ModType` t (`Add`, `Remove`, `Modify`), a matching condition b and an action a. If $t =$ `Add` then the pair (b, a) is added on top of the table (having thus highest priority), while if $t =$ `Modify` then the first pair in the flow table (b', a') with b implying b' is substituted with the new pair (b, a). In the remaining case when $t =$ `Remove`, the first pair in the flow table (b', a') with b implying b' is removed from the table. In this case, the action a does not play any role and therefore can be considered empty. The `FlowMod` message allows a controller to add or remove rules to the flow table, enabling the programmatic control of the behavior of the network. Note that the action of an entry in the flow table is executed only when a package arrives matching the respective conditions.

Those three types of messages plus dedicated packets to communicate data allow controllers to gather information about the network and manage it.

2.3.4 Evaluating the OpenFlow protocol

OpenFlow enables centralized control over network devices, allowing for a global view of the network and centralized control decisions. This facilitates dynamic and flexible network management and it enables programmatic control

by SDN controllers [38, 93]. The controller can be programmed to reconfigure the network automatically in response to changing the network status. This can help to improve the network's overall performance, reliability, and scalability.

OpenFlow primarily focuses on the communication between the control plane and data plane of network switches. The centralized control and programmability introduced by OpenFlow can also pose vulnerability and security challenges [107] [10]. Also, the centralized control model of OpenFlow can become a scalability bottleneck, and distributing the management of switches among several controllers may introduce race conditions. Finally, the matching criteria of a flow table are limited by the fields of a packet header. For example, predicates on the action result are not allowed, limiting some traffic patterns or specialized requirements.

2.4 Two examples

Consider the scenario presented in Figure 2.3 in which Switch 1 forwards all packets received from port A to port C, and sends packets received from port B to port Q as a `PktIn` messages for the controller. The controller in this example has a local view of the topology of the network. In particular, it is aware that ports A and B are connected to the host devices with IP addresses 192.168.0.1 and 192.168.0.2, respectively, while port C is connected to the host device with IP address 192.168.1.1. Further, the switch uses port P for incoming `PktOut` and `FlowMod` messages and port Q for outgoing `PktIn` messages.

The first time a packet is received by Switch 1, by using the OpenFlow protocol the controller will receive a `PktIn` message containing the packet and may, for example, decide to update the flow table of the Switch by sending a `FlowMod` message so that the action of the second line of the flow table is updated to forwarding a packet to port C. By sending also `PktOut` message with the original packet and the new action, the controller guarantees that no placket gets lost.

A second example is given by a controller connected to four switches as shown in Figure 2.4. Assume all switches have initially an empty flow table. When switch 1 receives a packet at port A, since the packet does not match any entries of the flow table, the switch sends it as a `PktIn` message to the controller. The controller calculates a path to the destination device (say a device with IP number 192.168.1.1) and it returns a `FlowMod` message to all switches involved in the path so that the packet is forwarded along ports B, C, D, E, F, G and H in the path

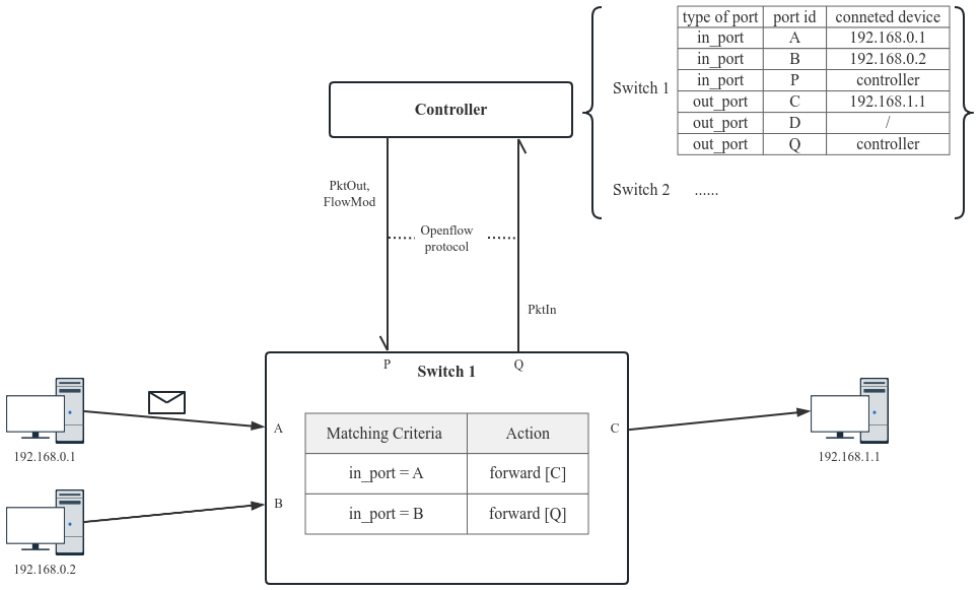

Figure 2.3: A single switch SDN network

(see Figure 2.4). Subsequently, the controller sends a `PktOut` message to switch 1 only, containing the original packet and the action for forwarding it to port B. As the packet moves along the path to its destination, the flow table of the other switches may or may not have received the `FlowMod` message for updating the table. If yes the packet is forwarded from one switch to another until it arrives at its destination.

However, because of an eventual race condition, if one of the switches along the path has not yet received the `FlowMod` message for updating the table, when the packet arrives it will be forwarded again to the controller inside a `PktIn` message, and the controller will have to send it back via a `PktOut` message with the corresponding forwarding action. However, once the first packet arrives at its destination, it can be guaranteed that the next packet that arrives at port A of Switch 1 will go directly to its destination without the need for a controller to intervene.

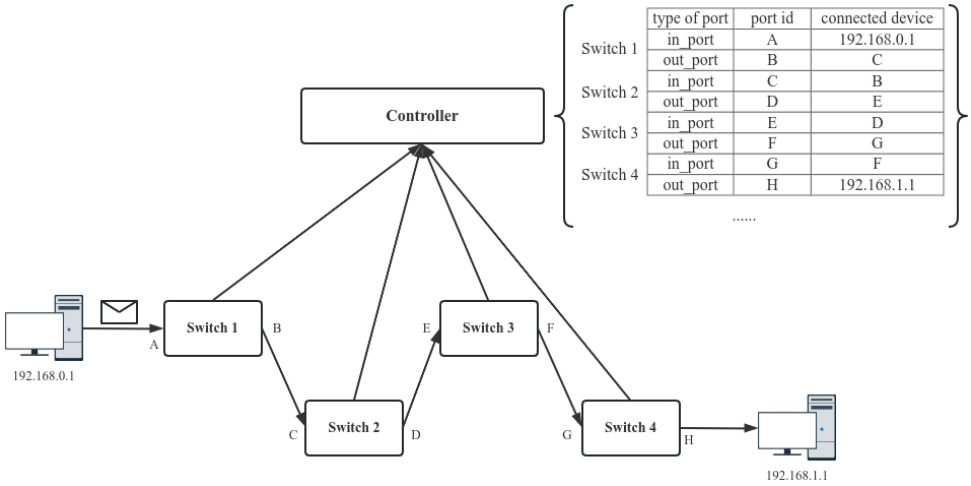

Figure 2.4: A multi-switch SDN network