4 Software Defined Networks

This chapter introduces Software-defined Networks (SDN) as a new networking approach in contrast to traditional networks. In order to understand the principles of SDN, the layer views of network functionality in traditional networks and in SDN are juxtaposed, the SDN architecture is then presented. The concepts of flow commonly used in the SDN literature is analysed in this text. A deployment example with P4-based SDN is followed. The exercise illustrates how to employ SDN to “program” a network, revealing a new way of performing network control and management.
本章介绍了软件定义网络（SDN）作为一种与传统网络形成对比的新型网络方法。
为了理解SDN的原理，本文将传统网络和SDN的网络功能分层视图进行对比，并随后介绍了SDN的架构。
本文还分析了SDN文献中常用的流的概念，并提供了一个基于P4的SDN部署示例。
该练习展示了如何利用SDN来“编程”网络，揭示了一种新的网络控制和管理方式。

4.1 Introduction

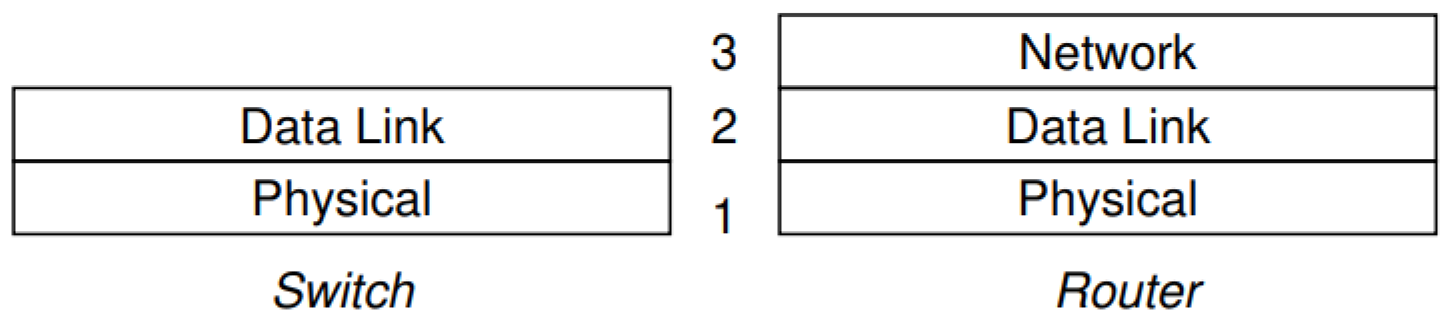

Figure 4.1: Switch and Router in OSI model [Danc 15]

There are different ways to layer networked systems. Two well-known ones are the ISO OSI (Open Systems Interconnection) and the TCP/IP reference models. Figure 4.1 shows an example of a switch and a router through the OSI view. The switch spans two OSI layers, namely Physical and Data Link whereas the router implements an additional Network layer. A network device is classified into its highest layer, which means that the switch belongs to the Data Link layer and the router to the Network layer. The OSI layering mechanism provides a common basis for the coordination of networking standards’ development as well as facilitates the understanding of these standards and the interaction of networked systems.
网络系统的分层方式有多种。两个众所周知的模型是ISO OSI（开放系统互连）模型和TCP/IP参考模型。图4.1展示了通过OSI视图观察交换机和路由器的示例。交换机跨越了OSI的两层，即物理层和数据链路层，而路由器实现了额外的网络层。网络设备按照

其最高层进行分类，这意味着交换机属于数据链路层，而路由器属于网络层。OSI分层机制为协调网络标准的开发提供了一个共同的基础，同时也促进了对这些标准的理解以及网络系统的交互。

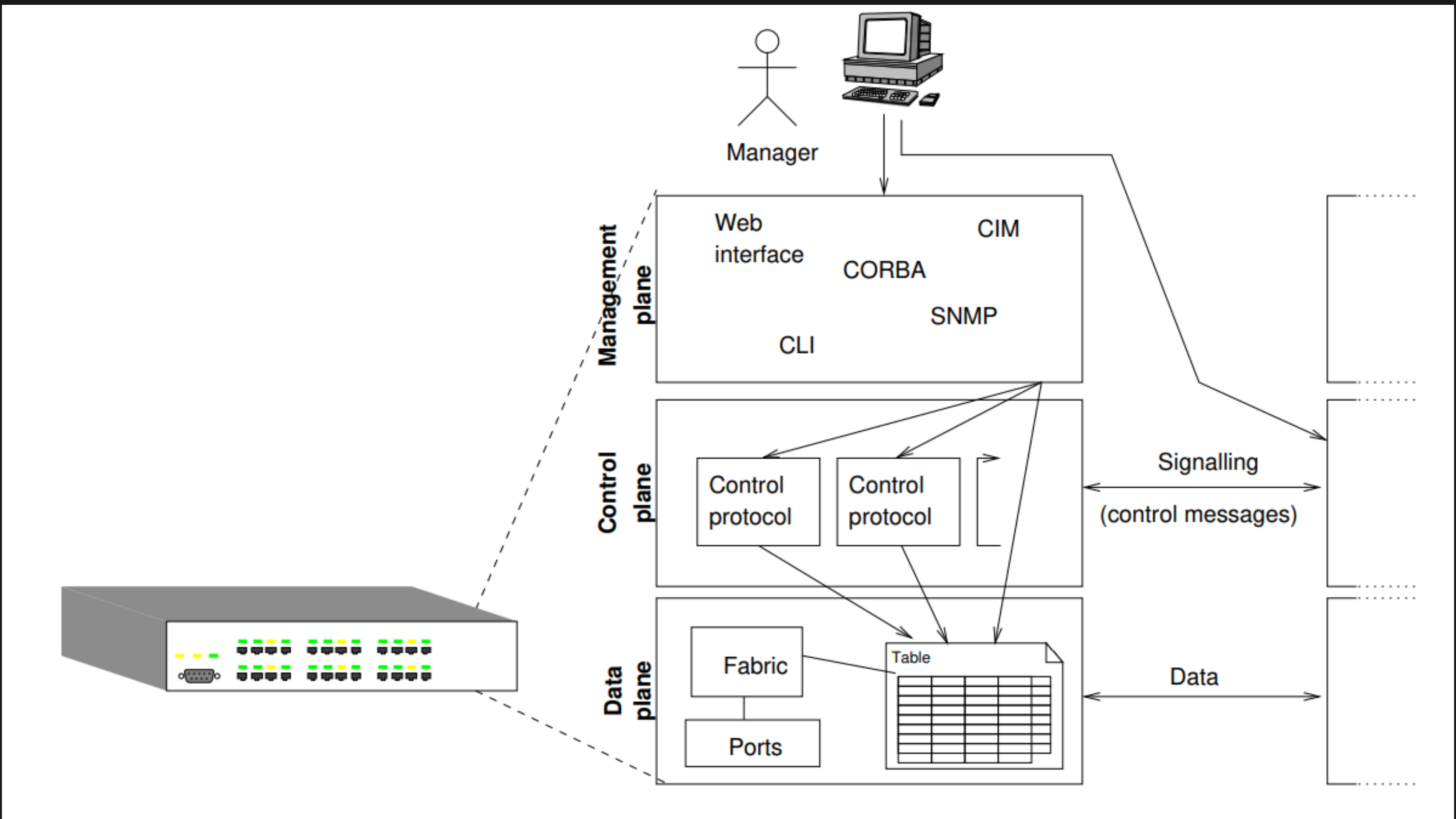

Figure 4.2: Layer view of networking device's functionality [Danc 15]

In another view, a traditional network device can be seen as being composed of the three planes (or layers) depicted in Figure 4.2: management plane, control plane, and data plane.
从另一种视角来看，传统网络设备可以看作由图4.2所示的三个平面（或层）组成：管理平面、控制平面和数据平面。

- The data plane consists of various ports for receiving and transmitting packets based on its forwarding table, and switching fabrics for transferring packets from an input buffer to an output buffer.
  数据平面包括用于接收和发送数据包的各种端口（基于其转发表）以及用于将数据包从输入缓冲区传输到输出缓冲区的交换结构。
- The control plane represents protocols used for populating forwarding tables in the data plane, e.g., the routing protocols like RIP, OSPF, BGP in a router.
  控制平面表示用于填充数据平面中转发表的协议，例如路由协议，如RIP、OSPF、BGP。
- The management plane includes software services used by a network administrator (manager) to monitor and configure the control functionalities.
  管理平面包括网络管理员（管理者）用于监控和配置控制功能的软件服务。

Figure 4.3 describes the roles of these planes and their interactions.
图4.3描述了这些平面的角色及其交互关系。

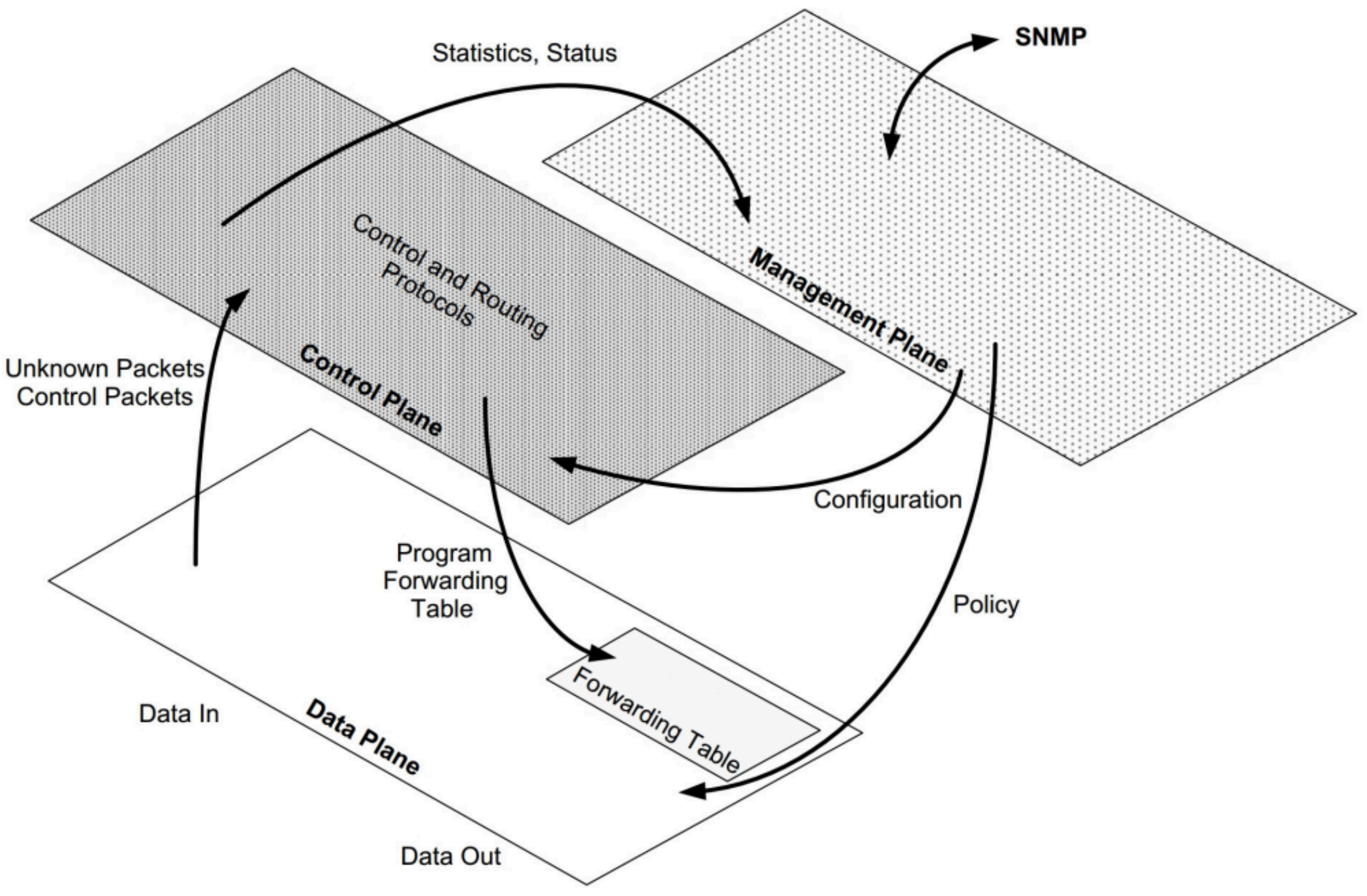

Figure 4.3: Roles of the management, control and data planes [GoBl 14]

Although traditional IP networks have widespread adoption, they are complex and hard to manage. To make any change to a network, operators need to configure each individual network device separately using low-level and vendor-specific commands. The vertical integration of the control and data plane in each network device reduces the flexibility and hinders the innovation and evolution of networking infrastructure. These limitations pose a question of new networking paradigms, leading to the introduction of Software-Defined Networking (SDN).

尽管传统的IP网络被广泛采用，但它们复杂且难以管理。要对网络进行任何更改，操作员需要使用底层且特定于供应商的命令单独配置每个网络设备。控制平面和数据平面在每个网络设备中的垂直集成降低了灵活性，并阻碍了网络基础设施的创新和发展。这些限制提出了关于新网络范式的问题，从而引入了软件定义网络（SDN）。

Some key ideas of SDN are the introduction of dynamic programmability in forwarding devices, the decoupling of the control and data planes, and the global view of the network by logical centralization of the “network brain” [KRV+ 15] in a single place, namely the controller.

SDN的一些关键理念包括在转发设备中引入动态可编程性、解耦控制平面和数据平面，以及通过逻辑集中化将“网络大脑”集中在单一位置（即控制器）以实现网络的全局视图 [KRV+ 15]。

4.2 Architecture

Having explained the three-plane view of traditional networks, we will now go deeper into the SDN architecture.

在解释了传统网络的三平面视图之后，我们现在将深入探讨SDN架构。

SDN is a network architecture where network control is decoupled from forwarding and is directly programmable [Foun 12].

SDN是一种网络架构，其网络控制与转发解耦并直接可编程 [Foun 12]。

A comparison of a traditional network and an SDN in the three-plane view is illustrated in Figure 4.4. In the traditional network, the routing table at the data plane of a router is populated by its control plane residing in the same device, such as the OSPF routing protocol running on the router's operating system in this example. In SDN, the controller acts as a "network operating system," and the OSPF routing protocol runs atop the controller, which instructs the controller to populate flow tables in the SDN devices via the southbound API. The mapping of the management plane's services in these two networks is not relevant in terms of network control and is not portrayed in this example to avoid unnecessary confusion.

图4.4展示了传统网络和SDN在三平面视图中的对比。在传统网络中，路由器的数据平面的路由表由其控制平面填充，控制平面位于同一设备中，例如运行在路由器操作系统上的OSPF路由协议。而在SDN中，控制器扮演"网络操作系统"的角色，OSPF路由协议运行在控制器之上，并通过南向API指示控制器填充SDN设备中的流表。在这一示例中，管理平面服务的映射与网络控制无关，因此未呈现，以避免不必要的混淆。

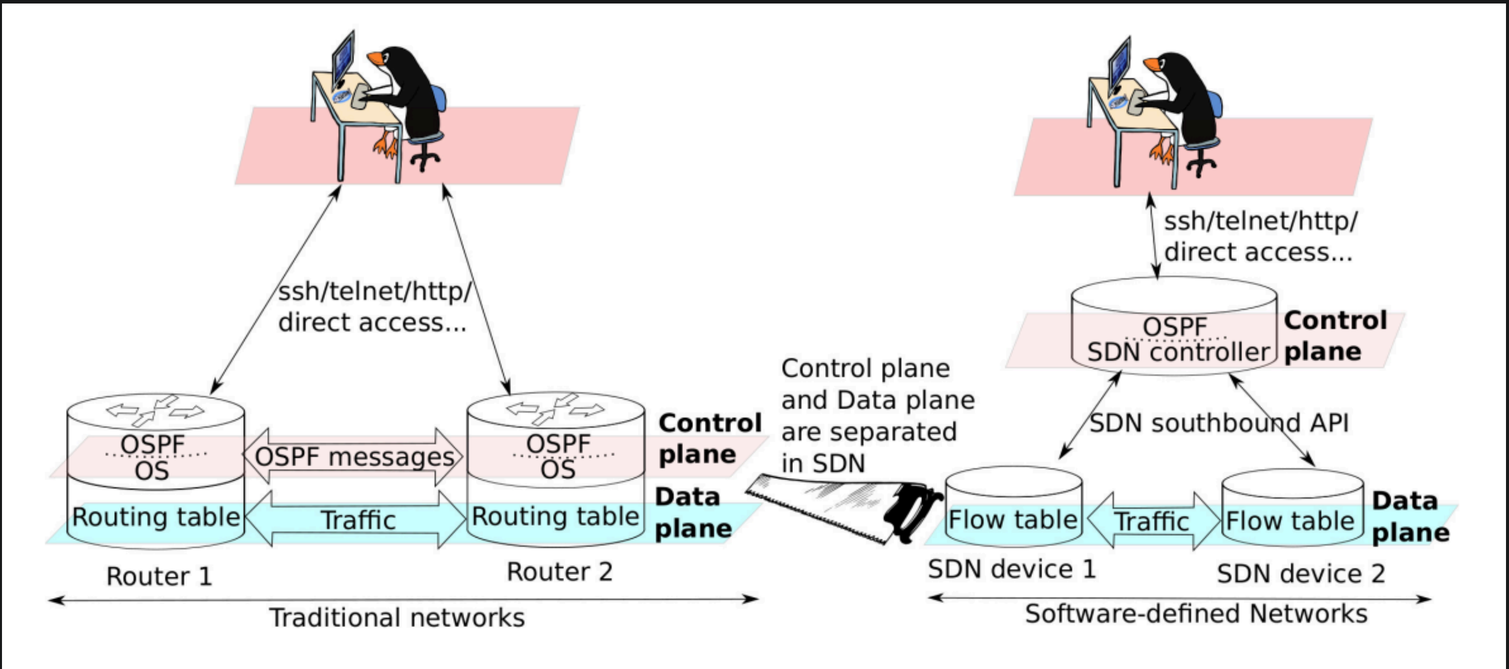

Figure 4.4: An example of functionality plane mapping in traditional networks and in SDN

The general SDN architecture is illustrated in Figure 4.5.

图4.5展示了通用的SDN架构。

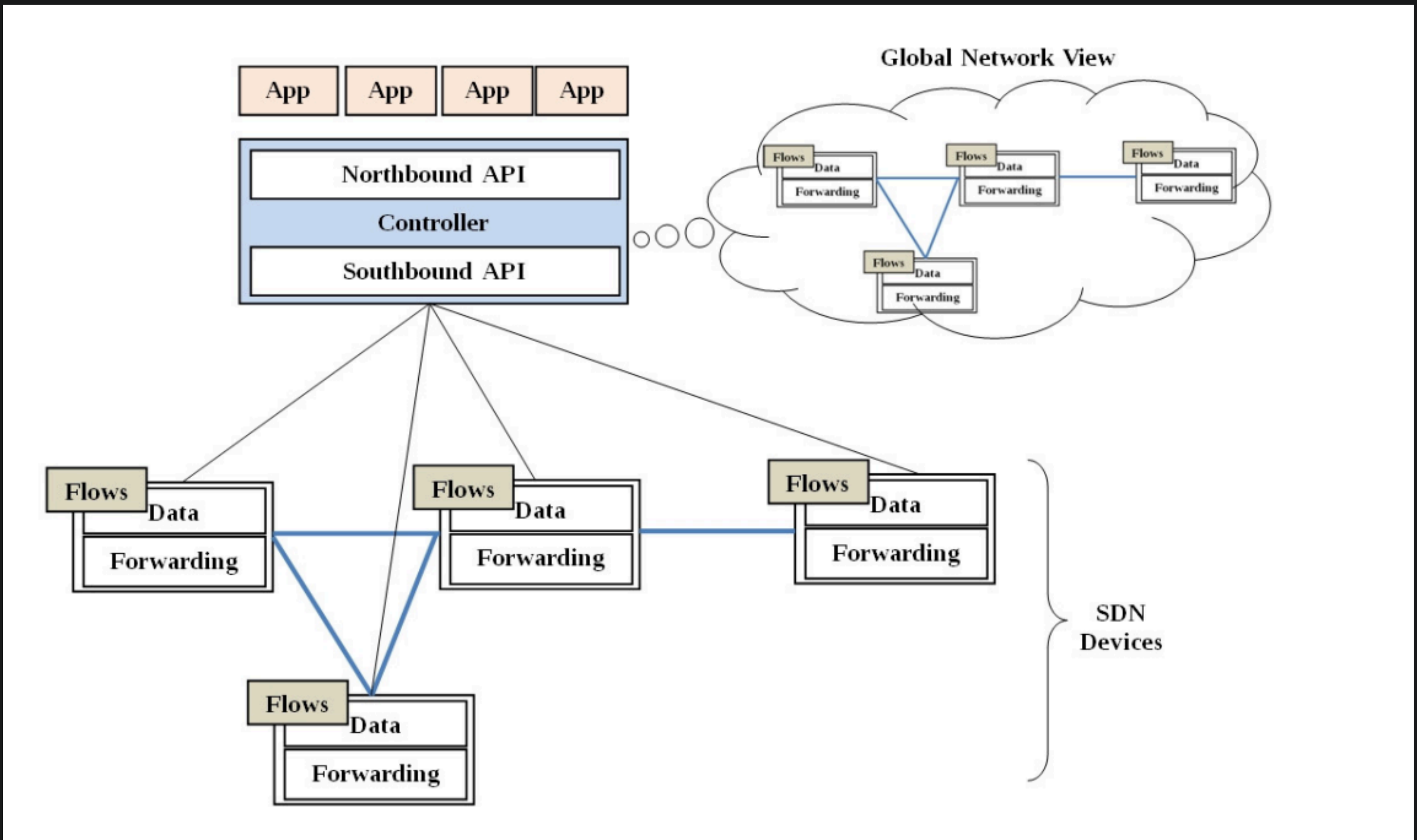

Figure 4.5: General architecture of SDN [GoBl 14]

Kreutz et al. [KRV+ 15] define SDN as a network architecture with four pillars:Kreutz等人 [KRV+ 15] 将SDN定义为一种具有以下四个支柱的网络架构：

1. The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.
   控制平面和数据平面解耦。控制功能从网络设备中移除，使这些设备变成简单的（数据包）转发元素。
2. Forwarding decisions are flow-based, instead of destination-based (refer to Section 4.3 for the explanation of the flow concept).
   转发决策基于流而不是基于目的地（详见第4.3节对流的概念的解释）。
3. Control logic is moved to an external entity, the so-called SDN controller.
   控制逻辑迁移到一个外部实体，即所谓的SDN控制器。
4. The network is programmable through software applications running on top of the controller that interacts with underlying data plane devices.
   通过运行在控制器之上的软件应用程序对网络进行编程，这些应用程序与底层数据平面设备交互。

Another important characteristic of SDN mentioned in [RFC 7426] is the standardization of the interfaces between the control and data planes.
[RFC 7426] 中提到的SDN的另一个重要特性是控制平面和数据平面之间接口的标准化。

4.2.1 Components

The main components of a SDN include devices in the data plane and one or more controllers.
SDN的主要组件包括数据平面的设备和一个或多个控制器。

SDN-devices

SDN devices, also commonly referred to as SDN switches, are simple forwarding elements without embedded control or software to take autonomous decisions like routing protocols or default/fixed forwarding behavior. The network intelligence is removed from them to a logically centralized controller. Packets are handled in these switches based on their flow tables, whose entries contain control information of different layers (e.g., layer 2 - 4). Standardized interfaces are introduced between the controller and the switches, which provide a means for the controller to install flow tables in these switches.

SDN设备，也常被称为SDN交换机，是简单的转发元素，没有嵌入的控制功能或用于做出自主决策的软件，如路由协议或默认/固定的转发行为。网络智能从这些设备中被移到逻辑集中化的控制器中。这些交换机根据其流表处理数据包，流表中的条目包含不同层（如第2层至第4层）的控制信息。在控制器和交换机之间引入了标准化接口，控制器可以通过这些接口向交换机安装流表。

Controller

The controller is a software stack that controls SDN devices. One important function of the controller is the provision of topology service, which maintains a consistent overview of the network. Any change in the network, such as new devices being added or some devices being removed, or some links being down, should be reflected immediately in the network overview at the controller. The controller changes the configuration of network devices based on applications' requests.

控制器是一个控制SDN设备的软件栈。控制器的一个重要功能是提供拓扑服务，它维护网络的一致概览。网络中的任何变化，例如新设备的添加、某些设备的移除或某些链路的中断，都应立即反映在控制器的网络概览中。控制器根据应用程序的请求更改网络设备的配置。

According to Goransson et al. [GoBl 14], four fundamental functions of an SDN controller are:
根据Goransson等人 [GoBl 14] 的说法，SDN控制器的四个基本功能是：

- **End-user device discovery**,
  终端用户设备发现，

- **Network device discovery**,
  网络设备发现，
- **Network device topology management**, which maintains information about the interconnection details of the network devices to each other and to the end-user devices to which they are directly attached,
  网络设备拓扑管理，维护网络设备彼此之间以及与直接连接的终端用户设备之间的互联细节信息，
- **Flow management**, which maintains a database of the flows being managed by the controller and performs all necessary coordination with the devices to ensure synchronization of the device flow entries with that database.
  流管理，维护由控制器管理的流的数据库，并与设备进行所有必要的协调，以确保设备的流表条目与该数据库同步.

Further functions of the network are realized by SDN applications (see Section 4.2.3).
网络的进一步功能由SDN应用程序实现（见第4.2.3节）。

4.2.2 Interfaces

The SDN architecture introduces two interfaces: the northbound APIs lying between the applications and the controller, and the southbound APIs between the controller and the SDN devices. While OpenFlow [MAB+ 08] and P4Runtime [P4RTSpec] appear to be the most dominant southbound APIs, there are no such counterparts for northbound APIs so far. Each controller has its own northbound APIs.

SDN架构引入了两种接口：位于应用程序与控制器之间的北向API，以及控制器与SDN设备之间的南向API。尽管OpenFlow [MAB+ 08] 和P4Runtime [P4RTSpec] 是最为主流的南向API，但目前北向API尚无类似的标准化对等接口。每个控制器都有其独特的北向API。

4.2.3 SDN Applications

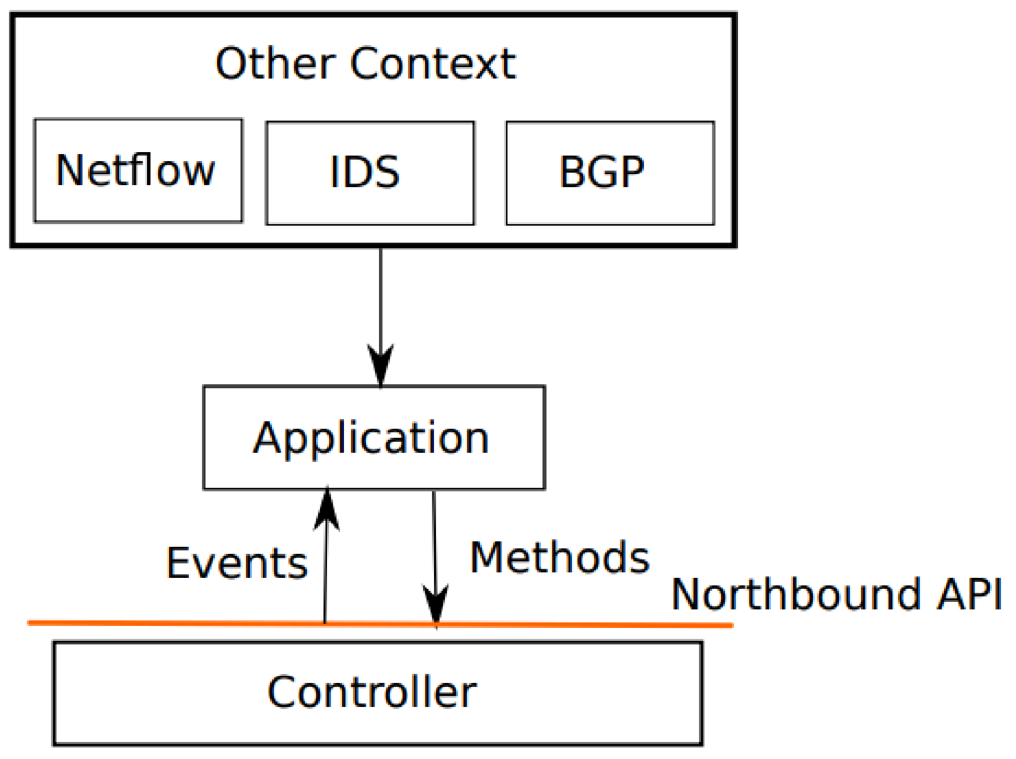

Figure 4.6: Application-Controller communication [GoBl 14]

The control functions of the network (e.g., MAC learning of switches, routing, enforcement of QoS, security...) are programmed in applications, which logically reside above the controller. Applications can change the configuration of switches via the controller's northbound API.

网络的控制功能（例如交换机的MAC学习、路由、QoS实施、安全性等）由应用程序编程实现，这些应用程序逻辑上位于控制器之上。应用程序可以通过控制器的北向API更改交换机的配置。

Taking a simple routing application as an example [KRV+ 15]: the logic of this application is to define the path through which packets flow from a point A to a point B. To achieve this goal, the routing application must, based on the topology input, decide on the path to use and instruct the controller to install the respective forwarding rules in all devices on the chosen path from A to B.

以一个简单的路由应用为例 [KRV+ 15]：该应用程序的逻辑是定义从A点到B点的数据包流经的路径。为了实现这一目标，路由应用程序需要根据拓扑输入确定要使用的路径，并指示控制器在所选路径上的所有设备中安装相应的转发规则，从A到B。

Once the controller has finished initializing devices and reported the network topology to the application, the application spends most of its processing time responding to events. Application behavior is driven by events from the controller as well as external inputs. The application affects the network by responding to the events as modeled in Figure 4.6.

一旦控制器完成设备初始化并将网络拓扑报告给应用程序，应用程序的大部分处理时间将用于响应事件。应用程序的行为由来自控制器的事件以及外部输入驱动。通过响应事件，应用程序影响网络的行为，这在图4.6中有所建模。

The SDN application registers as a listener for certain events, and the controller invokes the application’s callback method whenever such an event occurs. Some examples of events handled by an SDN application are:
SDN应用程序注册为某些事件的侦听器，控制器在这些事件发生时会调用应用程序的回调方法。SDN应用程序处理的事件包括：

- End-user device discovery,
- Network device discovery,
- Incoming packets.

In the first two cases, events are sent to the SDN application when a new end-user device (e.g., a MAC address) or a new network device (e.g., a switch, router, or wireless access point) is discovered. Incoming packet events are sent to the SDN application when a packet is received from an SDN device either due to a flow entry instructing the device to forward the packet to the controller or because no matching flow entry exists in the SDN device.

在前两种情况下，当发现新的终端用户设备（例如MAC地址）或新的网络设备（例如交换机、路由器或无线接入点）时，事件会发送到SDN应用程序。当从SDN设备接收到数据包时，也会发送数据包事件到SDN应用程序，这可能是由于流表条目指示SDN设备将数据包转发到控制器，或者因为SDN设备中没有匹配的流表条目。

When there is no matching flow entry, the default action is usually to forward the packet to the controller. However, depending on the nature of the application, the packet may instead be dropped [GoBl 14].

当没有匹配的流表条目时，默认操作通常是将数据包转发到控制器。然而，根据应用程序的性质，也可能选择丢弃数据包 [GoBl 14]。

4.3 Flows

In SDN, forwarding decisions are flow-based, instead of destination-based as in traditional networks. There is often confusion between the two concepts of path and flow, which need to be clearly distinguished and used correctly.

在SDN中，转发决策是基于流而非传统网络中的基于目的地。路径和流这两个概念经常被混淆，需要明确区分并正确使用。

A **path** is the sequence of communication links and devices connecting two endpoints. It is also known as a route and is the foundation of IP routing. The concept of path is topology-oriented, meaning it focuses only on the way through the network, without considering the type of traffic or its content (payload).

路径 是连接两个端点的通信链路和设备的序列。它也被称为路由，是IP路由的基础。路径的概念是拓扑导向的，这意味着它只关注通过网络的路径，而不关注流量的类型和内容（有效负载）。

A **flow**, or data stream, is a sequence of packets that share the same attributes. These attributes include fields in the protocol header, such as IP address, MAC address, status bit, etc.; they can also include the ingress port of a packet arriving at a network device. Thus, the concept of flow encompasses traffic types and possibly traffic content (payload). Flows are unidirectional and are determined by:
流 或数据流，是一系列共享相同属性的数据包。这些属性包括协议头中的字段，例如IP地址、MAC地址、状态位等；也可以包括到达网络设备的数据包的入口端口。因此，流的概念包含流量类型，甚至可能包括流量内容（有效负载）。流是单向的，由以下三个方面决定：

- The path(路径),
- The concerned attributes(相关属性),
- The direction (which can be considered a special case of the attributes, like address)方向（可视为属性的特殊情况，例如地址）.

Example: Traffic between two endpoints A and B exchanging audio streams and HTTP traffic can be seen as four different flows:

1. A flow for audio traffic along the path from A to B,
2. A flow for audio traffic along the path from B to A,
3. A flow for HTTP traffic along the path from A to B,
4. A flow for HTTP traffic along the path from B to A.

示例： 在两个端点A和B之间交换音频流和HTTP流量时，可以看作四个不同的流：

1. 从A到B的音频流量流，
2. 从B到A的音频流量流，
3. 从A到B的HTTP流量流，
4. 从B到A的HTTP流量流。

Figure 4.7 [Danc 15] illustrates how four data streams are routed in a traditional network (left picture) and in an SDN (right picture).

图4.7 [Danc 15] 说明了四个数据流如何在传统网络（左图）和SDN（右图）中路由。

- In a traditional network, data streams destined for the same destination are typically routed along the same path because the routing mechanism is destination-based.
  在传统网络中，发送到同一目的地的数据流通常沿同一路径路由，因为路由机制是基于目的地的。
- In SDN, these data streams are routed based on their flows' attributes, which means they may traverse different paths from the source to the destination.
  在SDN中，这些数据流是根据其流的属性路由的，因此它们可能沿不同路径从源到达目的地。

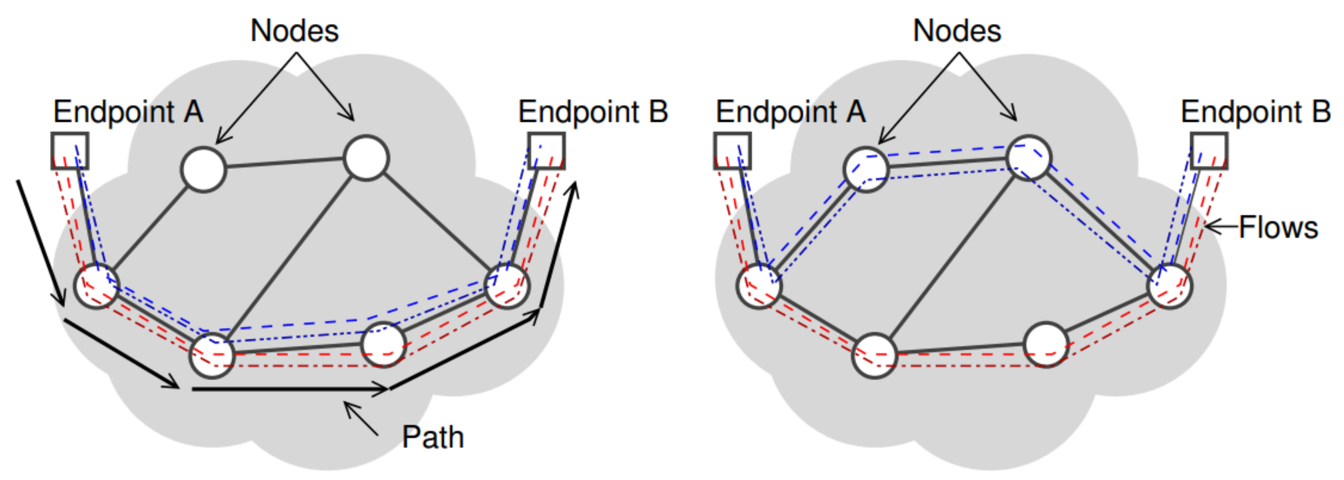

Figure 4.7: Flows along different paths in the traditional network (left) and in SDN [Danc 15]

It is important to note that a flow can be reflected differently in each device along the path. For instance, one network device might handle the flow based on the first attribute (e.g., destination IP address), while another might base its handling on the second attribute (e.g., destination TCP port number).

需要注意的是，一个流在路径上的每个设备中可能有不同的表现。例如，一个网络设备可能基于流的第一个属性（例如目的地IP地址）处理流，而另一个设备可能基于第二个属性（例如目的地TCP端口号）处理流。

Specification of Flow 流的规范

An SDN device stores information about flows in tables, referred to as flow tables or rule tables. Each table entry contains:
SDN设备将流的信息存储在表中，称为流表或规则表。每个表条目包含：

- **Match fields**, which define the flow based on protocol fields (e.g., IP, MAC, port, VLAN, etc.). **匹配字段**，基于协议字段（例如IP、MAC、端口、VLAN等）定义流。
- **Actions**, which are executed when the match fields are satisfied. **操作**，当匹配字段满足时执行相应的操作。

A match field can be wildcarded to match any value, thereby reducing the number of table entries.
匹配字段可以使用通配符匹配任意值，从而减少表条目的数量。

The actions applied to a matched flow can include:
匹配流所应用的操作可能包括：

- Sending the packets of that flow to a specific port (interface of the SDN device). 将该流的数据包发送到指定端口（SDN设备的接口）。
- Dropping the packets. 丢弃数据包。
- Forwarding the packets to the controller. 将数据包转发到控制器。
- Sending packets to some or all ports (flooding, broadcast, multicast). 将数据包发送到部分或所有端口（泛洪、广播、多播）。
- Modifying the packets. 修改数据包。
- A combination of the above actions. 以上操作的某种组合。

Priority 优先级
Table entries are organized in rows, each with a specified priority. More specific entries should have higher priority than general ones (which have more masked fields).
表条目按行组织，每行具有指定的优先级。更具体的条目应比更一般的条目（即具有更多掩码字段的条目）具有更高的优先级。

4.4 SDN Deployment Example with P4 and P4Runtime

The most common implementations of SDN include those based on OpenFlow [MAB+ 08] and P4 [BDG+ 14].
SDN最常见的实现包括基于OpenFlow [MAB+ 08] 和P4 [BDG+ 14] 的实现。

OpenFlow assumes that SDN devices have fixed behavior. It supports populating the rule tables in these devices but is constrained by their fixed capabilities. OpenFlow cannot be used to deploy a rule that influences a new packet header not yet available in these fixed-function devices.

OpenFlow假设SDN设备具有固定的行为。它支持在这些设备中填充规则表，但受其固定功能的限制。无法使用OpenFlow部署影响固定功能设备中尚未支持的新数据包头的规则。

P4 was introduced to address this limitation by providing a mechanism to program devices' behavior. P4Runtime enables communication between SDN controllers and P4 devices.P4的引入旨在解决这一问题，它提供了一种编程设备行为的机制。P4Runtime实现了SDN控制器与P4设备之间的通信。

4.4.1 P4

P4 is a language for programming the data plane of network devices. The name P4 comes from the original paper that introduced the language: *"Programming Protocol-independent Packet Processors"* [BDG+ 14]. The version of P4 introduced in 2014 is named P414, and the language was revised in 2016 to P416, which is used in this assignment.
P4是一种用于编程网络设备数据平面的语言。P4的名称来源于该语言的原创论文：*"Programming Protocol-independent Packet Processors"* [BDG+ 14]。2014年发布的P4版本被称为P414，该语言在2016年进行了修订，被称为P416，本次作业中使用的就是P416版本。

In a traditional switch, the manufacturer defines the data-plane functionality. The control plane manages the data plane by:
在传统交换机中，数据平面的功能由制造商定义。控制平面通过以下方式管理数据平面：

- Managing entries in their rule tables (e.g., routing tables), 管理规则表（例如路由表）中的条目，
- Configuring specialized objects (e.g., meters), 配置专用对象（例如计量器），
- Processing control packets (e.g., routing protocol packets), 处理控制数据包（例如路由协议数据包），
- Handling asynchronous events (e.g., link state changes or learning notifications). 处理异步事件（例如链路状态变化或学习通知）。

A P4-programmable switch differs from a traditional switch in two essential ways:
P4可编程交换机与传统交换机有两个主要区别：

1. **Data plane functionality is not predefined**: The functionality is defined by a P4 program. The data plane is configured at initialization to implement the behavior described by the P4 program and has no built-in knowledge of existing network protocols.

数据平面的功能不是预先定义的：其功能由P4程序定义。数据平面在初始化时配置，以实现P4程序描述的行为，并且对现有网络协议没有内置知识。

2. **Dynamic communication with the control plane**: The control plane communicates with the data plane using the same channels as in fixed-function devices, but the tables and objects in the data plane are not fixed. Instead, they are defined by a P4 program. The P4 compiler generates the API used by the control plane to interact with the data plane.
   与控制平面的动态通信：控制平面通过与固定功能设备相同的通道与数据平面通信，但数据平面中的表和对象不再固定，而是由P4程序定义的。P4编译器生成控制平面与数据平面交互所使用的API。

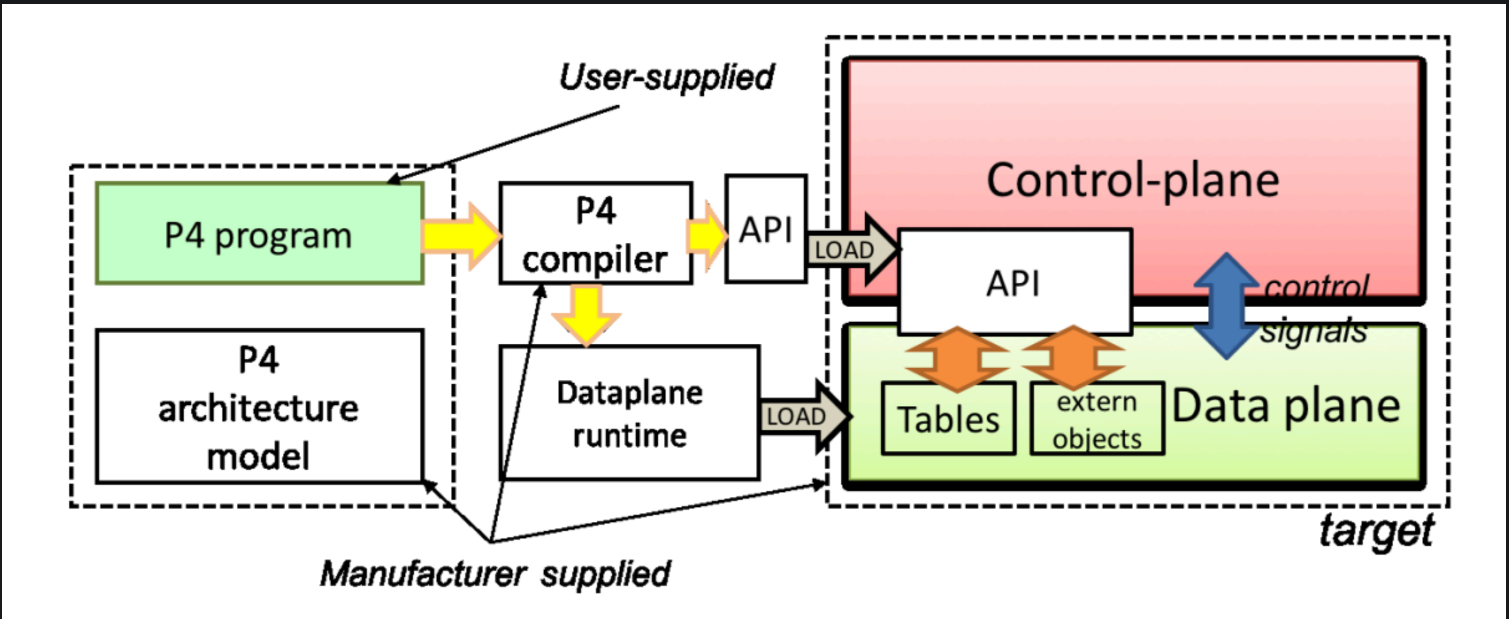

Figure 4.8: Programming a target with P4 [P4Spec]

Figure 4.8 shows a typical tool workflow for programming a target using P4. A *target* is defined as a packet-processing system capable of executing a P4 program. The term *target* is used interchangeably with *device* in most P4 specifications and in this manuscript.
图4.8 展示了使用P4编程目标设备的典型工具工作流程。*目标设备* 被定义为能够执行P4程序的数据包处理系统。在大多数P4规范以及本手稿中，*目标设备* 与*设备*的术语可以互换使用。

Target manufacturers provide the hardware or software implementation framework, an architecture definition, and a P4 compiler for the target. P4 programmers write programs specific to a target architecture, which defines a set of P4-programmable components and their external data plane interfaces.
目标设备的制造商提供硬件或软件实现框架、架构定义和针对目标设备的P4编译器。P4程序员为特定的架构编写程序，该架构定义了目标设备上可用的一组P4可编程组件及其外部数据平面接口。

Compiling a set of P4 programs produces two artifacts:
编译一组P4程序会生成两个成果：

1. A **data plane configuration** implementing the forwarding logic described in the program.
   数据平面配置，用于实现输入程序中描述的转发逻辑。
2. An **API** for managing the state of the data plane objects from the control plane.
   API，用于从控制平面管理数据平面对象的状态。

Architecture Model

The P4 architecture identifies the P4-programmable blocks (e.g., parser, ingress control flow, egress control flow, deparser, etc.) and their data plane interfaces.

P4架构定义了P4可编程模块（例如解析器、入口控制流、出口控制流、反解析器等）及其数据平面接口。

The P4 architecture can be thought of as a contract between the program and the target. Each manufacturer must provide both a P4 compiler and an accompanying architecture definition for their target.

P4架构可以被视为程序与目标设备之间的契约。因此，每个制造商必须为其目标设备提供P4编译器及其配套的架构定义。

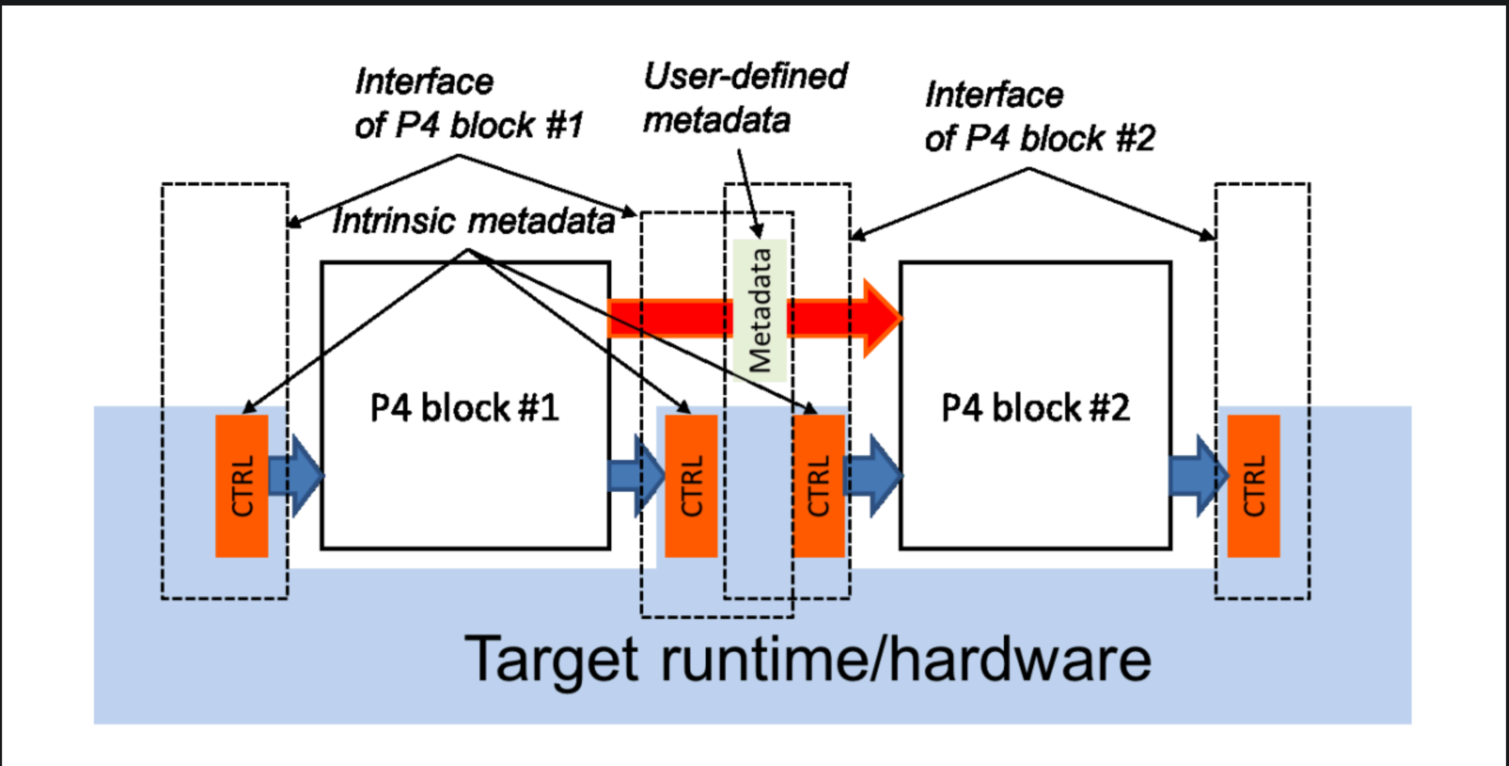

Figure 4.9: P4 program interfaces

Figure 4.9 illustrates the data plane interfaces between P4-programmable blocks. It depicts a target with two programmable blocks (#1 and #2), each of which is programmed through a separate fragment of P4 code. The target interfaces with the P4 program via a set of control registers or signals:
图4.9 展示了P4可编程模块之间的数据平面接口。图中显示了一个包含两个可编程模块（#1和#2）的目标设备，每个模块通过一个独立的P4代码片段进行编程。目标设备通过一组控制寄存器或信号与P4程序交互：

- **Input controls** provide information to P4 programs (e.g., the input port a packet was received from). **输入控制** 为P4程序提供信息（例如，接收到数据包的输入端口）。
- **Output controls** can be written to by P4 programs to influence the target's behavior (e.g., the output port where a packet should be directed). **输出控制** 可由P4程序写入以影响目标设备的行为（例如，数据包应被引导到的输出端口）。

Control registers/signals are represented in P4 as **intrinsic metadata**, while P4 programs can also define and manipulate **user-defined metadata** to store data related to each packet.

控制寄存器/信号在P4中表示为**内在元数据**，而P4程序还可以定义并操作**用户定义元数据**以存储与每个数据包相关的数据。

There are various architectures, such as **V1Model**, **SimpleSumeSwitch**, and the **Portable Switch Architecture (PSA)** [PSADraft]. In this assignment, we use the **V1Model** on the BMv2 (Behavioral Model version 2) Simple Switch target because of its ease of deployment as a software switch.

存在多种架构，例如 **V1Model**、**SimpleSumeSwitch** 和 **可移植交换架构 (PSA)** [PSADraft]。在本次作业中，由于易于作为软件交换机部署，我们使用了 **BMv2（行为模型第2版）Simple Switch目标** 上的 **V1Model**。

V1Model

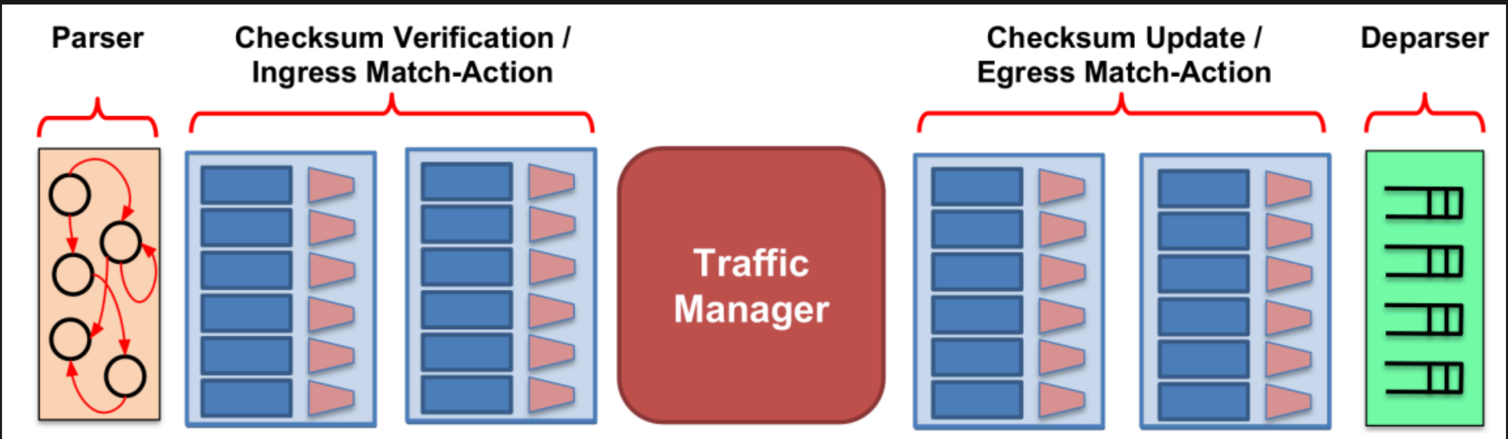
Parser
Checksum Verification / Ingress Match-Action
Traffic Manager
Checksum Update / Egress Match-Action
Deparser

Figure 4.10: V1Model architecture

Figure 4.10 illustrates the programmable blocks of the V1Model architecture. The **standard metadata** in V1Model includes:

```
struct standard_metadata_t {
    bit<9> ingress_port;
    bit<9> egress_spec;
    bit<9> egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1> drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    bit<19> deq_qdepth;
    bit<48> ingress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<1> resubmit_flag;
    bit<16> egress_rid;
    bit<1> checksum_error;
    bit<3> priority;
}
```

The **ingress_port** is the port on which a packet arrived. The **egress_spec** specifies the port to which the packet should be sent. The **egress_port** denotes the port from which the packet is departing and is read-only in the egress pipeline.
ingress_port 是数据包到达的端口。**egress_spec** 指定数据包应被发送到的端口。**egress_port** 表示数据包离开的端口，在出口管道中是只读的。

The file `v1model.p4` is extensively commented, providing details on these and other fields of the standard metadata.

文件 `v1model.p4` 中有详细的注释，提供了关于这些字段及其他标准元数据字段的详细信息。

The **P416 program template** for the V1Model architecture includes:
P416程序模板 适用于V1Model架构，包含以下部分：

- **头部规范 (HEADER specification)**
- **解析器 (PARSER)**
- **校验和验证 (CHECKSUM VERIFICATION)**
- **入口处理 (INGRESS PROCESSING)**
- **出口处理 (EGRESS PROCESSING)**
- **校验和更新 (CHECKSUM UPDATE)**
- **反解析器块 (DEPARSER blocks)**

- **V1Switch包**

```
#include <core.p4>
#include <v1model.p4>

/* 头部定义 */
struct metadata { ... }
struct headers {
    ethernet_t ethernet;
    ipv4_t ipv4;
}

/* 解析器 */
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t smeta) {
    ...
}

/* 校验和验证 */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
    ...
}

/* 入口处理 */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
    ...
}

/* 出口处理 */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
    ...
}

/* 反解析器 */
control MyDeparser(packet_out packet,
                     in headers hdr) {
    ...
}
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
    ...
}

/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
    ...
}

/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
    ...
}

/* SWITCH */
V1Switch(
    MyParser(),
    MyVerifyChecksum(),
    MyIngress(),
    MyEgress(),
    MyComputeChecksum(),
    MyDeparser()
) main;

```

P4_{16} defines various data types, such as `bit`, `bool`, `int`, `string`, `struct`, `enum`, and `header_union`. These data types are described in its specification [P4Spec] (see Section 7 of the specification).

P416定义了多种数据类型，例如 `bit`、`bool`、`int`、`string`、`struct`、`enum` 和 `header_union`。这些数据类型在其规范 [P4Spec]（参见规范第7节）中有详细说明。

Another useful reference is the **P4 Language Cheat Sheet**, which outlines the basic building blocks of P4.

另一个有用的参考资料是 **P4语言速查表 (P4 Language Cheat Sheet)**，其中概述了P4的基本构建模块。

An example of compiling a P4 source file into a **P4 Device Config** file, which can be executed in BMv2 switches, is provided in a later section.

有关将P4源文件编译为**P4设备配置 (P4 Device Config)** 文件，并在BMv2交换机中执行的示例，将在后续章节中提供。

4.4.2 P4Runtime

The **P4Runtime API** [P4RTSpec] is a control plane specification for managing the data plane elements of a device defined or described by a P4 program.

P4Runtime API [P4RTSpec] 是一种控制平面规范，用于管理由P4程序定义或描述的设备数据平面元素。

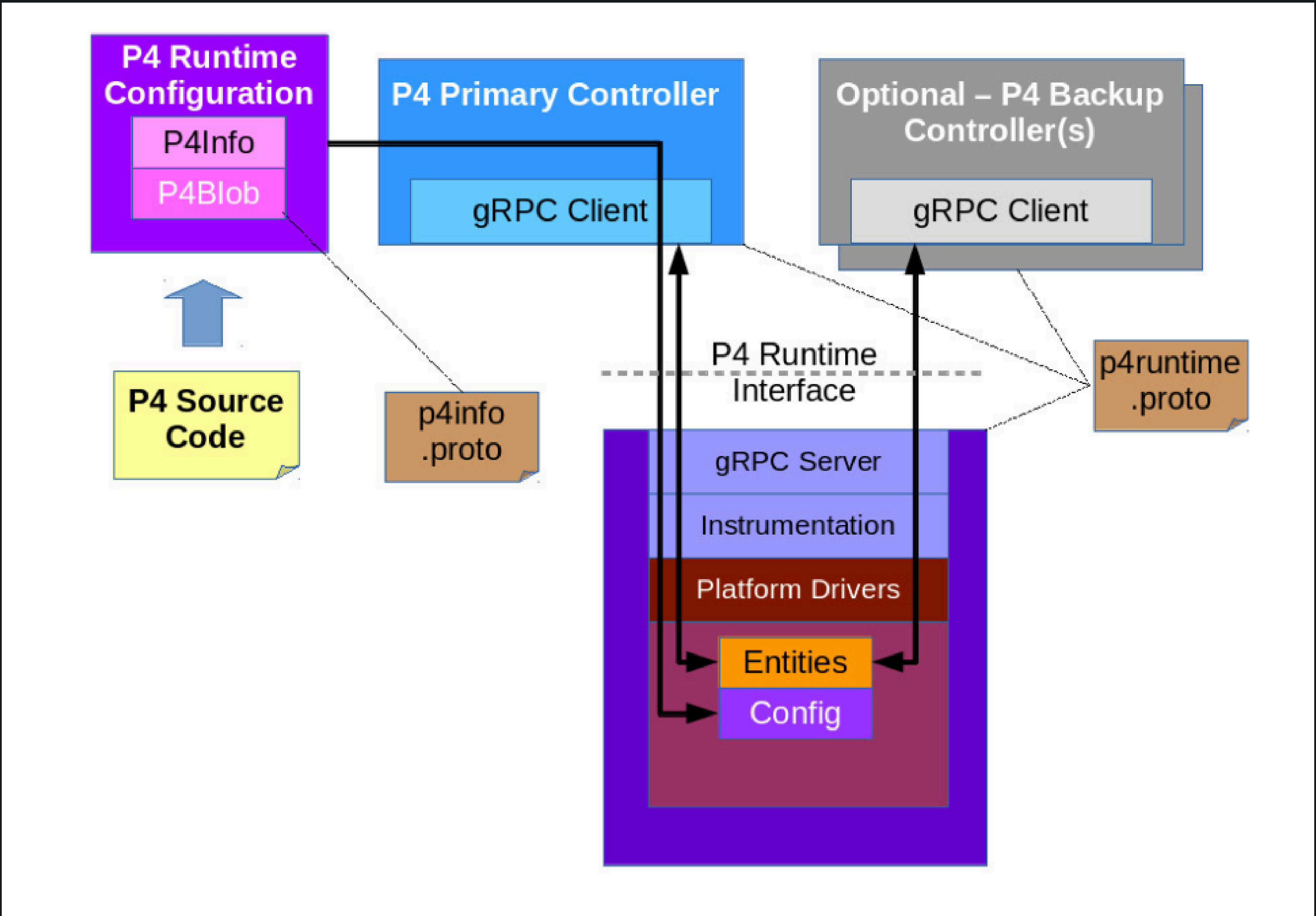

Figure 4.11: P4Runtime reference architecture [P4RTSpec]

Figure 4.11 illustrates the P4Runtime reference architecture. The device or target to be controlled is at the bottom, while one or more controllers are shown at the top. P4Runtime grants **write access** to only a single **primary controller** for each read/write entity.

图4.11 展示了P4Runtime参考架构。需要控制的设备或目标位于底部，而一个或多个控制器位于顶部。对于每个读/写实体，P4Runtime仅授予一个**主控制器**的**写访问权限**。

The P4Runtime API defines the messages and semantics of the interface between the client(s) and the server. The API is specified in the `p4runtime.proto` **Protobuf file**.

P4Runtime API 定义了客户端和服务器之间接口的消息和语义。该API在 `p4runtime.proto` Protobuf文件中进行了规范化。

The controller can access P4 entities that are declared in the **P4Info metadata**.

控制器可以访问在 **P4Info元数据** 中声明的P4实体。

The **P4Info structure** is defined in the `p4info.proto` file, another Protobuf file included as part of the P4Runtime standard.

P4Info结构 定义在 `p4info.proto` 文件中，这是P4Runtime标准的一部分。

The controller can set the **ForwardingPipelineConfig**, which involves:

控制器可以设置 **ForwardingPipelineConfig**，包括以下内容：

- Installing and running the compiled P4 program output included in the `p4_device_config` field of the Protobuf message.
  安装并运行包含在 Protobuf 消息 p4_device_config 字段中的已编译P4程序输出。
- Installing the associated **P4Info metadata**.
  安装相关的 **P4Info元数据。**

The controller can also query the target for the **ForwardingPipelineConfig** to retrieve the device configuration and the P4Info.

控制器还可以查询目标设备的 **ForwardingPipelineConfig** 以获取设备配置和P4Info。

The **P4Runtime API** is implemented by a program running a gRPC server that binds to an auto-generated P4Runtime Service interface. This program is called the **P4Runtime server**. By default, the server must listen on **TCP port 9559**, allocated by IANA for the P4Runtime service. However, servers should allow users to override the default port using a configuration file or a startup flag.

P4Runtime API 通过运行一个绑定自动生成的 P4Runtime 服务接口的 gRPC 服务器实现。这个程序被称为 **P4Runtime服务器。**默认情况下，服务器必须监听 **TCP端口9559**，这是IANA为P4Runtime服务分配的端口。不过，服务器应允许用户通过配置文件或启动标志覆盖默认端口。

In this assignment, we follow the **"idealized workflow"** described in the P4Runtime specification [P4RTSpec] (Section 3.2). This workflow includes:在本次作业中，我们遵循P4Runtime规范 [P4RTSpec] 中描述的 **"理想化工作流程"**（参见规范第3.2节）。该工作流程包括：

1. Compiling a P4 source program to produce: 将P4源程序编译为：
   - A **P4 device config**. **P4设备配置。**
   - **P4Info metadata**. **P4Info元数据。**
     These together comprise the **ForwardingPipelineConfig** message.
     这些共同组成 **ForwardingPipelineConfig** 消息。
2. The P4Runtime controller selects an appropriate configuration for a particular target and installs it via the **SetForwardingPipelineConfig RPC**.
   P4Runtime控制器选择适合特定目标设备的配置，并通过 **SetForwardingPipelineConfig RPC** 安装它。

The **P4Info metadata** describes: **P4Info元数据** 描述了：

- The overall program itself (**PkgInfo**). 整个程序本身（**PkgInfo**）。
- All entity instances derived from the P4 program, such as **tables** and **extern instances**. 所有从P4程序派生的实体实例，例如 **表** 和 **外部实例。**

Each entity instance is assigned a numeric ID by the P4 compiler, which serves as a concise "handle" for use in API calls.
每个实体实例由P4编译器分配一个数字ID，用作API调用中使用的简洁"句柄"。

In this workflow, **P4 compiler backends** are developed for each unique type of target and generate two outputs:
在此工作流程中，为每种独特类型的目标设备开发了 **P4编译器后端**，并生成两个输出：

- **P4Info**: A schema that is target- and architecture-independent, although its specific contents are likely to be architecture-dependent.
  P4Info：一个目标和架构无关的模式，但其具体内容可能与架构相关。
- A **target-specific device config**.
  特定目标设备的配置。

The compiler ensures that the P4 program is compatible with the specific target and rejects incompatible code.
编译器确保P4程序与特定目标兼容，并拒绝不兼容的代码。

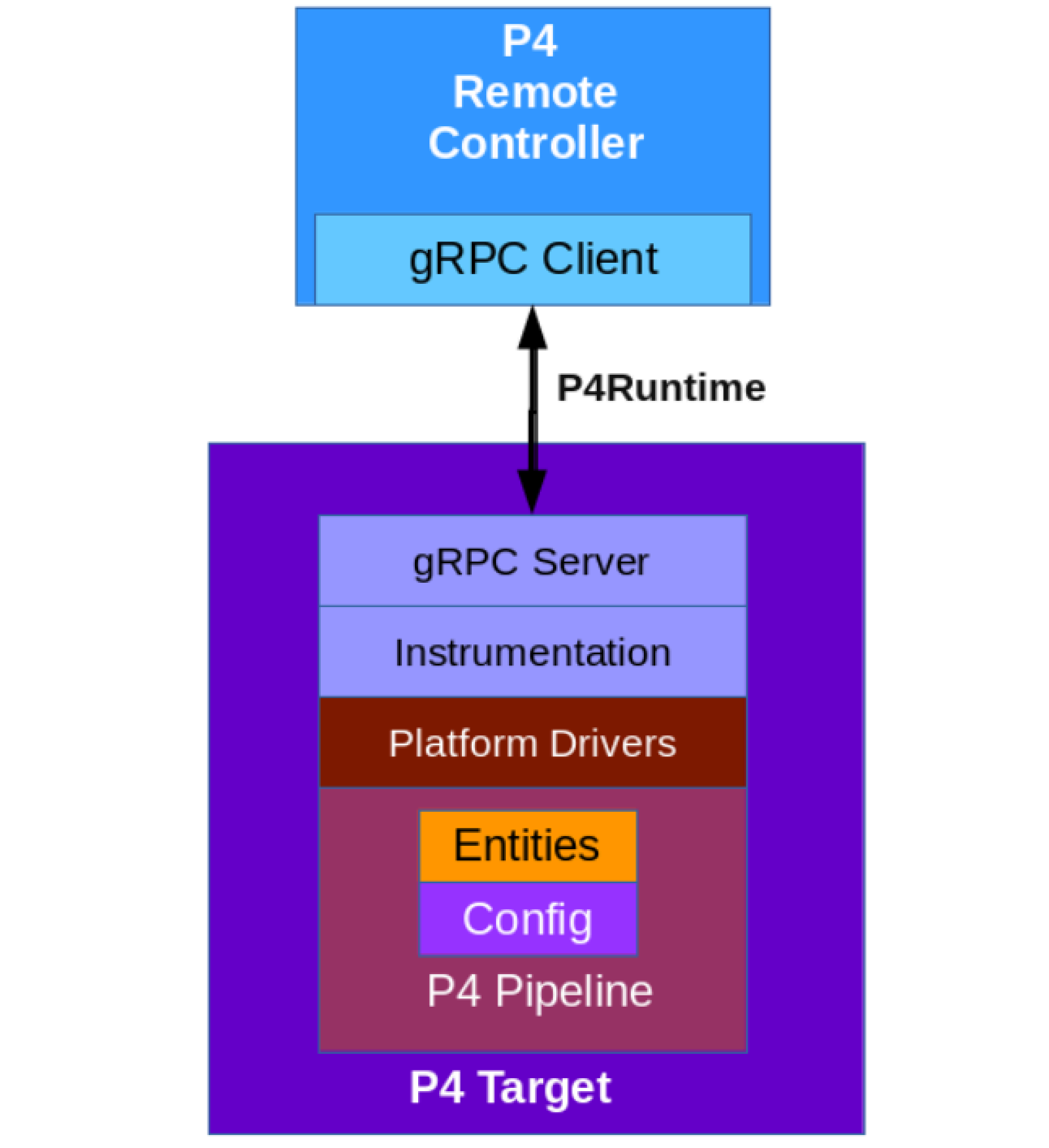

Figure 4.12: Single remote controller [P4RTSpec]

P4Runtime supports the use of multiple controllers. A controller can either be embedded in a P4 target or operate separately. For P4-based SDN, the preferred approach is to deploy controllers separately from the targets, as shown in **Figure 4.12**.
P4Runtime 支持多个控制器的使用。控制器可以嵌入在P4目标设备中，也可以独立运行。对于基于P4的SDN，我们选择将控制器与目标设备分离部署，如 **图4.12** 所示。

A single **logically centralized controller** can control multiple targets simultaneously.
一个 **逻辑集中式控制器** 可以同时控制多个目标设备。

4.4.3 Test-bed

The provided infrastructure for this assignment includes an **“outer” virtual machine** (`gruppeX.rnp.lab.nm.ifi.lmu.de`), which hosts seven **“inner” virtual machines**. These inner machines include:
该作业提供的基础设施包括一个 **“外部”虚拟机** (`gruppeX.rnp.lab.nm.ifi.lmu.de`)，其中创建了七个 **“内部”虚拟机**，包括：

- Hosts: `pc1`, `pc2`, `pc3`.
- Switches: `S1`, `S2`, `S3`.
- A controller deployed at `router4`.

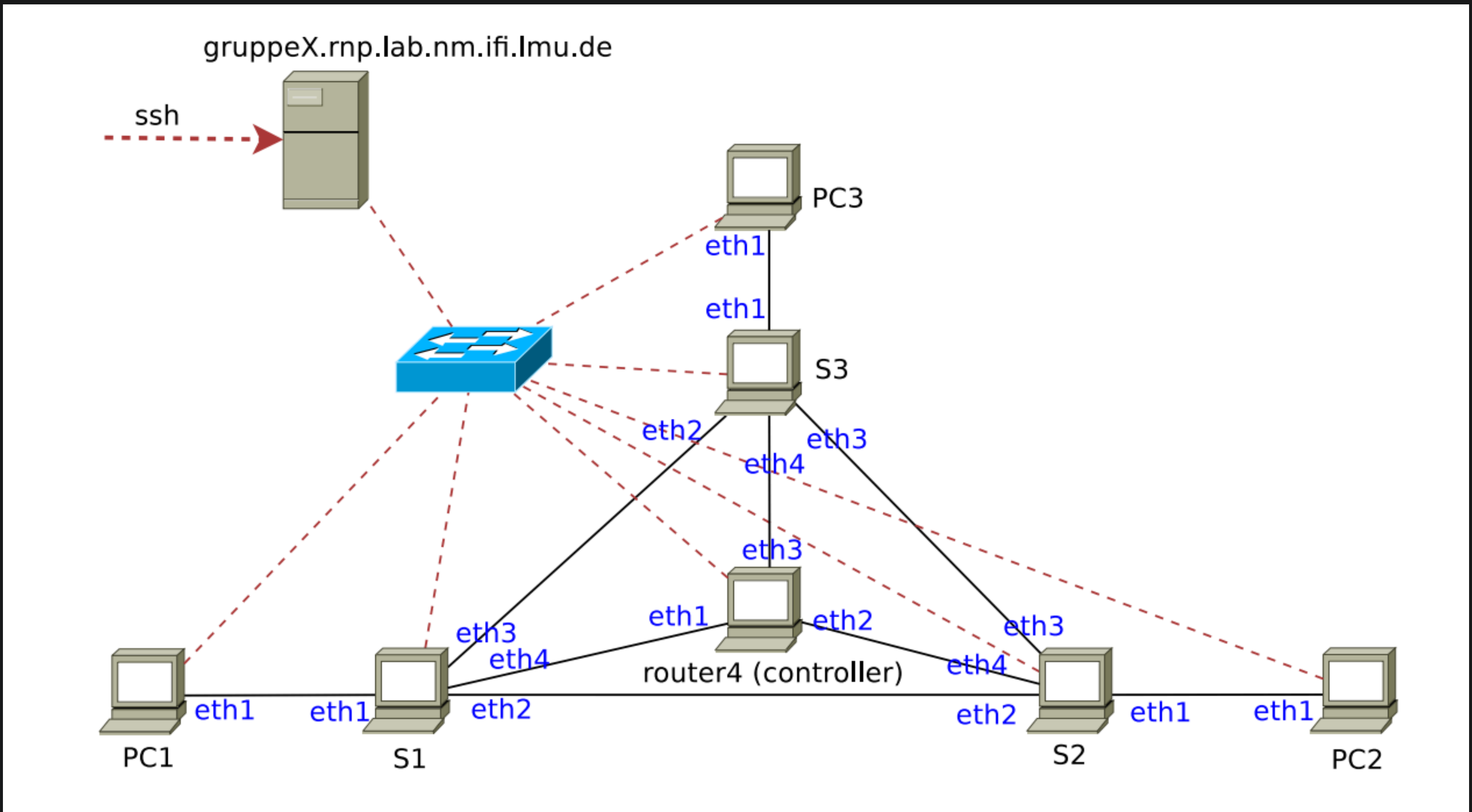

Figure 4.13: Provided RNP infrastructure

The connections between these machines are illustrated in **Figure 4.13**: 这些机器之间的连接如 **图4.13** 所示：

- **Dashed lines**: Represent the management network, used for accessing and configuring the inner machines. **虚线**：表示管理网络，用于访问和配置内部机器。
- **Solid lines**: Represent the data network, used for communication between the inner machines.
  All `eth0` interfaces of the inner machines are connected to the same switch in the management network. **实线**：表示数据网络，用于内部机器之间的通信。所有内部机器的 `eth0` 接口都连接到管理网络的同一交换机。

The infrastructure is pre-configured with all the software necessary for this assignment. The **P4 compiler** is installed only on the controller (`router4`). As a result, P4 source files must be compiled on the controller to generate: 基础设施已预装完成该作业所需的所有软件。**P4编译器**仅安装在控制器（`router4`）上。因此，P4源文件必须在控制器上编译，以生成：

- **P4 Device Config**: Consumed by P4 switches. **P4设备配置文件**：供P4交换机使用。
- **P4Info files**: Consumed by controllers. **P4Info文件**：供控制器使用。

Instructions for installing P4-related software are provided in the **P4 language tutorials**, which include a sample deployment on **Ubuntu 20.04**. Additionally, the development team has facilitated P4 installation via the **p4lang repository** for package managers on **Debian 11** and **Ubuntu 22.04**. Installation can also be performed directly from the **source code**.

有关安装P4相关软件的说明，可以参考 **P4语言教程**，其中包括 **Ubuntu 20.04** 上的示例部署。此外，开发团队还通过 **p4lang 仓库** 提供了适用于 **Debian 11** 和 **Ubuntu 22.04** 的包管理器安装方式。也可以直接从 **源代码** 安装。

4.4.4 Example: deploying a simple repeater

We demonstrate how to execute P4 switches and have them controlled by a controller using a simple **repeater application**. This example reuses material provided by the **ETH Networked Systems Group**.

我们通过一个简单的 **中继器应用 (repeater application)** 演示如何运行P4交换机，并由控制器进行控制。本示例复用了 **ETH网络系统组 (ETH Networked Systems Group)** 提供的材料。

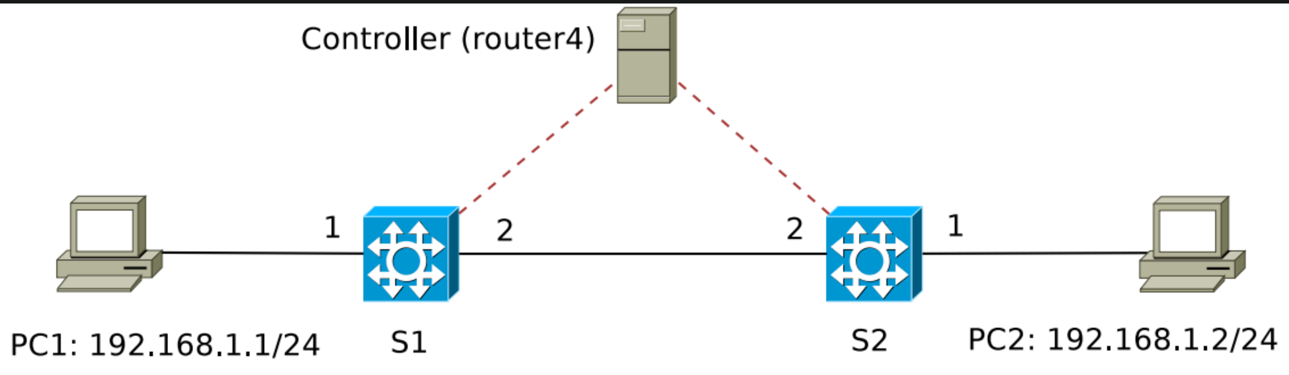

Figure 4.14: Network topology for the deployment example. The numbers surrounding a switch indicate its port names.

Figure 4.14 illustrates the network topology used in this example, which includes:

- Two switches: **S1** and **S2**.
- Two endpoints: **PC1** and **PC2**.

- A **controller**.

The **P4 source files** and the **controller applications** are stored on the controller (`router4`) in the directories `p4` and `app`, respectively.

P4源文件 和 **控制器应用程序** 分别存储在控制器 (`router4`) 的 `p4` 和 `app` 目录中。

Compiling repeater.p4

As mentioned earlier, the P4 compiler is installed on the controller (`router4` in this case). To compile `repeater.p4` and generate the **P4 Device config** and **P4Info files**, execute the following commands:

如前所述，P4编译器安装在控制器上（在本例中为 `router4` ）。要编译 `repeater.p4` 并生成 **P4设备配置文件** 和 **P4Info文件**，请执行以下命令：

```
ssh root@router4
cd p4
p4c-bm2-ss --p4v 16 --p4runtime-files build/repeater.p4info.txt \
           -o build/repeater.json repeater.p4
```

This produces two files:

- `repeater.json`: The P4 Device config file for the P4 switches.
- `repeater.p4info.txt`: The P4Info file for the controller application `repeater_controller.py`.

生成的两个文件分别是：

- `repeater.json`： 用于P4交换机的P4设备配置文件。
- `repeater.p4info.txt`： 控制器应用程序 `repeater_controller.py` 使用的P4Info文件。

Distributing the P4 Device Config File

The `repeater.json` file needs to be copied to the switches `S1` and `S2`. This can be done from the outer machine using the following commands:

需要将 `repeater.json` 文件复制到交换机 `S1` 和 `S2`。可以通过外部机器执行以下命令：

```
scp -3 root@router4:p4/build/repeater.json root@s1:
scp -3 root@router4:p4/build/repeater.json root@s2:
```

Creating P4 Switches

At Switch S1

Use the following commands to create a P4 switch on `S1`:

使用以下命令在 `S1` 上创建一个 P4 交换机：

```
ssh root@s1
ip link set eth1 up
ip link set eth2 up
simple_switch_grpc -i 1@eth1 -i 2@eth2 --pcap pcaps \
--nanolog ipc:///log.ipc --device-id 1 repeater.json \
--log-console --thrift-port 9090 \
-- --grpc-server-addr 0.0.0.0:50051 --cpu-port 255
```

The `simple_switch_grpc` command creates a **SimpleSwitchGrpc target,** which is a version of **SimpleSwitch** with P4Runtime support.

`simple_switch_grpc` 命令创建了一个 **SimpleSwitchGrpc 目标**，这是支持 **P4Runtime** 的 **SimpleSwitch** 版本。

Explanation of parameters:

- `-i 1@eth1 -i 2@eth2`: Binds the switch ports to the "physical" interfaces of the virtual machine. `-i 1@eth1 -i 2@eth2`: 将交换机端口绑定到虚拟机的“物理”接口。
- `--pcap pcaps`: Generates PCAP files for interfaces. `--pcap pcaps`: 为接口生成 PCAP 文件。
- `--nanolog ipc:///log.ipc`: Specifies the IPC socket for nanomsg pub/sub logs. `--nanolog ipc:///log.ipc`: 指定 nanomsg 发布/订阅日志的 IPC 套接字。
- `--device-id 1`: Sets the device ID for identifying the device in IPC messages. `--device-id 1`: 设置设备 ID，用于在 IPC 消息中标识设备。
- `--log-console`: Enables logging to stdout. `--log-console`: 启用标准输出的日志记录。

- `--thrift-port 9090`: Sets the TCP port for the Thrift runtime server. `--thrift-port 9090`: 设置 Thrift 运行时服务器的 TCP 端口。
- `--grpc-server-addr 0.0.0.0:50051`: Binds the gRPC server to the specified address. `--grpc-server-addr 0.0.0.0:50051`: 将 gRPC 服务器绑定到指定地址。
- `--cpu-port 255`: Specifies the logical port where a switch can communicate with a controller (packet-in/out). `--cpu-port 255`: 指定逻辑端口，交换机和控制器可以通过该端口通信（数据包输入/输出）。

For additional information, use the command: 如需更多信息，请使用命令:

```
simple_switch_grpc --help
```

At Switch S2

To create a P4 switch on `S2`, use similar commands: 在 `S2` 上创建 P4 交换机的命令类似:

```
ssh root@s2
ip link set eth1 up
ip link set eth2 up
simple_switch_grpc -i 1@eth1 -i 2@eth2 --pcap pcaps \
--nanolog ipc:///log.ipc --device-id 2 repeater.json \
--log-console --thrift-port 9090 \
-- --grpc-server-addr 0.0.0.0:50051 --cpu-port 255
```

Executing the Control Application

To execute the control application: 运行控制应用程序:

```
ssh root@router4
cd app
python3 repeater_controller.py
```

The **p4-utils library** is used (with some modifications for packet-in, packet-out, and idle timeout support) to implement control applications. It acts as a wrapper for **p4runtime-shell**, simplifying controller plane development.

p4-utils库（经过一些修改以支持 packet-in、packet-out 和 idle timeout）用于实现控制应用程序。它作为 **p4runtime-shell** 的封装器，简化了控制平面的开发。

Useful API information (e.g., `table_add`, `table_delete_match`) is available on the linked **GitHub repository**.

有关 API（如 `table_add`、 `table_delete_match`）的详细信息，可以在 **GitHub 仓库** 中找到。

The **repeater application** uses the network topology information encoded in `topo_repeater.json`. For other control applications, you need to provide the relevant topology information in a similar file.

repeater 应用程序 使用编码在 `topo_repeater.json` 中的网络拓扑信息。对于其他控制应用程序，需要在类似文件中提供相关的网络拓扑信息。

Generating Traffic Between End-Points

After deploying the repeater application, you can test its functionality by generating traffic between `PC1` and `PC2`. 部署 repeater 应用程序后，可以通过在 `PC1` 和 `PC2` 之间生成流量来测试其功能。

1. Configure the `eth1` interface on **PC1**:

   ```
   ssh root@pc1
   ip addr add 192.168.1.1/24 broadcast 192.168.1.255 dev eth1
   ip link set eth1 up
   ```

2. Configure the `eth1` interface on **PC2** and generate traffic using the `ping` command:

   ```
   ssh root@pc2
   ip addr add 192.168.1.2/24 broadcast 192.168.1.255 dev eth1
   ip link set eth1 up
   ping 192.168.1.1
   ```

If the PCs successfully communicate via `ping`, the deployment of the repeater application is confirmed.

如果 `PC1` 和 `PC2` 能够通过 `ping` 互相通信，则说明 repeater 应用程序部署成功。

Debugging the Data Plane Using `simple_switch_CLI`

The `simple_switch_CLI` command is useful for debugging the data plane. For example, while `S1` is running, open another terminal to observe or manipulate its rule tables: `simple_switch_CLI` 命令可用于调试数据平面。例如，当 `S1` 正在运行时，可以打开另一个终端窗口观察或操作其规则表：

1. Access **switch S1**:

   ```
   ssh root@s1
   simple_switch_CLI
   ```

2. Use commands to debug:
   - Show existing tables:

     ```
     show_tables
     ```

   - Show contents of a specific rule table (e.g., `repeater`):

     ```
     table_dump repeater
     ```

   - Add a rule to the `repeater` table:

     ```
     table_add repeater forward 1 => 2
     ```

     This adds a rule with `match key = 1` , `action = forward`, and `action parameter = 2`.

More commands and details can be found in the **P4 language GitHub repository**.

更多命令和详细信息可在 **P4 language GitHub 仓库** 中找到。

4.4.5 Other P4 Sources and Control Applications for Reference

An advanced control application, `arpcache.py`, is provided on `router4` alongside the earlier example. Its corresponding P4 source file is `packetinout.p4`.

一个高级控制应用程序 `arpcache.py` 与前面的示例一起提供在 `router4` 上。其对应的 P4 源文件为 `packetinout.p4`。

Additional useful examples are available in the **MNM-GitHub repository** MNM-GitHub. These examples should be practiced in the order specified on the repository's main page (`README.md`) for a better understanding and smooth progression. Start with the **simple_demo application**, then proceed to **simple_switch**, and so on.

更多有用的示例可以在 **MNM-GitHub 仓库** MNM-GitHub 中找到。这些示例应按照仓库主页面 (`README.md`) 中指定的顺序进行练习，以便更好地理解和顺利过渡。从 **simple_demo 应用程序** 开始，然后依次进行 **simple_switch** 等其他示例。

Other P4 sources and control applications can also be found in:

其他 P4 源文件和控制应用程序也可以在以下链接找到：

- The **P4 language tutorials** P4 Tutorials.
- The **p4-learning** repository P4 Learning.

4.5 Assignment

Students will "program" the provided infrastructure using P4-based SDN to meet specific requirements based on a practical scenario.

学生将使用基于 P4 的 SDN 对提供的基础设施进行编程，以满足基于实际场景的特定需求。

4.5.1 Scenario

The Academic Affairs Office (AAO) of University A has:

1. A **web server** hosting the AAO-Website, publishing important student information.
2. An **E-learning Server** providing experimental e-learning services, such as lectures, exercises, and evaluations.

大学 A 的教务办公室 (AAO) 拥有以下资源：

1. 一个 **Web 服务器**，托管 AAO 网站，用于发布重要的学生信息。
2. 一个 **E-learning 服务器**，提供实验性的在线学习服务，如讲座、练习和评估。

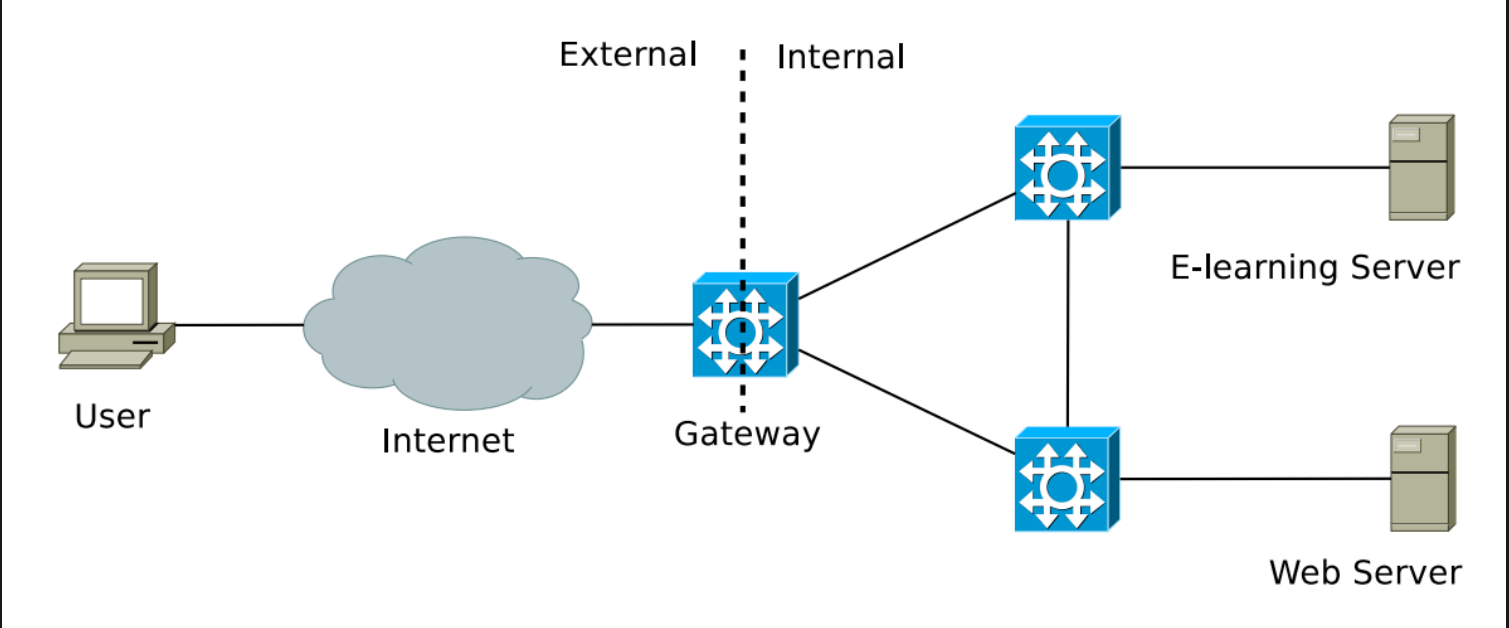

Figure 4.15: Simplified university network with two servers

The network containing these servers is shown in **Figure 4.15**.

- The **E-learning Server** is underutilized since the service is in the testing phase.
- The **web server** experiences high loads, especially during university entrance exam result publications.

包含这些服务器的网络如 **图 4.15** 所示：

- 由于服务仍处于测试阶段，**E-learning 服务器** 的资源大部分时间处于空闲状态。
- 在大学发布入学考试结果时，**Web 服务器** 的负载意外地增加。

Temporary solution:

1. Deploy an additional instance of the AAO-Website on the **E-learning Server**.
2. Perform **load balancing** at the gateway to distribute traffic flows alternately between the two servers.
3. If the traffic volume reaches **80% of the link bandwidth**, drop the highest-load traffic flow.

The internal network of University A is **SDN-based**. Students, acting as network administrators, will program the network to achieve these requirements.

临时解决方案：

1. 在 **E-learning 服务器** 上部署 AAO 网站的额外实例。
2. 在网关执行 **负载均衡**，将流量交替转发到两个服务器。
3. 如果流量达到 **链路带宽的 80%**，丢弃负载最高的流量。

大学 A 的内部网络基于 **SDN**。学生将作为网络管理员，编程网络以实现这些需求。

4.5.2 Demo

For demonstration purposes, the scenario is simplified, as shown in **Figure 4.16**:

- **PC1** represents the external party.
- **Switch S1** acts as the gateway.
- **Switches S2, S3** and **PC2, PC3** form the internal network.

为了演示目的，将场景简化为 **图 4.16**：

- **PC1** 代表外部实体。
- **交换机 S1** 作为网关。
- **交换机 S2, S3** 和 **PC2, PC3** 构成内部网络。

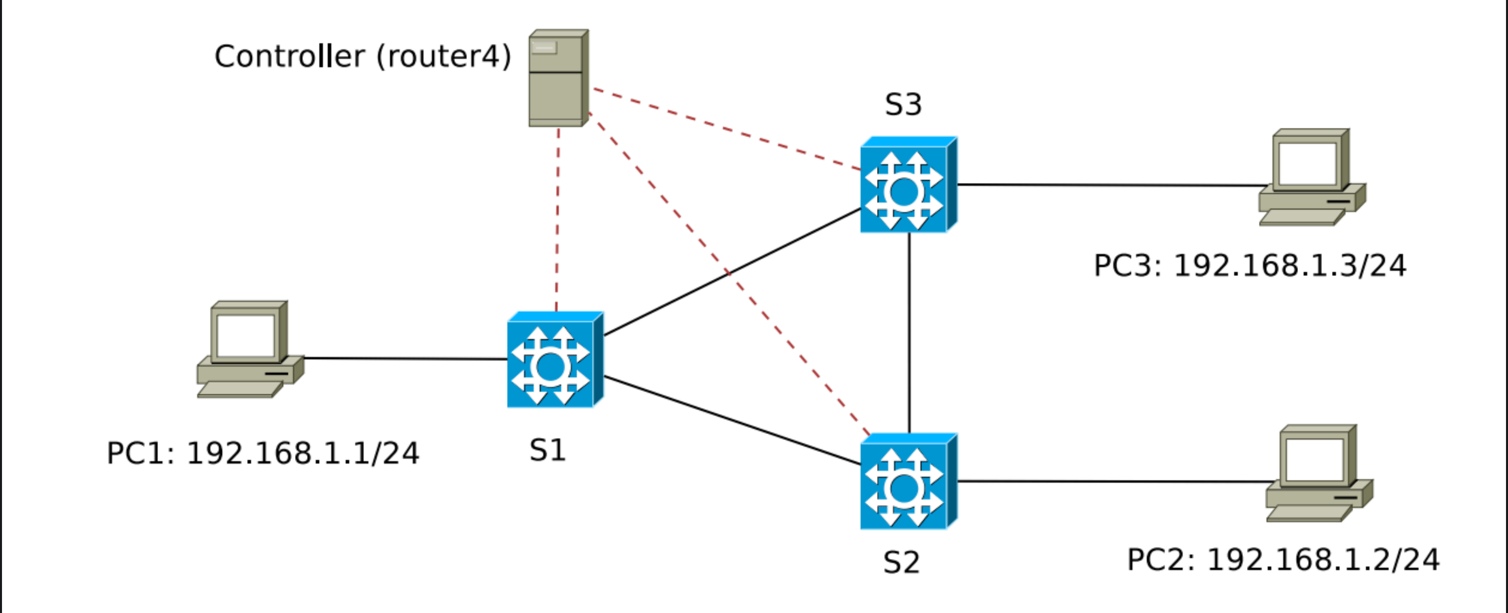

Figure 4.16: Network for deployment

Requirements:

1. If the traffic volume from **PC1 to PC2** is ≥ **512 Kbps**:
   - Perform **load balancing** at the gateway. Traffic flows will alternately be delivered to **PC3** and **PC2**, as PC3 also provides the same service as PC2.
   - The response traffic from **PC3 to PC1** must be modified at the gateway, changing its **source IP address** and **source MAC address** to those of **PC2**.
2. **Optional**: If the traffic volume from **PC1** to either **PC2** or **PC3** is ≥ **768 Kbps**, drop the traffic flow causing the highest load.

需求：

1. 如果从 **PC1 到 PC2** 的流量达到或超过 **512 Kbps**：

    - 在网关执行 **负载均衡**，流量交替转发到 **PC3** 和 **PC2**，因为 PC3 也提供与 PC2 相同的服务。
    - 从 **PC3 到 PC1** 的响应流量必须在网关处修改，将其 **源 IP 地址** 和 **源 MAC 地址** 更改为 **PC2** 的地址。
2. **可选**：如果从 **PC1 到 PC2** 或 **PC3** 的流量达到或超过 **768 Kbps**，丢弃负载最高的流量。

Note: The traffic mentioned refers only to the direction from **PC1 to PC2**. Traffic in the reverse direction (**PC2 to PC1**) or other directions can be ignored.

注意： 以上提到的流量仅指从 **PC1 到 PC2** 的方向。来自 **PC2 到 PC1** 的流量以及其他流量可以忽略。

####

Implementation

Student groups will:

- Develop appropriate SDN applications and P4 programs.
- Deploy these on the provided infrastructure to meet the above requirements.

学生小组需要：

- 开发适当的 SDN 应用程序和 P4 程序。
- 将其部署到提供的基础设施中，以满足上述需求。

4.5.3 Submission

- Submit the source code and a manual detailing how to deploy and run the applications. 提交源代码以及详细说明如何部署和运行应用程序的手册。
- The submission should be compressed into a **zip file**. 提交内容需压缩为 **zip 文件**。

4.5.4 Assessment

The assignment will be assessed based on: 作业的评估基于以下两点：

1. The **final demo** presented in class (at the "Baracke"). 在课堂上进行的 **最终演示**（地点："Baracke"）。
2. The **manual** submitted with the source code. 提交的 **手册**。

4.5.5 Important Dates

- **14.01.2025 23:59**:
  Submission of the source code that measures the traffic volume from **PC1 to PC2**.
  - Refer to the example on Counters available at **https://github.com/mnmteam/p4-sdn/tree/main/counters**.

  提交用于测量从 **PC1 到 PC2** 流量的源代码。
  - 参考 **https://github.com/mnmteam/p4-sdn/tree/main/counters** 中关于计数器的示例。
- **21.01.2025 13:59**:
  Submission of the **final source code** and the **manual**.
  提交 **最终源代码** 和 **手册**。
- **21.01.2025 14:00**:
  Demo presentation