第5章

软件定义网络技术

SDN doesn't allow you to do the impossible. It just allows you to do the possible much more easily.

SDN 不是万能的，它只是让网络变得更容易实现了。

——Scott Shenker

本章目标

学习完本章之后，应当能够：

(1) 理解并给出软件定义网络的意义和技术原理。

(2) 理解并给出 OpenFlow 的基本技术原理。

(3) 理解并给出 SDN 控制器的基本技术原理和列举常用的 SDN 控制器。

(4) 理解 Fibbing 协议的原理和实例。

随着人们对各种应用的需求变化和技术革新，使今天的计算机网络愈渐复杂。而传统的网络架构在面对这些复杂性时，表现出适应性低和更新能力不足等缺点。软件定义网络(Software Defined Network，SDN)的出现为计算机网络带来了极大的可编程性，能够更好地适应应用需求的变化，促进网络技术的革新。SDN 通过将网络设备的控制平面与数据平面分离，从而实现了网络流量的灵活控制，让网络成为一种可灵活调配的资源。本章简要介绍 SDN 的背景和技术原理，以及目前最主流的 SDN 南向接口 OpenFlow 协议和 SDN 控制器等。最后，还通过介绍 Fibbing 协议，展示了一种基于传统分布式网络协议实现软件定义网络的方式。

5.1 SDN 背景

5.1.1 SDN 与传统网络

SDN 是一种新型的网络体系架构，它将网络的数据和控制平面进行分离，控制器利用通信接口对数据平面的设备进行集中式管理，从而实现可编程化控制底层网络硬件，达到对网络资源灵活地按需分配的目的。

如图 5-1 所示，在传统的计算机网络中，控制平面的功能通常是分布式地部署安装在各个网络设备中，如路由器、交换机等。网络设备上运行着各种不同

的分布式网络协议(Protocol),来保证网络数据的传输。因此,当需要在网络中部署新型的网络功能和应用时,所有涉及的网络设备都需要升级,这一限制严重影响了网络的创新发展。SDN 将数据平面和控制平面分离,数据平面可以采用通用的网络设备并提供强大的可编程能力,控制平面上则部署对网络管理和控制的各种逻辑上集中式的应用(Application)。这样,当需要部署新型的网络功能时,只需要在控制节点进行统一的软件升级即可,从而极大地促进了网络技术的创新研究和发展。

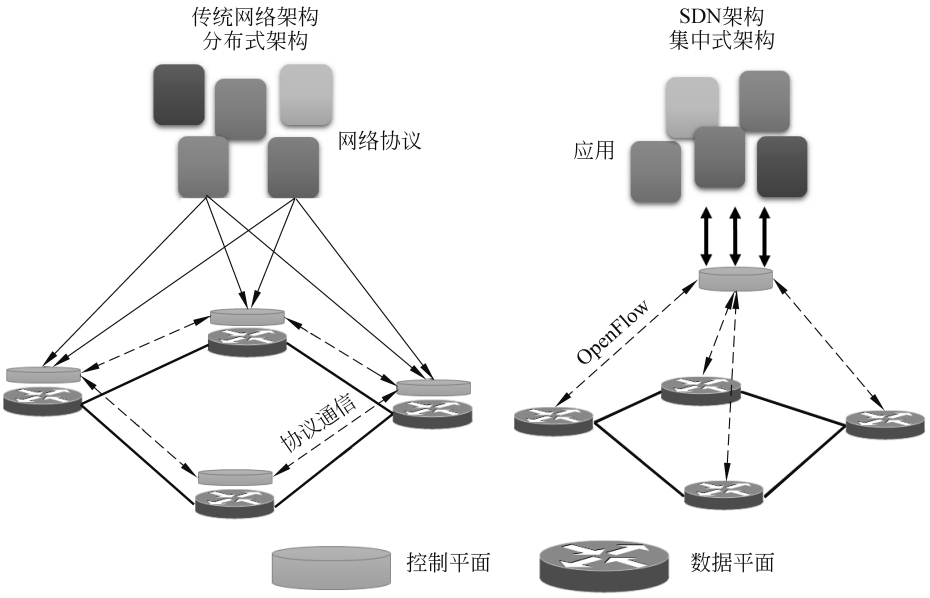

图 5-1 传统网络架构与 SDN 架构

传统网络设备相对比较封闭。网络设备商通常只提供一些接口来方便网络管理员对网络设备进行配置,如命令行接口(Command Line Interface,CLI)等,而往往隐藏了底层的技术设计和实现细节,这也极大地限制了网络管理的灵活性。当面对新的计算机技术和应用所带来新的网络需求,需要在网络中部署新的网络协议和功能时,传统网络架构的弊端就会显现出来。除了带来巨大的网络配置成本外,传统网络架构还具有很长的研发周期和高昂的研发成本,使网络技术升级变得非常困难。由此,转发与控制单元分离(Forwarding and Control Element Separation, ForCES)和路由控制平台(Routing Control Platform,RCP)两种技术应运而生,它们都旨在将数据平面和控制平面分离来提高网络的灵活性。然而,由于商业元素和厂商利益的碰撞等原因,最终这两种协议都没有得到后续的发展。

5.1.2 SDN 发展历史

2007 年,在 ACM SIGCOMM 会议上,斯坦福大学的 Martin Casado 博士和 Nick McKeown 教授等人发表了一篇名为 *Ethane: Taking Control of the Enterprise* 的论文,这篇论文尝试把数据平面和控制平面进行完全解耦合,允许控制器向交换机分发策略,对

网络进行集中式的管理。Ethane 包含的思想就是 SDN 技术的雏形。同年，Martin Casado 博士联合 Nick McKeown 教授和 Scott Shenker 教授等人共同创建了 Nicira 公司，专注于网络虚拟化，并提出了 SDN 的概念。

2008 年，Nick McKeown 教授等人在 ACM SIGCOMM 会议上又发表一篇名为 *OpenFlow: Enabling Innovation in Campus Networks* 的论文，首次提出了 OpenFlow 并应用于校园网中的部署。OpenFlow 是一个用于控制平面和数据平面之间进行数据交互的协议，OpenFlow 保证了控制平面和数据平面完全分离并且各司其职，这使得 SDN 具有了高度的灵活性和强大的可编程性能力。因此，OpenFlow 开始引起了学术界和业界的巨大关注。

2011 年，开放网络基金会（Open Networking Foundation，ONF）在 Nick McKeown 教授等人联合业界一些重量级企业（如 Facebook、Google 等）的推动下成立，其主要致力于推动 SDN 架构、技术的规范和发展工作。同年 4 月，美国印第安纳大学、Internet2 联盟与斯坦福大学 Clean Slate 项目宣布联手发起了网络开发与部署行动计划（Network Development and Deployment Initiative，NDDI），旨在共同创建一个新的网络平台，以革命性的新方式支持全球科学研究的发展，并与欧洲的 GEANT、日本的 JGN-X 和巴西的 RNP 等国际实验平台协作，实现了北美洲、欧洲、亚洲和南美洲的互联。

2013 年 4 月，OpenDaylight 的横空出世引发了业界的一次巨大的轰动，它是由 Cisco、Juniper、Broadcom、IBM 等传统网络设备巨头们主导的开源项目。OpenDaylight 提供开源代码和架构，以推动 SDN 的标准化和 SDN 平台的发展。OpenDaylight 的出现代表着传统网络设备厂商对 SDN 的认可，对于 SDN 技术的发展具有重要意义。同年 8 月，在 ACM SIGCOMM 会议上，Google 公司发表了一篇名为 *B4: Experience with a Globally-Deployed Software Defined WAN* 的论文，首次介绍了 Google 公司如何使用 SDN 技术改造自己的数据中心之间 WAN 流量传输的问题，并在实际部署中证明其链路带宽利用率达到接近 100%的水平。这是 SDN 技术在商业化上成功的案例，它把学术上认可的颠覆性技术成功地应用在了商业项目上，并取得了巨大的商业收益，同时也令 SDN 技术在工业界引起巨大的反响。

2014 年，面向运营商级别的 SDN 控制器 ONOS 正式开源，打破了 OpenDaylight 在商业级别开源控制器项目上的垄断。同年 7 月，协议无关的包处理器 P4 在 ACM SIGCOMM 上诞生。P4 将网络硬件的处理细节交由网络工程师设计，极大地增加了数据平面的可编程性。一些大型公司也逐渐在其网络架构中采用了 SDN，其中 Facebook 公司最具代表性。Facebook 公司还公布了其数据中心的网络架构，并且向外界开源了其交换机操作系统 FBOSS 和 ToR 交换机 Wedge。

2015 年，ONF 发布了开源 SDN 项目社区，SD-WAN 成为第二个成熟的 SDN 应用市场。这一年 NFV 大热，SDN 和 NFV 的融合成为学术界和工业界的热捧。

SDN 从学术界的热捧到在工业上的成熟部署并取得巨大成功，这不仅仅是因为其数控分离和网络可编程等特性带来的优势，还可以认为它是处在网络难以创新、设备封闭这样背景下的一个必然产物，它体现的是整个网络体系架构亟待进行改变的强烈需求。

5.2 SDN 体系结构

5.2.1 SDN 架构

SDN 的分层架构如图 5-2 所示。从上至下分别为应用平面(包含各种网络应用模块,如流量均衡模块和防火墙等)、北向接口(Northbound API,SDN 控制器提供给上层应用的接口,供其调用控制器的功能)、控制平面(主要为各类控制器)、南向接口(Southbound API,SDN 控制器用于管理交换机的接口,最常用的为 OpenFlow 协议)以及数据平面(各类支持 SDN 的网络设备)。

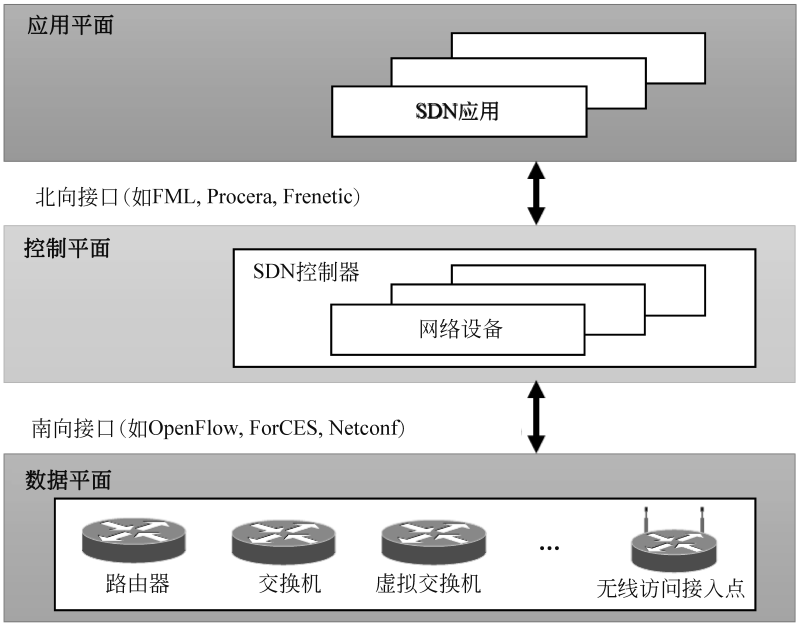

图 5-2 SDN 的分层架构

在 SDN 的体系结构中,数据平面只负责按照控制器下发的流表来转发数据包,而控制器则负责整个网络的管理工作。各平面之间使用不同的接口进行信息交互,共同实现了 SDN 的各项功能。3 个平面的具体内容如下。

(1) 数据平面:由各种网络设备组成,包括路由器、交换机、虚拟交换机和无线访问接入点等。数据平面的主要任务是负责用户数据的转发。它主要根据控制器下发的转发表项,来实现不同逻辑的数据转发。数据平面本身通常不做决策。在数据平面的网络设备中,最核心的部件是转发引擎,实现了对数据包转发和处理。数据平面和控制平面的交互主要依靠南向接口,如使用最广泛的是 OpenFlow 协议。数据平面通过南向接口来处理控制器下发的转发表项,同时数据平面也能向控制器上报网络中的资源和网络状态等信息,由控制器做进一步处理。

(2) 控制平面:整个 SDN 架构里的控制中心,相当于 SDN 中的“大脑”,主要负责网

络内部的路径交换和边界路由的生成，同时负责处理各种网络事件。SDN 控制器(Controller)是控制平面实现的实体，也是 SDN 架构中最核心的部分。在控制平面中，可以允许由多台控制器实例协同组成。在地理位置上，所有控制器的实例可以部署在同一个位置，或者分布在不同的位置。控制器通过南向接口对数据平面中的网络设备进行逻辑上的集中式管理，包括下发转发决策、监测网络状态等功能。同时，控制器通过北向接口向应用平面提供网络业务接口和应用交互，允许用户自定义各种适用于不同网络场景的应用。

(3) 应用平面：主要由一些基于 SDN 的应用组成，这些 SDN 应用是完成用户需求的应用程序，它们通过北向接口与 SDN 控制器进行交互。并且，这些应用是允许用户通过编程的方式，来实现任意的网络行为，并通过控制器向下层网络设备下发对应的转发表项。

5.2.2　数据平面和控制平面分离

传统的网络设备(如路由器)是将数据平面和控制平面紧密耦合的，网络设备之间通过分布式协议进行组网。这种在物理上直接结合的方式实现了数据平面和控制平面的快速交互，保证了网络设备的性能。然而，这种分布式的控制方式也导致了一些问题。①数据平面和控制平面的紧密耦合给网络管理员增加了设备配置的工作量，特别是当网络设备数量庞大时，一个微小的配置错误都可能会引起大面积的网络管理失效。②灵活性很差，当网络功能需要升级时，需要将所有涉及的网络设备逐一全部更新，并且对于设备商部署新的网络功能时所面临的成本和周期也是一项挑战。在面对大规模的云数据中心网络环境时，这些由于数据平面和控制平面的紧密耦合所导致的缺点，体现得尤为明显。

目前，SDN 在实现时的核心思想就是将数据平面和控制平面分离。SDN 中的数据平面和控制平面以转发表为界进行分割。数据平面中的网络设备只保留转发信息，并且拥有高速转发的能力，同时所有的网络决策都交由远端的控制器来完成。在控制平面上，控制器可以获取所有的全局网络信息和状态，并且可以根据这些网络信息和状态，通过南向接口协议提供的数据平面可编程性，做出相应的决策。

SDN 数据平面和控制平面分离的特点是 SDN 的优势之一。这使数据平面能够完全专注于提供高速的数据转发能力，同时也让控制平面能够进行全局网络视野的优化和调度。这种分离的特点能让数据平面和控制平面可以各司其职、互不干扰，同时又可以让开发人员更专注于所开发层面的网络功能，如开发转发性能时可以无须考虑控制逻辑的干扰。另外，从网络设备商的角度来看，SDN 是以转发表为界进行分割的，这依然能隐藏商用设备实现高速转发的细节，因此也使网络设备商能够更容易接受 SDN 的理念。

数据平面和控制平面的分离给 SDN 架构带来很多优势的同时，也带来了不同于传统网络的挑战。SDN 所面临的主要挑战如下。

(1) 控制节点故障：SDN 架构中的“大脑”是控制器，数据平面和控制平面分离后，控制器承担着对整个网络的逻辑控制。因此，一旦控制器出现故障，整个 SDN 将会面临瘫痪的风险。所以，单个控制节点容易成为整个网络性能和可靠性的瓶颈。通过引入多个控制器节点，可以非常有效地解决单点故障问题，并且还能够在多控制器间实现网络负载

均衡。但是,多控制器的管理也同样面临一些挑战,如多控制器资源调度问题等。对多控制器问题的讨论将在 5.4 节中进行具体介绍。

(2) 一致性问题:传统网络中网络状态的一致性是靠分布式协议保证的,而在 SDN 中,这项工作则交由控制器来完成。如何监测到网络节点之间状态不一致的情况,并在网络问题产生前做出相应的措施来解决问题,也是 SDN 架构面临的一个重要挑战。

(3) 高可用性问题:传统网络中,数据平面和控制平面通常同时位于网络设备中,这保证了设备是高可用的,即数据平面向控制平面的请求都能得到实时的回应。然而,在 SDN 架构中,数据平面和控制平面的设备分离后,甚至会不在相同的地理位置,因此控制平面的响应延迟将会直接影响数据平面的可用性。

5.2.3 SDN 可编程性

传统网络的可编程性主要是指开发人员根据设备厂商提供的规范代码,直接控制网络设备硬件,来实现所需要的功能。这种直接控制底层硬件可编程,一般都是针对单台网络设备,十分缺乏灵活性。并且,一般这些编程方式是由网络设备厂商提供,是一些比较底层的基于硬件的编程语言,相当于计算机编程语言中的汇编语言。而网络管理员则希望使用类似 Java 等这样更通用的高级编程语言来更方便地对网络设备进行编程,降低网络配置的成本和时间。

早在 20 世纪 90 年代,美国国防部高级研究计划局(DARPA)在关于未来网络发展方向的研讨会上,就提出了主动网络(Active Networking)的新型网络体系结构,来解决网络的可编程性等问题。主动网络的基本思想是允许用户可以自定义网络节点如何处理接收到的数据包。例如,主动网络的用户通过发送一个定制的功能包,这个功能包中包含了用户所定义的节点将如何处理接下来收到的数据。网络节点收到这个功能包后,将按照用户的定义来处理相对应的数据包。主动网络控制网络节点的方式是通过提供编程接口,并且支持用户指定特定的节点来实现对数据包的自定义行为。因此,主动网络的节点除了能够正常转发数据包外,还能执行用户定制的程序来重新处理收到的数据包。这种可编程的网络体系结构能够在一定程度上加快网络的创新。但是,由于当时能支持这样高可编程性的硬件造价太高、成本太大和市场需求等原因,导致主动网络的体系结构并未被普及,最终该项目被终止。

主动网络等对网络可编程性的尝试,为 SDN 可编程性的网络架构的研究和实践提供了很好的思想借鉴和参考依据。可编程性作为 SDN 架构的一个重要特性,从分层可编程的角度来看,控制平面的可编程性能够更好地实现网络编排和自动化,提供更稳定、高效的网络,并降低网络运营成本;同时,数据平面的可编程性则可以更快地开发新的应用,加速网络服务的更新换代。与主动网络不同的是,SDN 向用户提供了强大的编程接口,从而使用户能对网络进行最大限度的编程控制。SDN 的编程接口主要体现在北向接口和南向接口上。北向接口直接允许开发者操作、控制网络行为,而不需要关心底层硬件的具体设计和实现细节等。并且,通过南向接口,控制器可以兼容不同的网络设备,同时在控制平面实现对上层平面的应用逻辑。

相比于主动网络,SDN 的应用多集中在控制平面上进行开发,通过北向接口实现上

层应用与控制平面交互，通过南向接口用于控制器和数据平面进行交互，这样就解绑了应用与厂商硬件的专门设计。主动网络主要是在数据平面上提供可编程性，这就导致了网络应用和数据平面耦合度较高，因此缺乏灵活性。这是 SDN 可编程优于主动网络的主要原因之一。另外，SDN 更强调的是为网络管理者和开发者提供强大的编程能力，并提供了一整套编程接口，使 SDN 拥有强大的可编程能力；主动网络关注的是为终端用户提供服务，然而终端用户一般并不关心网络编程。这也是 SDN 得以快速发展和普及的重要原因之一。

5.3　数据平面与 OpenFlow

5.3.1　数据平面简介

传统网络的网络交换设备主要是交换机和路由器等。这些网络设备的架构是数据平面和控制平面共同协作，对数据包进行处理。并且，数据平面和控制平面在物理上是紧密耦合在一起的。

在 SDN 架构中，则将数据平面和控制平面完全解耦，交换设备只保留在数据平面，专注于数据包的高速转发。而网络转发决策等控制逻辑，都由控制平面通过南向接口统一分发。这使得数据平面上转发设备的复杂度得以降低，在设计上无须考虑复杂的控制逻辑，从而提升了网络控制和管理的效率。

5.3.2　OpenFlow

2007 年，Nick McKeown 教授等人在 ACM SIGCOMM 会议上首次提出了 OpenFlow，随后主要由 ONF 主导负责 OpenFlow 的标准化、维护和开发等工作。OpenFlow 主要由 OpenFlow 协议和 OpenFlow 交换机两部分组成。其中，OpenFlow 协议是面向控制平面和数据平面之间的通信接口；OpenFlow 交换机定义了支持 OpenFlow 的设备规范，包括基本组件和功能等。

随着 SDN 技术的发展，由 ONF 组织维护的 OpenFlow 也在不断更新和完善中，图 5-3 列举了 OpenFlow 主要版本更迭的时间和重要更新。OpenFlow 各个版本的核心功能和更新情况如下。

(1) OpenFlow v1.0：第一个比较成熟的 OpenFlow 版本。该版本定义了 OpenFlow 交换机的组件和基本功能，同时定义了 OpenFlow 协议；该版本采用单流表结构，支持 12 个匹配字段，简单且易于实现。该版本只支持 IPv4。

(2) OpenFlow v1.1：开始支持多级流表，将流表匹配拆分成多个步骤形成流水线处理，同时新增了组表以支持多播和广播，还增加了 VLAN 和 MPLS 标签的处理。其他变动还包括 Cookies 增加方案、TTL 递减、ECN 动作和定义消息处理等。

(3) OpenFlow v1.2：采用 OXM 编码，用户可以灵活地下发自定义的匹配字段，增加了更多匹配字段，同时节省了对流表空间的消耗；该版本新增允许多台控制器对同一台交换机进行操作来增加可靠性，新增控制器角色转换机制，并且增加了对 IPv6 的支持。

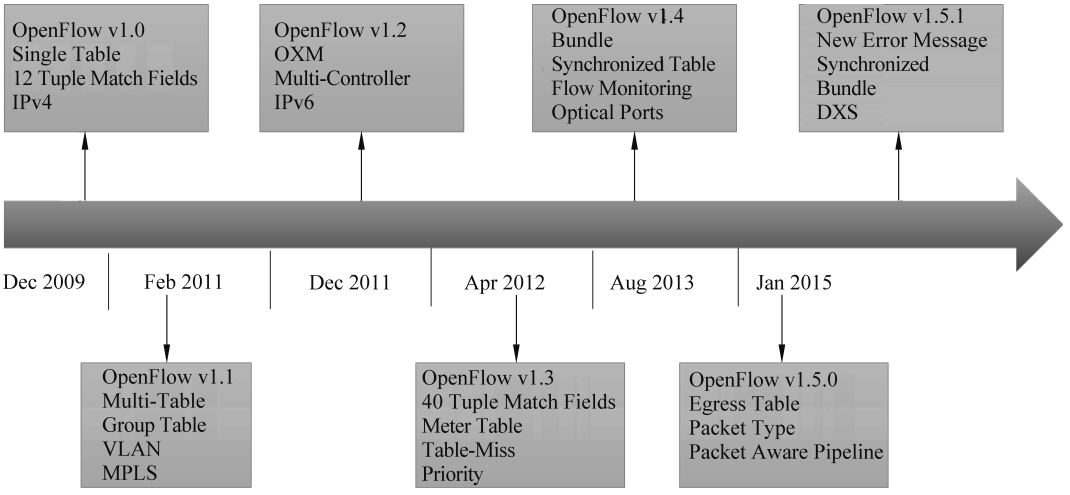

图 5-3 OpenFlow 主要版本更迭的时间和重要更新

(4) OpenFlow v1.3：目前业界使用最广泛的版本。该版本将匹配字段增加到了 40 个，满足网络应用的需求，新增重构能力协商，增加了 Meter 表，控制数据包的传输速率；该版本新增支持 IPv6 扩展包头和 Table-Miss 表项，允许交换机创建辅助连接，提高交换机的处理效率；其他变动包括流表项增加优先级、超时和 Cookie，Packet-in 消息中添加 Cookie 字段等。

(5) OpenFlow v1.4：增加了流表同步机制，确保多个流表共享相同的匹配字段的同时定义不同的动作；该版本还增加了 Bundle 消息，用于保证控制器下发消息的一致性问题，同时增加光端口性能字段，并且新增流表回收机制，当流表满时交换机能自动处理一些流表，允许控制器监控其他控制器对交换机改变流表的行为等。

(6) OpenFlow v1.5.0：引入了 Egress 表，允许在输出端口处理数据包；引入包感知流水线(Packet Aware Pipeline)，允许处理其他类型的数据包；该版本还改进了流表统计信息的处理，新增了对 TCP 标志位匹配等。

(7) OpenFlow v1.5.1：OpenFlow 最新的版本。在该版本的 Asynchronous 中增加了 3 种消息类型(即 Role-Status、Controller-Status 和 Flow-Monitor)，并重新定义了 Error 消息；该版本还在原有的 Bundle 机制中新增了 Schedule Bundle，对 Multipart Statistics 消息重新进行了定义，并引入了 OXS 编码格式。

OpenFlow v1.3 版本架构如图 5-4 所示。其中，安全通道(Secure Channel)、流表(Flow Table)和 OpenFlow 协议(OpenFlow Protocol)是 OpenFlow 架构的核心部分。在 OpenFlow 架构中，控制器通过流表来指导数据平面的流量转发操作；安全通道与控制器(Controller)相连，用于 OpenFlow 交换机与控制器之间进行信息交互通道；OpenFlow 协议是用于信息交互的协议，定义了各种交互消息。

5.3.3 OpenFlow 协议消息

OpenFlow v1.3 主要支持 3 种消息类型：控制器到交换机(Controller-to-Switch)消

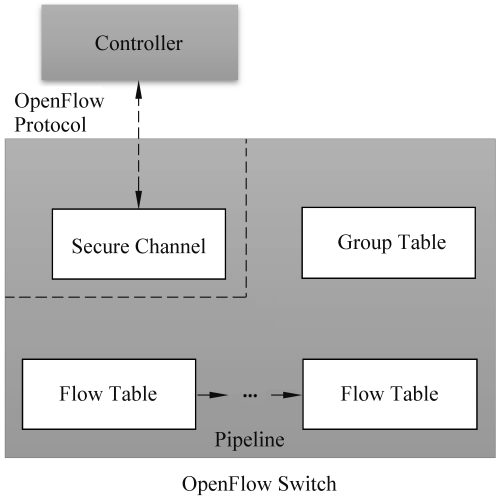

图 5-4　OpenFlow v1.3 版本架构

息、异步(Asynchronous)消息和对称(Symmetric)消息。每种类型的消息又包含多种子消息。其中,Controller-to-Switch 消息是由控制器发起的,用于查询和管理交换机中的状态和信息,这些消息的发起不一定需要交换机应答;Asynchronous 消息主要是由交换机发起,交换机通过此类型的消息将网络事件或交换机状态的变化通知到控制器;Symmetric 消息可由交换机或控制器发起,主要用于建立连接等。

3 种类型的消息所包含的各个子消息如下。

1. Controller-to-Switch 消息

- Features Message:包括 Feature Request 和 Feature Reply。当控制器和交换机建立会话时,控制器向交换机发送 Feature Request 消息来请求交换机的身份及功能;同时,交换机需要通过 Features Reply 消息进行应答。
- Configuration Message:用于控制器设置和查询交换机的参数和配置,交换机需要应答请求,这些配置包括交换机信息配置、流表配置和队列(Queue)配置。
- Modify-State Message:用于控制器管理交换机的流表、组表和端口等信息,这些操作包括增加、删除和修改。
- Multipart Messages:控制器用于收集各种交换机上的统计信息,这些信息包括流统计信息、流表和组表统计信息、端口和队列统计信息等,交换机需要应答相应的请求。
- Packet-Out Message:控制器向交换机发送的数据包,其中包含了指导交换机的动作(Action)信息,空动作列表将会导致数据包丢弃。
- Barrier Message:控制器用于请求交换机关于消息依赖满足情况和下发操作完成情况的信息,通常用于确定消息和命令的执行信息,交换机需要应答请求。
- Role Request Message:当交换机连接了多台控制器并且控制器需要进行角色转

换时，控制器可以发送该消息完成角色的转换，交换机需要应答相应的请求。

- Set Asynchronous Configuration Message：控制器能够选择接受特定 OpenFlow 通道的异步请求消息，通常用于多台控制器的网络环境中。

2. Asynchronous 消息

- Packet-In Message：交换机收到无法处理的数据包时，会发送该消息请求控制器进行处理，允许交换机设置缓存数据包，并且只需要发送数据包部分内容和缓存 ID 给控制器，否则需要发送整个数据包到控制器进行处理。
- Flow Removed Message：当流表由于有效时间截止或是控制器的要求而被删除，交换机将会发送该消息通知控制器。
- Port Status Message：该消息用于通知控制器关于端口配置信息的改变，如增加、修改和删除等。
- Error Message：交换机或控制器发生问题时会触发这一消息，通常用于请求操作失败或匹配错误等情况。

3. Symmetric 消息

- Hello：该消息只由一个 OpenFlow 头部组成，用于控制器和交换机建立连接。
- Echo Request：该消息用于测量延迟和保持连接，可由交换机或者控制器一方发出，接收方必回复 Echo Reply 消息。
- Echo Reply：用于回复 Echo Request 消息。
- Experimenter：用于在 OpenFlow 消息类型空间中为 OpenFlow 交换机提供额外的功能。

OpenFlow 交换机启动后，交换机与控制器之间的消息交互顺序如图 5-5 所示。当一个 SDN 内的 OpenFlow 交换机启动后，交换机将首先使用 TCP 建立“三次握手”的有效连接。

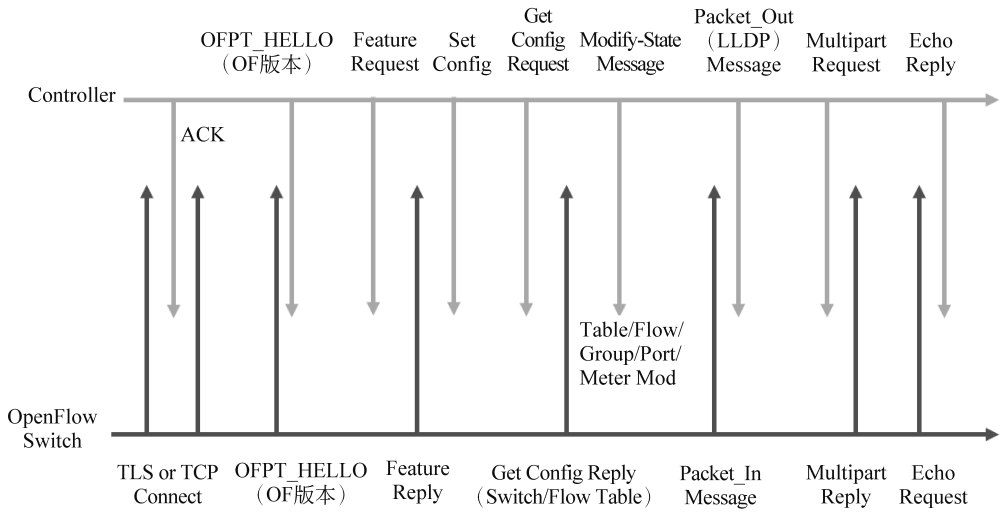

图 5-5　交换机与控制器之间的消息交互顺序图