上海交通大学硕士学位论文

软件定义网络中基于事件机制的网络状况监控与流调度

硕 士 研 究 生：易弢

学　　号：1130339061

导　　师：沈耀 副教授

申 请 学 位：工学硕士

学　　科：计算机科学与技术

所 在 单 位：计算机科学与工程系

答 辩 日 期：2016 年 1 月 18 日

授予学位单位：上海交通大学

Dissertation Submitted to Shanghai Jiao Tong University
for the Degree of Master

NETWORK MONITORING AND FLOW SCHEDULING BASED ON EVENT MECHANISM IN SOFTWARE DEFINED NETWROKS

Candidate：	Yi Tao
Student ID:	1130339061
Supervisor：	Assoc. Prof. Shen Yao
Academic Degree Applied for：	Master of Engineering
Speciality：	Computer Science and Technology
Affiliation：	Department of Computer Science and Engineering
Date of Defence：	Jan 18,2016
Degree-Conferring-Institution：	Shanghai Jiao Tong University

软件定义网络中基于事件机制的网络状况监控与流调度

摘 要

软件定义网络中通过中心化的控制器来控制网络中所有设备的转发的架构简化了网络管理方面应用的开发。OpenFlow 协议作为软件定义网络中控制器与交换机间通信协议的事实标准，已经提供了很多接口以收集交换机上的统计数据，用于网络状况监控。

然而，利用 OpenFlow 提供的收集统计数据的接口进行网络状况监控的应用具有其局限性。OpenFlow 提供的大部分收集统计数据的接口都由控制器发起请求以读取交换机上的统计，缺乏交换机主动向控制器主动上报统计数据的途径。这种方式存在着统计数据收集不及时、网络开销较大等缺陷。

为解决这一问题，本文提出并在控制器与交换机上实现了 OpenEvent 事件框架，作为 OpenFlow 的一个扩展，以允许交换机在一定条件下按照控制器的要求，通过上报事件来主动上报自身的统计数据。我们在这一框架下实现了收集端口统计数据与流统计数据的事件，基于这些事件，我们在控制器上实现了在 OpenFlow 网络中基于事件机制的链路带宽利用状况监测与大象流探测等网络监控方面的应用，以及流调度上的应用。

我们在模拟的 OpenFlow 网络环境下进行了一系列实验来验证 OpenEvent 框架下，基于事件机制的网络状况监控与流调度方面应用的有效性。实验结果表明，基于事件机制的网络状况监控方法在监控的有效性与及时性上优于基于轮询的方法，并显著减少网络开销；基于事件机制的流调度应用能够有效地动态调度流量，提高网络中的带宽利用率。

关键词： 软件定义网络（SDN），OpenFlow，事件机制，网络状况监控，流调度

NETWORK MONITORING AND FLOW SCHEDULING BASED ON EVENT MECHANISM IN SOFTWARE DEFINED NETWORKS

ABSTRACT

Software defined networks (SDN) simplifies development of applications on network management by centralized controlling. As the de facto standard of protocol between controllers and switches, OpenFlow has provided interfaces to collect statistics from switches for applications on network monitoring.

However, the applications on network monitoring using existing interfaces in OpenFlow have their disadvantages. Most of the interfaces for collecting statistics from switches in OpenFlow are pull-based, and there are few approaches for switches to report their statistics to their controllers. So, the existing methods have their flaws such as lack of timeliness of collected statistics and high network overhead.

In order to solve the problems, we propose and implement OpenEvent as an experimenter extension in OpenFlow, which is a framework that enables switches to report their statistics to their controllers through event mechanism. We have implemented events to collect statistics on ports and flows. Based on the events, we develop applications on the controller for network monitoring, such as link utilization monitoring and elephant flow detection, and an application flow scheduling in OpenFlow networks.

Experiments done on emulated OpenFlow network has proved effectiveness of our applications based on event mechanism supported by OpenEvent. The applications for network monitoring based on event mechanism gives a better performance on accuracy and timeliness of statistics, and consume significantly fewer network resources. The application for flow scheduling can dynamically schedule flows in

OpenFlow network and achieves a better utilization of bandwidth in the whole network.

KEY WORDS: software defined networking (SDN), OpenFlow, event mechanism, network monitoring, flow scheduling

目 录

第一章 绪论 ……1
1.1 软件定义网络介绍 ……1
1.2 软件定义网络的研究与应用现状 ……3
1.3 系统总体介绍 ……6
1.4 论文的主要内容与章节安排 ……7
第二章 软件定义网络中的网络监控与流调度的发展现状与存在的问题 ……8
2.1 基于 OpenFlow 协议的网络监控研究现状 ……8
2.2 OpenFlow 框架外的软件定义网络中的网络监控 ……10
2.3 软件定义网络中的流调度 ……11
2.4 本章小结 ……12
第三章 OPENEVENT 事件框架设计 ……13
3.1 OpenEvent 框架架构设计 ……13
3.2 OpenEvent 框架运行流程设计 ……15
3.3 OpenEvent 中的事件类型实例 ……16
3.4 本章小结 ……17
第四章 OPENEVENT 事件框架与框架中事件实例的实现 ……18
4.1 通信协议 ……18
4.1.1 事件请求 ……19
4.1.2 事件请求回复 ……20
4.1.3 事件报告 ……21
4.1.4 OpenEvent 中两类事件的相关消息格式 ……21
4.2 控制器端的实现 ……25
4.3 交换机端的实现 ……26
4.4 本章小结 ……29
第五章 OPENEVENT 在网络状况监控与流调度上的应用 ……30
5.1 带宽利用状况监测 ……30
5.2 大象流探测 ……31
5.3 Fat tree 网络中的流调度 ……32

5.4 本章小结 ……36
第六章 实验设计、结果与分析 ……38
6.1 实验环境介绍 ……38
6.2 大象流探测的相关实验设计、结果与分析 ……39
6.3 Fat tree 网络中流调度的相关实验设计、结果与分析 ……42
6.4 带宽利用状况监测的相关实验设计、结果与分析 ……44
6.5 本章小结 ……46
第七章 结束语 ……47
7.1 主要工作与创新点 ……47
7.2 后续研究 ……48
7.3 本章小结 ……49
参 考 文 献 ……50
致 谢 ……55
攻读硕士学位期间已发表或录用的论文 ……56

图 录

图 1-1 软件定义网络的架构 ……2
图 1-2 OpenFlow 消息的格式 ……3
图 3-1 OpenEvent 的架构示意图 ……14
图 4-1 OpenFlow 中自定义类型消息的格式 ……18
图 4-2 OpenEvent 中事件请求的消息格式 ……19
图 4-3 OpenEvent 中事件请求回复消息的格式 ……20
图 4-4 OpenEvent 中事件报告的格式 ……21
图 4-5 端口统计事件的事件请求描述格式 ……22
图 4-6 端口统计事件的事件报告格式 ……22
图 4-7 不同版本的 OpenFlow 协议下的 OpenEvent 中对流统计事件的描述格式 23
图 4-8 流统计事件的事件报告头部格式 ……24
图 4-9 流统计事件的事件报告中单个流表项的统计数据描述格式 ……24
图 4-10 ofproto_event_run 函数检查事件是否触发并上报的流程图 ……28
图 5-1 流调度应用架构 ……34
图 6-1 不同流量模式下大象流发现时已传输字节数的累积分布 ……40
图 6-2 两种方法中大象流探测中的网络开销 ……41
图 6-3 各种流量模式下不同路由与调度方法达成的网络总带宽 ……43
图 6-4 两种方法测得的带宽利用状况 ……44
图 6-5 带宽状况监控的两种方法中传输的数据包数与数据总量 ……45

表 录

表 4-1 OpenEvent 中几种事件状态值的定义 ……20

第一章 绪论

1.1 软件定义网络介绍

随着云计算的蓬勃发展，大数据在越来越多的领域得到广泛应用，用于承载其产生的海量数据和大规模计算任务的数据中心的规模也在不断扩大。随之而来的是，数据中心内连接各节点，以及数据中心之间互相连接的网络的规模也在不断增大，相互连接的关系也趋于复杂。与传统互联网络相比，数据中心中的网络具有如下特点：带宽需求高，一般数据中心采用 1Gbps 以太网连接服务器与接入层，10Gbps 以太网连接接入层与核心交换机，在如此高带宽的连接下，仍然会出现 60~70%带宽占用率持续 10~100 秒的现象[1,2]；流量分布复杂，一次计算任务通常会在多个服务器之间产生大量流量；允许集中控制，数据中心通常对其中所有的网络设备和网络设备之间的互相连接具有完全的控制权。传统网络的分布式配置、难于扩展的缺陷使得它在数据中心的网络中的适用程度降低，为适应数据中心的网络的需求，需要全新的技术对传统的互联网络做出扩展和改变。

软件定义网络（Software Defined Network, SDN）起源于斯坦福大学在 2006 年开始的一个 clean state 项目，这个项目的目的在于改变已经不合时宜且难以扩展的传统互联网络。2008 年，斯坦福大学的 Nick McKeown 等人发表 OpenFlow 白皮书，提出了 OpenFlow 这一具有中心化控制结构和可编程性的网络框架[3]。2009 年，软件定义网络被 MIT Technology Review 评为十大前沿技术[4]。

软件定义网络的核心特性在于以下两点：第一，控制平面（负责控制数据转发的规则）与数据平面（负责具体数据包的转发）分开；第二，逻辑上的集中式控制[5]。软件定义网络的架构总体上分为三层：应用层、控制层、数据转发层，如图 1-1 所示。应用层与控制层通过 SDN 北向接口（SDN Northbound Interface, NBI）与控制层交互。SDN 应用程序运行在应用层，通过北向接口向控制器告知自身对网络资源的需求以及对网络行为的需要，同时通过北向接口从控制器中读取网络的当前状态与各种统计数据。SDN 控制器运行在控制层，负责将 SDN 应用给出的对网络的需求翻译并转达至网络设备上，并向上层应用提供抽象化的网络视图，包括网络状态与统计数据等。SDN 控制器在逻辑上是一个中心化的实体，与网络设备之间通过控制平面-数据平面接口（Control-Data-Plane Interface，CDPI）通信。SDN 数据通路是对网络设备的抽象，对应于传统网络中的交换机、路由器

等，运行在数据转发层，负责实际的网络数据处理与转发工作。

OpenFlow 自从提出以来，得到了工业界与学术界的大力支持和广泛应用。2009 年 12 月，OpenFlow 的第一个版本的正式标准 OpenFlow Switch Specification version 1.0 发表[6]；2011 年，开放网络基金会（Open Networking Foundation，ONF）成立，接管了 OpenFlow 协议和标准的制订工作。这一组织包括了思科、华为等设备制造商，沃达丰、中国电信、中国移动等网络运营商，VMware、微软等软件与云服务提供商，以及 Google、百度、腾讯等互联网企业。从此，OpenFlow 成为了 SDN 的事实标准，目前 SDN 的控制器与网络设备之间的通信接口绝大多数采用 OpenFlow 协议。OpenFlow 协议现有的最新版本是 1.5 版。

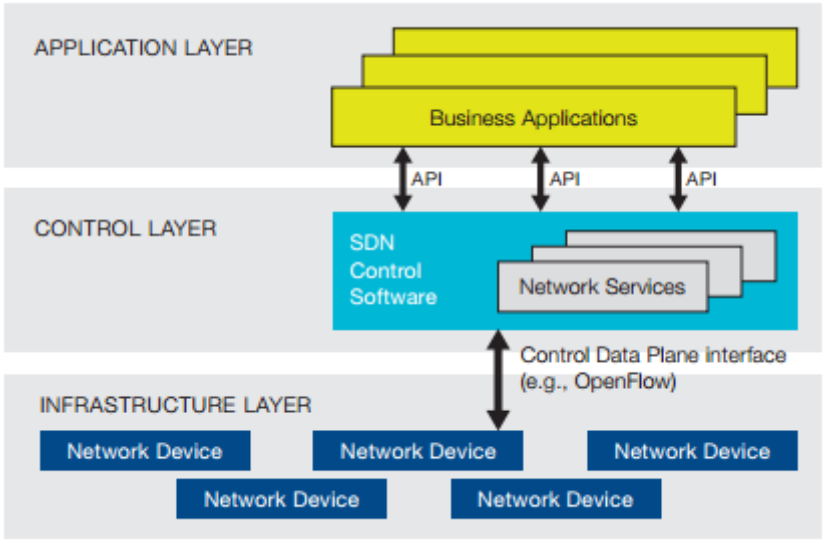


图 1-1 软件定义网络的架构[7]

Fig.1-1 Architecture of software-defined network (SDN)

OpenFlow 框架体现了“控制与转发分开”的原则。 OpenFlow 网络中的网络设备称为 OpenFlow 交换机（以下均称为交换机），对应于 SDN 架构中的数据转发层。OpenFlow 网络中，每一台交换机有一个唯一的 DPID，供控制器区别不同的交换机。交换机对数据的转发规则由流表决定。一个交换机中有一个或多个流表，每个流表中有若干条具有优先级的流表项。每条流表项包含一个匹配规则，和一组动作或指令。一个数据包到达交换机时，交换机将会执行与该数据包匹配的最高优先级的流表项所指定的动作或指令。所有交换机的流表都由控制器控制，控制器可以通过对流表中的流表项的操作来配置交换机的转发规则。交换机需要为每条流表项维护一组计数器，记录该流表项的创建时间、空闲时间、匹配的数据包数、匹配的数据包的总字节数等统计数据。控制器可以通过查询这些统计数

据，掌握网络的全局信息。

OpenFlow 协议运行在应用层，控制器需要与其控制的交换机建立 TCP 连接以发送和接收 OpenFlow 消息。OpenFlow 控制器监听的 TCP 端口号已被 IANA 确定为 6653[8]，在该标准确定前，OpenFlow 控制器大多监听 6633 端口。OpenFlow 协议中的消息如图 1-2 所示。OpenFlow 版本表示这条消息所使用的 OpenFlow 协议的版本，1 表示 OpenFlow1.0，2 表示 OpenFlow1.1，依此类推。长度表示该 OpenFlow 消息的总长度。类型表示该 OpenFlow 消息所属的类型。事务 ID 用于标识控制器发出，需要交换机给出回复的消息。交换机接到这类需要回复的消息时，应在回复的消息中使用与原始消息相同的事务 ID 以表示该消息是对原始消息的回复。其他情况下，事务 ID 没有实际作用，可以任意指定。OpenFlow 消息可以分为如下三大类：控制器向交换机发送的单向消息：控制器发送消息，交换机无需回复；对称消息：控制器向交换机发送消息，交换机需要对该消息做出回复；异步消息：交换机主动向控制器发出消息。

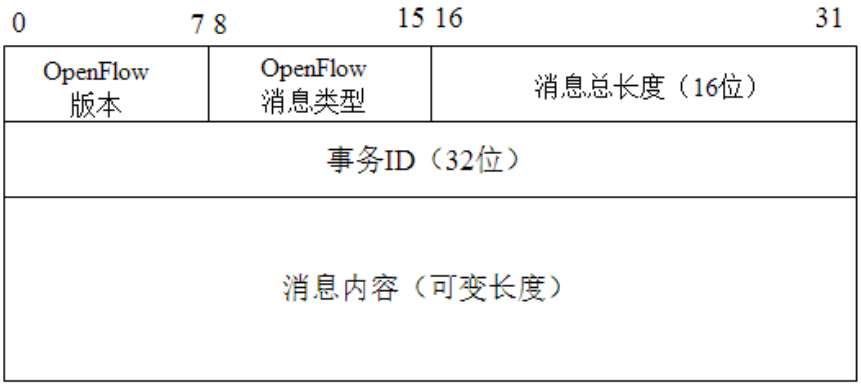


图 1-2 OpenFlow 消息的格式
Fig.1-2 Format of messages in OpenFlow

目前，基于 OpenFlow 的软件定义网络已在研究和应用上得到了快速的发展，OpenFlow 控制器、OpenFlow 软硬件交换机等设备已经大量出现。各种在传统网络中无法实现或难以实现的网络应用与网络管理功能在 SDN 中得到了实现。下一节将更加具体地介绍 SDN 目前在研究与实际应用上的进展。

1.2 软件定义网络的研究与应用现状

目前，由于 OpenFlow 已经成为 SDN 中数据平面与控制平面之间通信协议的事实标准，绝大多数对于 SDN 的研究和应用都以 OpenFlow 为基础。

OpenFlow 交换机与 OpenFlow 控制器是组建 OpenFlow 网络的必需基础设施，本文将首先介绍目前已经出现的 OpenFlow 交换机与 OpenFlow 控制器的实现。

OpenFlow 交换机可以分为软件交换机与硬件交换机两大类。软件交换机可以运行在其他的操作系统上，通过操作系统提供的接口对网络设备进行操作，完成解析 OpenFlow 消息、转发数据包等工作。硬件交换机使用专门的操作系统，转发数据包的功能由专门的硬件完成。

软件交换机中，影响最大、应用最广泛的是 Open vSwitch（OVS）。OVS 由 OpenFlow 提出者 McKeown 创办的 Nicira 公司开发[9]，目前已经开放源代码。OVS 目前可以在 Linux 上运行，支持 OpenFlow 协议的 1.0 至 1.4 版本，并且实现了很多 Nicira 公司创造的扩展。OVS 包括一系列用户态程序与一个 Linux 内核模块，用户态程序用于维护交换机状态、解析 OpenFlow 消息，内核模块部分用于调用操作系统的网络转发功能，完成数据包的转发。Linux3.3 版及之后的内核已经把 OVS 的内核模块包含在内。此外，BigSwitch 公司开发的 Indigo Virtual Switch[10]（IVS）也可以支持 OpenFlow，底层接口兼容 OVS 内核模块与某些硬件交换机。OFSoftswitch 运行在 Linux 的用户态，是纯软件交换机，支持 OpenFlow1.0 至 1.3[11]。相比之下，OFSoftswitch 对于 OpenFlow 提出的功能支持比较全面，例如，OVS 不支持 OpenFlow1.3 的 meter 功能，而 OFSoftswitch 全面支持。然而，OFSoftswitch 在数据转发上的性能远低于 OVS 内核态交换机，因为 OFSoftswitch 中，数据包的处理总是需要从内核态转移到用户态，造成大量开销。

OpenFlow 硬件交换机可以通过 NetFPGA 实现[12]，但实际应用的硬件交换机多由设备厂商生产。目前的 OpenFlow 交换机大多支持 OpenFlow1.0，少数厂商实现了 OpenFlow1.3 的支持。思科的 Catalyst 4500E 系列交换机[13]等都支持 OpenFlow。Pica8 推出的 P-3290 等交换机是最早支持 OpenFlow1.4 的硬件交换机[14]。有些硬件交换机的操作系统中，也使用 OVS 的用户态部分作为上层来进行 OpenFlow 协议解析等工作。

同样，控制器可以分为软件控制器与硬件控制器。由于控制器需要向 SDN 应用提供接口，目前流行的 OpenFlow 控制器多为运行在 Linux 上的软件控制器。较为流行的开源 OpenFlow 控制器有以下一些：NOX[15]、POX[16]、Beacon[17]、floodlight[18]、Ryu[19]等。这些控制器的架构大体相同，都提供一个基本的管理 OpenFlow 连接、解码 OpenFlow 消息的框架，并对其他应用开放处理 OpenFlow 消息的接口。IBM、NEC 等厂商也实现了其硬件控制器[20,21]。HyperFlow 提供了一种分布式运行，但对外仍然表现为单中心控制的分布式 OpenFlow 控制器的实

现[22]。在这些控制器的基础上，FlowVisor[23]、FlowN[24]等实现了通过 OpenFlow 进行网络资源虚拟化的平台；Onix[25]、OMNI[26]等分布式网络功能控制平台也在 OpenFlow 的控制器上得到实现，并提供更上层的接口。

OpenFlow 交换机功能复杂，实现上需要关注的细节繁多，因此，OFTest[27]、OFLOPS[28]等用于测试 OpenFlow 交换机的测试平台出现了。OFTest 主要验证交换机对 OpenFlow 中各种功能实现的正确性，目前支持 OpenFlow1.0 和 OpenFlow1.1。OFLOPS 除检测功能实现的正确性外，还实现了检测数据转发性能的功能。在硬件实现的 OpenFlow 平台上进行研究需要大量网络设备，为方便研究与验证，Mininet 通过在 Linux 上建立虚拟网卡，并配置 OVS 的方式模拟大规模的 OpenFlow 网络[29]。这大大地减少了在 OpenFlow 平台上进行研究的时间、金钱、人力成本。为克服 Mininet 只能运行在单台机器上的问题，Maxinet 在 Mininet 的基础上提供了在多台机器上搭建分布式虚拟 OpenFlow 网络的功能[30]；为减少 Mininet 模拟与实际情况的偏差，Mininet-HiFi 提出了基于 container 的实现方式[31]，利用 container 增强网络中各个节点的隔离程度与可控制性。Estinet[32]通过对 Linux 内核中 TCP/IP 的 socket 实现进行修改，实现了 OpenFlow 模拟环境，比 Mininet 具有更强的可配置性。但由于其并未开放源代码，获取需要付费购买，其在学术上的应用范围小于 Mininet。

SDN 与 OpenFlow 已经因为控制平面与数据平面分开、中心化控制的特点，在数据中心内的网络，以及其他的大规模互联网络中得到了大量的研究成果与实际应用。Google 在连接其各个数据中心的骨干网络 B4 中已经应用了 OpenFlow[33]。微软在其数据中心之间的网络上也采用了 OpenFlow 交换机[34]。基于 OpenFlow 的网络管理应用，尤其是在传统互联网络中无法实现或难以实现的应用已经有了许多积累。在支持 OSPF 的路由器连接的网络中进行基于 OpenFlow 的中心化路由已经实现[35]，通过在控制器一端预先配置部分路由，OpenFlow 交换机就可以通过收到的 OSPF 消息计算转发路径，并向其他路由器发布 OSPF 消息以告知该路径。这大大加快了路由算法收敛的速度。OpenFlow 在负载均衡上也得到了很好的应用[36,37]。负载均衡应用通过控制器控制数据转发的规则，让交换机按照应用的需求把需要处理的数据均衡分配到各台主机上。OpenFlow 在网络状况监控与流调度方面的研究与应用的成果也很丰富，但也普遍存在一定的局限性，本文将在第二章进行详细的介绍。

虽然目前绝大多数 SDN 的研究与应用都以 OpenFlow 为基础，也存在脱离 OpenFlow 平台的 SDN 的研究和应用。思科提出 OpFlex 作为控制器与网络设备之

间的通信协议[38]。OpFlex 提供比 OpenFlow 更加抽象的接口，它通过给出抽象的网络策略来管理网络资源。

1.3 系统总体介绍

OpenFlow 提供了大量从交换机读取统计数据的接口，控制器可以通过这些接口读取多种类型的统计数据：端口统计数据，包括一个端口发送的数据包数量、发送字节数、接收数据包数量、接收字节数、传输错误次数等；单独流表项统计数据，包括符合某一特定条件的所有流表项各自的匹配数据包数、匹配字节数、空闲时间、创建时间等；队列统计数据，包括通过该队列输出的数据包数量和字节数等，以及其他各种类型的统计数据。但这些接口都是通过控制器下发请求，交换机回复结果的方式获取统计数据的，缺乏交换机在一定条件下主动向控制器上报其统计数据的接口。OpenFlow 也提供了异步消息供交换机上报事件，但是能通过异步消息上报的事件类型极为有限，目前仅包括端口状态变化（连接或断开连接）、流表项被删除、OpenFlow 错误等几类，不能满足网络监控与流调度的应用中获取统计数据的需求。

因此，本文提出了 OpenEvent 这一基于 OpenFlow 的事件机制框架，允许控制器要求交换机在给定的统计数据达到或超过事先设定的阈值时，触发一个事件，并向控制器上报。交换机上报事件时，应上报控制器所需要的统计数据。通过这一框架，控制器可以通过配置交换机上的事件，要求其主动上报统计数据。我们基于 OpenEvent 这一框架定义了两类事件，允许控制器要求交换机主动上报端口的统计数据与流表项的统计数据。在此基础上，我们实现了在 OpenFlow 网络中，基于事件机制的网络状态监控与流调度的应用。

OpenEvent 实现为对 OpenFlow 的扩展，其实现包括通信协议、控制器端、交换机端几个部分。从设计上，OpenEvent 不限制事件的类型，交换机上的任何状态变化和统计数据变化都可以被设定为事件。OpenEvent 中，控制器可以要求交换机安装新的事件、变更已安装事件的配置或删除已安装的事件，交换机应对这些请求给出回应。当交换机上的事件被触发时，交换机通过异步消息上报事件的发生与相关的统计数据。OpenEvent 中，关于事件的所有消息均为 OpenFlow 自定义类型消息，通信协议将对与事件相关的消息格式与含义进行介绍。控制器端在 Ryu OpenFlow 控制器框架下实现[19]，包括了对 OpenEvent 中使用的消息的解析，以及开放给其他应用的接口。交换机端包括对通信协议的解析，以及各类事件的监测与上报。

基于 OpenEvent 框架，我们定义并实现了监控交换机端口统计数据的事件与监控流表项统计数据的事件。在这两类事件的基础之上，我们开发了监测链路带宽利用状况、探测大象流等网络状态监控方面的控制器端应用，以及在网络状态监控基础上的流调度应用。

1.4 论文的主要内容与章节安排

在本章中，我们介绍了软件定义网络与 OpenFlow 的基本概念与它们在研究与应用上的现状，以及 OpenEvent 这一用于网络状况监控与流调度的事件框架的总体概况。在论文的剩余部分，我们将介绍 SDN 中目前的网络监控与流调度的研究现状以及存在的问题，OpenEvent 框架的设计、实现，基于该框架的网络监控与流调度的实现，验证该框架效果的实验的设计、结果与分析。

本文的剩余章节将按照以下顺序安排：第二章将介绍现有的网络状况监控与流调度的发展，以及现有的网络状况监控与流调度的应用中存在的问题；第三章将从架构、运行流程方面介绍 OpenEvent 框架，以及该框架内两类事件的设计；第四章将从通信协议、控制器端、交换机端等三个方面介绍 OpenEvent 的实现细节；第五章将介绍基于 OpenEvent 的网络状况监控与流调度的应用，包括链路带宽利用状况监测、大象流探测以及 Fat tree 网络上的流调度；第六章将展示以上带宽利用状况监测、大象流探测、流调度等几个应用在 Mininet 模拟网络环境下的实验，包括实验的环境、设计、结果以及对结果的分析；第七章总结全文，做出结论，并指出在本文基础上可能的后续研究方向。

第二章 软件定义网络中的网络监控与流调度的发展现状与存在的问题

2.1 基于 OpenFlow 协议的网络监控研究现状

OpenFlow 协议中规定了用于读取交换机状态和各种统计数据的消息类型，控制器可以通过向交换机发送请求，让交换机上报自身的统计数据。大多数基于 OpenFlow 协议的网络监控上的应用都采用了这些消息来从交换机上获取各类统计数据。此外，用于交换机通知控制器流表项已删除的 flow removed 消息也可以携带该流表项的存在时间，以及该流表项在存在期间匹配的数据包数和字节数等统计数据，因此 flow removed 消息也可以用于网络监控。

OpenTM[39]提出了一种在 OpenFlow 网络中估计各主机之间的流量矩阵的方法，以及其在 NOX 控制器[15]上的实现。该方法利用了 OpenFlow 交换机能够维护流表项的统计数据的功能，对于某一对主机在路径上的交换机上安装匹配这一对主机的地址的流表项，定时从交换机上读取各个活动的流表项的匹配数据包数与字节数，以流表项的统计数据作为该对主机之间流量的估计值。据称，该方法能在经过 10 次左右的测量后收敛。但是，周期性地对交换机发送请求的方法带来的网络开销较大。

文献[40]利用 OpenFlow 交换机可以将匹配失败的数据包通过 packet in 消息转发给控制器的特性，提出了一种利用 OpenFlow 检测网络延迟的方法。该方法通过控制器向起点的 OpenFlow 交换机发送一个 packet out 消息，令其通过被测链路发送一个数据包，目标交换机收到此数据包后向控制器发送 packet in 消息，控制器即可根据发送 packet out 消息与收到 packet in 消息的时间差，计算出两交换机之间的延迟。这种方法虽然克服了传统网络中利用 ICMP 中的 ping 命令测量延迟的难以集中收集数据等不足，但本身也存在局限性：需要测量的链路的两端必须都是连接在同一控制器上的 OpenFlow 交换机。

为了减少周期性轮询所需要的大量请求带来的大量网络开销以及控制器端的负担，Payless[41]提出了以可变时间间隔向交换机发送请求的方法以减少开销。当一个交换机上的所有流表项在一次轮询中，单位时间内匹配数据包的数量与字节数变化量低于一定范围时，下次请求的时间间隔变长；超过该范围，下次请求的时间间隔变短。这样，当网络负载稳定时，控制器向交换机请求统计数据的时间

间隔较长，减少了在这种情况下的网络开销与控制器负担。用这种方式测量，获得的统计数据和减少网络开销的效果随着时间间隔的变化范围而变化，时间间隔的下限越短，测量的精确度越高，但网络开销也越大。

OpenNetMon[42]在 POX 控制器[16]中集成了测量 OpenFlow 网络中链路带宽使用情况、链路延迟以及丢包率等功能。对于链路带宽使用情况，OpenNetMon 也采取通过发送请求，从交换机上读取从特定端口输出的流表项的单独的统计数据，并在控制器端计算总和的方法测量；对于延迟，OpenNetMon 使用了文献[40]中的方法；对于丢包率，OpenNetMon 采取了测量链路两端的交换机对匹配项相同的流表项的匹配数据包数量差值的方法。OpenNetMon 也提出了类似的用可变时间间隔轮询的以减少网络开销的方法。

为了在一个交换机中为数众多的流表项中找出数量很少，但数据量占绝大多数的“大象流”并避免一次性读取大量流表项的单独统计信息，文献[43]提出利用聚合流统计消息，将交换机中的流表项根据匹配范围划分成不同区域，通过发送聚合流统计请求以获取一定范围内的所有流表项的匹配数据包的数量与字节数总和，并不断缩小包含大象流的流表项范围，直到该范围足够小，再读取该范围内的流表项的单独的统计消息。相比直接读取全部流表项的单独的统计数据，这种方法减少了大象流探测在控制路径上传输的数据量与控制器的处理负担，但需要发送更多次的请求才能完成探测。

FlowSense[44]提出了利用交换机匹配失败时向控制器发送的 packet in 消息以标志流的开始，以交换机在流表项过期删除时发送的 flow removed 标志流的结束，利用 flow removed 消息中携带的流表项的统计数据来获取该流在存活期间匹配的数据包数量和字节数。虽然其宣称这种方法不会带来额外的网络开销，但通过该方法获取的统计数据存在较大的问题：通常一个流表项在没有数据包匹配时不会立即过期删除，而是会继续存在数秒到几十秒（由控制器决定），将这一段没有数据包匹配的时间计入流表项的存活时间会带来误差；更重要的是，使用这种方法，只有一个流表项被删除之后才能获取其统计数据，也就是说使用该方法无法获取存活中的流的统计数据，对后续的流调度、负载均衡等任务的意义不大。

现有的基于 OpenFlow 协议的网络监控方法受限于 OpenFlow 协议的局限，要获得较为精确的统计，必须以较高的频率让控制器去读取交换机中各流表项各自的统计数据，在控制路径（控制器与交换机连接的路径）上带来了较大的网络开销，也给控制器增加了负担。虽然 OpenFlow1.5 版在扩展 EXT-335 中提出了新的指令[45]，可以要求流表项在统计数据达到一定阈值时上报一个消息，但目前还没

有控制器与交换机实现这个扩展。

2.2 OpenFlow 框架外的软件定义网络中的网络监控

目前，在软件定义网络的研究与应用中，也出现了一些不完全在 OpenFlow 框架下运行的网络监控方法。有些方法在 OpenFlow 协议之外自定义一些类似于 OpenFlow，同样能实现中心化控制的协议来使得控制器从交换机上获取所需要的状态信息与统计数据；也有一些方法借用交换机上已经存在的功能，与 OpenFlow 控制器进行集成，以达到在基于 OpenFlow 协议的软件定义网络中实现网络监控的效果。

OpenSketch[46]定义了一个类似 OpenFlow 的网络流量测量的架构，提出了在交换机上运行的“测量平面”，以实现对网络流量测量的集中控制与统计数据的集中获取。OpenSketch 对网络流量的测量以 sketch 这一数据结构为基础，每个 sketch 对满足一定条件的数据包（例如，目的端口为 80 的 TCP 包）根据一定标准分类，并对数据包按照分类分别计数。Sketch 可以由控制器来指定过滤标准与分类标准，并提供接口给控制器以查询计数器内的数据。通过对 sketch 的操作，控制器可以完成中心化地控制交换机上的统计数据收集工作，并获取所需统计数据。但 OpenSketch 需要控制器端与交换机端实现一组完整的异于 OpenFlow 的通信协议，实现代价较大。

DevoFlow[47]提出了一个将 OpenFlow 中由控制器集中控制的部分功能下放回到交换机的框架。对于网络监控，DevoFlow 提出了采样与流触发两种方法以补充和取代 OpenFlow 中全部由控制器发送请求，交换机回复的收集统计数据的方式。采样即从交换机的每一个端口上以一定概率抽取一部分数据包（或其开头若干字节），发送至控制器进行分析。流触发即让交换机在某一流表项在满足一定条件时，向控制器发送异步消息上报。这些做法有效地减少了收集流量大的流的统计数据所耗费的网络开销。然而，使用采样的方法，统计数据的准确性受到采样率的影响；流触发的方法需要额外的触发机制。

PLANCK[48]利用了目前商用交换机广泛支持的端口镜像功能。该功能可以把从交换机的某一端口发出的数据包全部转发到某个指定地址。控制器通过配置端口镜像，可以收集指定的交换机从指定的端口发出的所有数据包，从而获取任何所需要的统计数据。该方法能够获取所有的数据包以收集和计算统计数据，因此用这一方法获取的统计数据相当准确；然而通过端口镜像转发所有的数据包这一操作，对交换机、网络连接和控制器都是非常大的负担，可行性并不高。

OpenSample[49]将 sFlow[50]的采样功能集成到 OpenFlow 控制器中，让各个交换机将采样得到的数据包发送到控制器上，然后由控制器来分析被采样的包。OpenSample 除了直接根据收到属于各个流的包的数量来估计各个流的传输速率外，还利用被采样的 TCP 包头部中的 TCP 序列号来计算其所属 TCP 连接的传输速率。直接估计的方法需要较高的采样率才能获取较为准确的结果，但通过 TCP 序列号来估计可以大大降低所需要的采样率，这也是 OpenSample 的重点之一。但是，通过 TCP 序列号来估计不能用于估计非 TCP 流的数据传输速率，也不能用于估计按照除 TCP 连接以外的其他标准划分的流的速率。

OpenFlow 框架之外的这些方法虽然突破了 OpenFlow 协议的局限，但仍然存在各自的不足。另外，由于它们的实现部分脱离了 OpenFlow，甚至完全脱离了 OpenFlow 的框架，实现的难度高于基于 OpenFlow 的这些方法。

2.3 软件定义网络中的流调度

由于 OpenFlow 协议中，交换机的转发规则直接由控制器指定，流调度功能的实现远比传统网络中简单。流调度也是 SDN 中的重要应用之一，对此人们已经做出了深入而细致的研究。

对于数据中心中的动态流调度，通常数据中心中，某些流数量占所有流的 10%，但传输的数据量却占了所有数据量的 90%，一般称这些流为“大象流”。Hedera[51]提出了只对大象流进行调度，而对于数量占大多数，传输数据量很少的“老鼠流”依然采用等价多路径（Equal Cost Multipath, ECMP）的方式转发。对于这些大象流，Hedera 总是假设它们会尽可能地占满网络的带宽资源，根据这个假设，Hedera 通过计数每台主机所发送和接收的大象流的数量来估计每个大象流所需要的网络带宽，并通过一定的算法将这些大象流安排到网络中的各条路径上，通过发送 flow modification 消息让交换机重新安排转发这些大象流的端口以实现这样的安排。对于安排大象流的转发路径，Hedera 提出了全局最先适配与模拟退火两种算法，其效果都显着优于 ECMP。对于大象流的检测，Hedera 使用了定时读取交换机中所有流表项的统计数据的方式，然而这种方式给交换机和控制路径的网络带宽带来了比较大的负担，有着一定的优化空间。

Mahout[52]试图将大象流探测的功能放到终端的主机上，通过修改主机的操作系统，让主机在一个 TCP 连接传送的数据超过一定限度之后对数据包头部作一个标记，让交换机把携带这一标记的数据包转发给控制器，达到检测大象流的目的。MicroTE[53]使用了这种方法以检测大象流，根据主机之间的流量来估算边缘交换

机之间的流量，并根据估算得到的边缘交换机之间的流量矩阵重新安排各个流的转发路径，修改交换机中的流表项。考虑到数据中心中的主机上使用的操作系统不一定便于修改以实现大象流探测的功能，并且可能安装有虚拟机，这种将大象流探测转移到主机上的方式可能并不适用。

MiceTrap[54]除了使用与 Hedera[51]相同的方法来调度大象流之外，也关注为数众多的老鼠流的转发路径的优化。OpenFlow1.1 及以后的版本支持 group 动作，其中的 select 类型的 group 允许交换机在一组动作列表中以一定的概率选取一个来执行[45]。MiceTrap 把未经调度的老鼠流的转发目标设定为一个 select 类型的 group，它的每一个选项是一个达到目的地的可行输出端口。MiceTrap 根据网络中各条链路的负载情况，调整每一个这样的 group 中各选项的权重，以调节老鼠流选择每一条可能转发路径的概率，使得各条链路上的负载（包括老鼠流和大象流）尽可能均衡，以最大化地利用网络带宽。

此外，OFScheduler[55]通过主机对 IP 头部的 ToS（Term of Service，服务选项）域进行标注，以让控制器区分不同类型的流分别调度，提升了异构网络中 map-reduce 任务的性能；SOTE[56]实现了 SDN 与 OSPF 混合网络中的路由与流调度。

2.4 本章小结

本章介绍了目前已经存在的利用软件定义网络进行网络监控和流调度的研究和应用的现状，展示了已有的通过软件定义网络进行网络监控的应用，包括基于 OpenFlow 协议的和 OpenFlow 框架之外的，以及在软件定义网络中进行流调度的应用。本章介绍了这些应用的设计与实现的基本思路，并指出了这些应用仍然存在的局限性，以及造成这些局限性的原因。这些应用普遍存在的局限性是由 OpenFlow 协议缺乏由交换机主动向控制器上报其状态与统计数据的途径这一限制带来的。因此，某些应用为保证准确性，让控制器向交换机下发大量读取统计数据的请求，带来较大的网络开销；有些应用为减少网络开销牺牲了统计数据的准确性和有效性；有些应用跳出 OpenFlow 框架，增加了实现难度，降低了通用性。

第三章 OpenEvent 事件框架设计

为了实现交换机主动通过事件机制向控制器上报控制器所需要的统计数据的功能，我们设计并实现了 OpenEvent 这一事件框架，让控制器能够配置交换机上的事件，交换机能够正确地监测与上报事件。OpenEvent 实现为 OpenFlow 的扩展，兼容于 OpenFlow。

虽然目前开放网络基金会已经发表 OpenFlow notification framework 框架[57]用于 OpenFlow 交换机向控制器或其他 OpenFlow 网络实体通知事件的发生，但据我们所知，目前并无任何控制器或交换机实现了这一标准。OpenFlow 协议的 1.5 版中也提出了新的扩展，让交换机在流表项的统计数据达到某一阈值时上报一个事件通知控制器[45]，但这一功能只能以流表项为单位上报事件，不具备通用性和可扩展性。而且，目前也没有控制器和交换机已经实现了这一扩展。因此，我们设计并实现了 OpenEvent 这一 OpenFlow 基础上的扩展，以在 OpenFlow 中实现事件机制让交换机根据控制器的要求主动上报自身的统计数据，并且定义与实现了两类事件以获取交换机端口的统计数据和交换机中流表项的统计数据。

3.1 OpenEvent 框架架构设计

OpenEvent 这一框架的设计目标有如下三点：通用性，可以容许在这个框架下定义所需要的任何类型的事件；可扩展性，可以容许这个框架下的事件拥有各种类型的属性和运行方式；与 OpenFlow 的兼容性，OpenEvent 将成为 OpenFlow 中的一个扩展而不是完全独立的协议。

在通用性方面，因为 OpenEvent 框架需要用于定义各种类型的事件以使得交换机主动地向控制器上报各种类型的状态信息和统计数据，所以 OpenEvent 不限制在该框架下定义的事件类型。在交换机上发生的，可以有效监测并需要通知控制器的任何事都可以被定义为 OpenEvent 框架下的事件。这里所指的“任何事”，包括但不限于以下事件：交换机的状态发生变化（例如，某个端口连接或断开连接）；交换机上的某个统计数据达到或超过控制器给定的一定阈值（例如，满足一定条件的流表项在一定时间间隔内匹配的数据包数量或字节数超过控制器给定的阈值）；交换机的配置发生变化等。

在可扩展性方面，OpenEvent 只规定了作为一个事件所需要的最基本的一些属性作为事件的必需属性。事件的属性中，只有这些必需的基本属性的格式与含

义由 OpenEvent 框架指定，其他的属性 OpenEvent 不做规定。事件的其他所有属性的表示格式与含义是由具体类型的事件定义的，其形式可以任意给定。这样，OpenEvent 框架就可以允许各种类型的事件根据自身需要定义所需要的属性。

在与 OpenFlow 兼容性方面，OpenEvent 实现为 OpenFlow 的一个扩展，而非完全独立的协议。OpenEvent 中使用的所有消息均为 OpenFlow 中的自定义类型消息。OpenEvent 完全兼容 OpenFlow 从 1.0 到 1.5 的各个版本。对于 OpenEvent 中使用的消息，控制器和交换机只需要用与处理其他 OpenFlow 自定义类型消息相同的方式来处理。如控制器或交换机不支持 OpenEvent，它们只需无视相关的消息并发出一条 OpenFlow 错误以报告自身无法处理这一类消息。

OpenEvent 包括控制器端和交换机端两大部分，以及控制器与交换机之间的通信协议。OpenEvent 的整体架构如图 3-1 所示。该图包括了 OpenEvent 框架中，控制器端与交换机端的各个模块及模块之间的关系，以及控制器端与交换机端、上层应用于控制器端之间的交互接口。

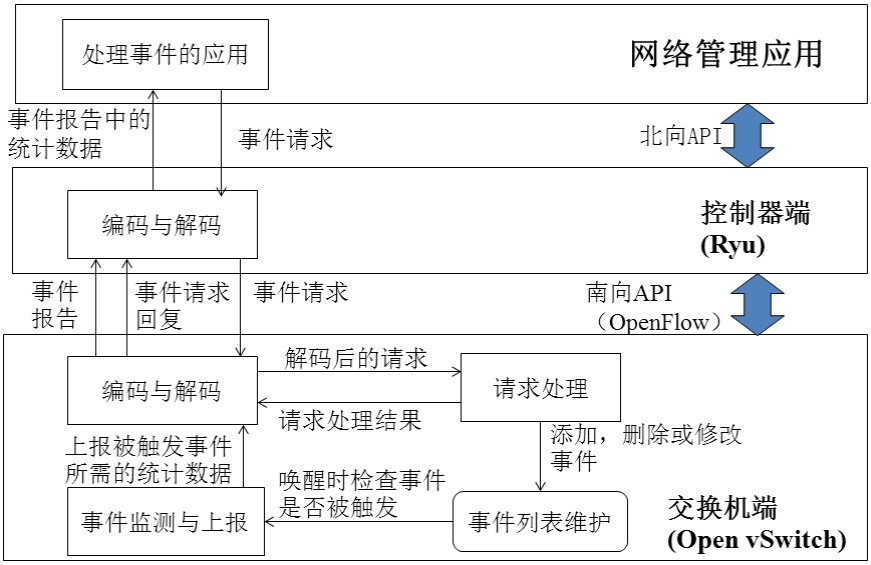


图 3-1 OpenEvent 的架构示意图
Fig.3-1 Architecture of OpenEvent

控制器端的主要模块是用于支持通信协议，编码与解码 OpenEvent 中使用的消息的编码与解码模块。控制器端的编码与解码模块将上层应用需要安装的事件编码为 OpenEvent 中的事件请求，发送给交换机，并解码从交换机上收到的事件报告，将解码后的统计数据发送给需要使用这些统计数据的上层应用。

交换机端的模块主要包括以下几个：编码与解码模块，请求处理模块，事件监测与上报模块，事件列表维护模块。编码与解码模块负责以下的编码与解码工

作：解码控制器端发来的，对交换机上的事件进行配置的事件请求消息，并将解码后的请求发送至请求处理模块；接收请求处理模块处理完成事件请求之后的结果，将结果编码成事件请求回复的消息，回复控制器的事件请求；接收来自事件监测与上报模块的触发事件的消息和统计数据，并编码为事件报告，发送给控制器。请求处理模块负责接收已经解码的事件请求，并在事件列表中添加、删除或修改所需要配置的事件，然后向编码与解码模块通知处理该请求的结果。事件监测与上报模块负责检查事件列表中已经安装的事件是否满足了触发条件，如有事件满足了其触发条件，事件监测与上报模块将该事件中规定的统计数据收集起来，上报给编码与解码模块，让其向控制器发送事件报告。事件列表维护模块维护该交换机上已经安装的事件的配置，以及其上次触发时间、触发条件、触发次数等信息，并负责在列表中添加、删除或者修改一个事件的上述信息。

在 OpenEvent 框架中，一个事件所必需的属性包括：周期性、事件类型、事件 ID。为保证可扩展性，事件的其他属性在框架中不做规定，由具体的事件类型来定义。周期性表示该事件是否可以重复被触发，若设定为周期性触发，该事件将在每次满足其触发条件时被触发，并产生事件报告，向控制器发送统计数据；若设定为非周期性，则该事件仅在第一次满足其触发条件时被触发。事件类型为一个 16 位整数，用来区别事件的所属类型。事件 ID 是一个事件的唯一标识符，在一个交换机内，所有事件的事件 ID 都是唯一的。事件 ID 为一个 32 位整数，从 1 到 0xFFFFFF00 的事件 ID 均为可用 ID，事件 ID 为 0 表示该事件的事件 ID 未确定，超过 0xFFFFFF00 的事件 ID 为特殊用途保留。

3.2 OpenEvent 框架运行流程设计

本节将介绍 OpenEvent 在控制器与交换机上运行的基本流程。在启动时，交换机上没有任何事件，交换机上的所有的事件都由控制器来配置。控制器可以通过向交换机发送请求，在交换机上添加新的事件，修改交换机上已经安装的事件，或者删除交换机上已经存在的事件。交换机收到控制器发来的添加、修改或删除事件的请求后，处理请求并将处理的结果回复给控制器。在交换机上，当一个事件的触发条件被满足时，该事件被触发，交换机向控制器发送一个报告以通知控制器该事件被触发。报告中包含被触发事件中规定的自身状态信息与统计数据。

当控制器要求交换机添加事件时，交换机无论是否已经安装触发条件相同的事件，均应添加新的事件。如果添加成功，交换机中安装的事件增加一个，并且该事件的存在时间从安装成功时计算，上次触发时间为无意义值，被触发次数为

0。若添加失败，交换机状态无任何变化。控制器要求交换机修改事件的配置时，交换机如果修改成功，从下一次触发事件开始，该事件的配置应当已经修改为新的配置。如果修改失败，事件的配置应当仍然保持收到修改请求之前的配置。控制器要求交换机删除一个已安装的事件时，该事件应当立即从交换机中删除，如成功删除，该事件将不再被触发；若删除失败，该事件的配置应当仍然保持收到删除请求之前的配置，并且仍然可以在满足触发条件时被触发。

在交换机上，当一个事件满足其触发条件而被触发时，该事件的上次触发时间应被更新为当前时间，触发次数增加 1，并将该事件中规定的自身状态信息与统计数据上报给控制器。触发之后，如果该事件为非周期性触发，该事件应当被删除而不再被触发；若为周期性触发，该事件的状态应恢复到触发之前，下一次满足触发条件时仍然会被触发。

3.3 OpenEvent 中的事件类型实例

OpenEvent 这一框架本身可以支持各种类型的事件以获取各种类型的统计数据。本节将介绍我们定义并实现的两类事件，用于获取两类在 OpenFlow 网络的网络监控与流调度工作中最常用到的两类统计数据：端口统计数据与流表项统计数据。

为了获取 OpenFlow 交换机上的端口的统计数据，我们定义了端口统计事件。这类事件将在某个端口在一定时间内发送的数据包数量或字节数，或者接收的数据包数量或字节数在一定时间间隔内达到或超过一定阈值的时候被触发。端口统计事件所监测的端口必须是交换机上实际存在的端口，其配置中必须给定一个交换机上存在的端口对应的端口号，且该端口号不能是特殊端口号，如控制器端口（OFPP_CONTROLLER）、全部端口（OFPP_ALL）等。端口统计事件给定的时间间隔以毫秒为最小单位。其监测的指标可以包括以下四个指标中的一个或多个：端口发送的数据包数量、端口发送的数据字节数、端口接收的数据包数量、端口接收的数据字节数。这四个指标可以分别给定不同的阈值。当监测的指标包括四个指标中的多个时，任意一个指标在时间间隔内达到或超过了给定的阈值，都视为满足了事件的触发条件，该事件应当被触发。端口统计事件的该“时间间隔”从安装事件成功或者上一次触发事件开始计算。

为了获取满足一定条件的流表项的统计数据，我们定义了流统计事件。此类事件当满足一定条件的任意一条流表项在给定的时间间隔内新匹配的数据包数量或匹配数据包字节数达到或超过一定阈值，或者在该给定时间间隔内匹配的数据

包总数或匹配的总字节数刚好越过了一定阈值时被触发。当流统计事件被触发时，交换机应当上报所有满足触发条件，也即满足事件中给定条件并且其给定统计数据超过了阈值的流表项各自的统计数据。这些统计数据包括满足触发条件的所有流表项各自的存在时间、在给定时间间隔内匹配的数据包数量与字节数、存在期间内匹配的数据包总数与总字节数等。给定的流表项的条件类似于 OpenFlow 中读取流表项统计数据中给定的流表项的条件类似，包括流表项所在流表的 ID、流表项的输出端口、流表项的匹配范围、流表项的 cookie 值与 cookie 掩码等条件。满足事件中给定条件的流表项都在这个事件的监测范围内。流统计事件给定的时间间隔也以毫秒为最小单位。流统计事件所监测的指标包括如下四种，事件可以选择监测其中的一种或多种：流表项在给定时间范围内匹配的数据包数量、流表项在给定时间范围内匹配的字节数、流表项存在期间匹配的数据包总数、流表项存在期间匹配的字节总数。这四个指标的阈值可以分别给定。如果事件监测的指标包括多个，则任意一个指标达到或超过阈值均满足触发条件。当监测范围内的任意一个流表项的给定统计数据达到或超过了设定的阈值而触发事件，交换机应向控制器上报监测范围内所有统计数据达到或超过了这一设定阈值的流表项的统计数据。流统计事件的“时间间隔”同样从安装事件或者上一次被触发开始计算。

除了这两类事件之外，我们也可以根据实际需要在 OpenEvent 框架中定义其他类型的事件。

3.4 本章小结

本章从基本架构和运行流程两个方面介绍了 OpenEvent 这一在 OpenFlow 网络中实现事件机制的框架，以及在 OpenEvent 框架下定义的两种类型的事件。OpenEvent 是 OpenFlow 的自定义扩展，可以容许在这个框架下定义各种类型的事件。控制器可以通过 OpenEvent 框架指示交换机添加、修改与删除事件，令交换机在指定的条件下上报需要的统计数据。OpenEvent 包括控制器端、交换机端两大部分，以及规定控制器端与交换机端交互的 OpenFlow 消息的格式与含义的通信协议。OpenEvent 框架具有通用性、可扩展性、与 OpenFlow 兼容等特点。本章还介绍了 OpenEvent 框架下的两类事件：端口统计事件，用于获取交换机的端口上发送或接收的数据包数量或字节数；流统计事件，用于获取一定范围内的流表项的匹配的数据包数量或匹配的字节数。这两类事件可以用于获取网络监控与流调度最为需要的一些统计数据，为网络监控与流调度的上层应用提供必需的支持。

第四章 OpenEvent 事件框架与框架中事件实例的实现

OpenEvent 事件框架与我们在 OpenEvent 框架中定义的两类事件的实现都包含通信协议、控制器端和交换机端三部分。本章将分别介绍这三部分的实现细节。

4.1 通信协议

OpenEvent 事件框架定义了一系列的 OpenFlow 消息，用于控制器与交换机之间对事件请求、事件请求回复和事件上报。所有的消息均属于 OpenFlow 自定义类型（vendor / experimenter）消息。这一类消息是预留给设备制造商或者其他使用者实现自定义 OpenFlow 扩展的预留接口。OpenFlow 自定义类型消息的格式如图 4-1 所示。OpenFlow 自定义类型消息需要包含制造商或扩展开发者的 32 位使用者 ID（vendor ID / experimenter ID）。子类型部分表示厂商或开发者对属于同一使用者 ID 下各类型消息的分类。OpenFlow 对自定义类型消息内容部分的格式，以及控制器与交换机对此类消息的处理方式完全没有规定，可以自主扩展。Open vSwitch[9]中就存在 Nicira 公司定义和实现的大量扩展消息，它们采用的使用者 ID 为 0x00002320。

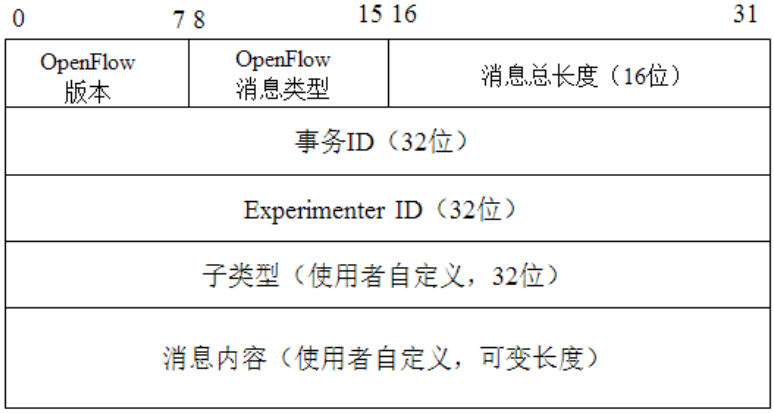


图 4-1 OpenFlow 中自定义类型消息的格式
Fig.4-1 Format of vendor/experimenter messages in OpenFlow

OpenEvent 中的消息均用使用者 ID 0xEBCC3118 来区分。该框架中包含三大类消息：事件请求、事件请求回复、事件报告。这三大类消息用子类型来区分。事件请求用于控制器对交换机上的事件进行操作；事件请求回复用于交换机向控制器回复事件请求的执行结果；事件报告用于交换机在某一事件发生时，向控制

器报告该事件的发生以及该事件中指定的统计数据。

4.1.1 事件请求

事件请求消息由控制器向交换机发出，用于描述控制器对交换机上的事件的添加、删除或者修改配置的请求。事件请求消息的子类型为 0。该类消息的消息内容部分的结构如图 4-2 所示。“周期性”（PERIODIC）标志用于指示该事件是否可以重复被触发。当该标志被设为 1 时，该事件具有周期性，每当事件的触发条件被满足时，该事件都会被触发，并上报控制器；设为 2 时，该事件为一次性事件，仅在第一次满足触发条件时被触发。事件 ID 为该事件在交换机中的唯一标识符。事件类型表示该事件所属的类型，我们为目前已经定义和实现的两类事件指定了事件类型的值，其他的值可以供将来定义的事件类型使用。

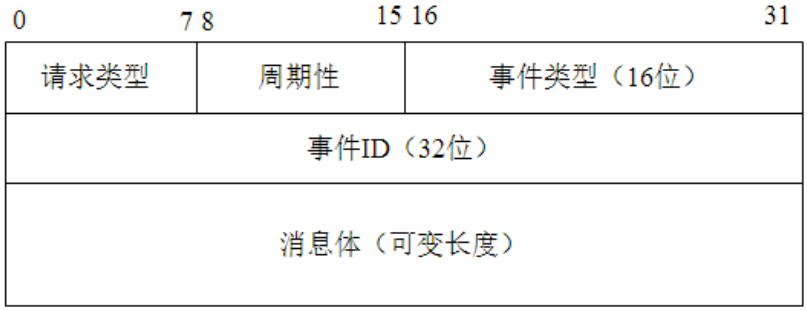


图 4-2 OpenEvent 中事件请求的消息格式
Fig.4-2 Format of event request messages in OpenEvent

目前，事件请求分为三种类型，由请求类型域的值来指定。这三种请求类型为：添加事件请求、修改事件请求、删除事件请求。

添加事件请求使用请求类型 0。控制器通过向交换机发送添加事件请求来要求交换机安装并监测一个新的事件。在添加事件请求中，事件 ID 总是被设为 0，以表示这个事件的 ID 还没有被确定。交换机应当为该新安装的事件指定一个非 0、在该交换机中唯一的事件 ID，并在回复中上报。其他必要的对该事件的配置在请求的消息体部分指定，消息体部分的格式与解释方式由各类具体的事件来规定。

修改事件请求的请求类型为 1。控制器通过向交换机发送修改事件请求来修改一个交换机上已经存在的事件的配置。在请求中，控制器必须在事件 ID 和类型部分给出所要修改的事件的正确的 ID 和类型。如果在交换机中，事件 ID 所对应的事件不存在或者事件类型与控制器给出的不符，交换机将拒绝执行该请求并在回复中报告错误。与添加事件请求类似，修改事件请求的消息体部分由具体的事件类型指定，一般应包含修改后的事件配置。

删除事件请求使用的请求类型为 2。删除事件请求用于控制器要求交换机删除一个已经安装的事件。与修改事件请求类似，删除事件请求需要正确地给出被删除的事件的 ID 和事件类型。如果与 ID 对应的事件不存在或者类型不符，该请求也不会被执行，并且交换机将给出出错信息。删除事件请求的消息体为空。

4.1.2 事件请求回复

事件请求回复消息由交换机向控制器发出，用于回复控制器发出的事件请求。事件请求回复消息与它所回复的事件请求具有相同的 OpenFlow 事务 ID。事件请求回复消息的子类型为 1，它的消息内容部分的格式如图 4-3 所示。事件请求回复消息应当包含交换机执行其所回复的事件请求的执行结果。

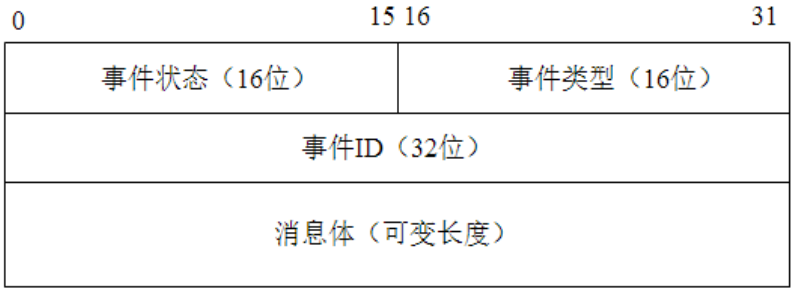


图 4-3 OpenEvent 中事件请求回复消息的格式
Fig.4-3 Format of event reply messages in OpenEvent

事件请求的执行状态在事件状态中指定。本文在表 4-1 中定义了几种状态对应的事件状态值，用于表示几种常见的可能出现的执行状态。

表 4-1 OpenEvent 中几种事件状态值的定义
Table 4-1 Definition of several constants for event status in OpenEvent

常量名称	数值	含义
EVENT_ADDED	1(0x01)	事件添加成功
EVENT_MODIFIED	2(0x02)	事件修改成功
EVENT_DELETED	3(0x03)	事件删除成功
UNSUPPORTED	5(0x05)	不支持此类型的事件
NO_PORT	6(0x06)	指定的端口不存在
WRONG_TYPE	14(0x0E)	事件类型错误
NO_EVENT_ID	15(0x0F)	事件 ID 对应的事件不存在
UNKNOWN_ERROR	65535(0xFFFF)	以上未定义的其它错误

事件类型部分应当与所回复的事件请求中指定的事件类型相同。对于事件 ID 部分，若所回复的事件请求是添加事件，那么事件请求中的事件 ID 为 0，交换机

成功添加事件后，应当将生成的事件 ID 填入事件 ID 部分。若事件请求中给出的事件 ID 不为 0，而请求执行成功，那么回复中的事件 ID 应当与请求中的相同。如果请求执行失败，事件 ID 应当被设为特殊值 EVENT_ID_ERROR = 0xFFFFFFF0 以表示在执行事件请求时发生了错误。消息体部分由具体的事件类型指定，也可以为空。

4.1.3 事件报告

事件报告消息由交换机向控制器主动发送，用于通知控制器自身的某一事件达到了触发条件，已被触发。事件报告消息应当向控制器上报被触发事件中规定应当上报的统计数据。事件报告消息所使用的子类型为 2，它们的消息结构如图 4-4 所示。

图 4-4 OpenEvent 中事件报告的格式
Fig.4-4 Format of event report messages in OpenEvent

上报原因部分表示交换机向控制器发送本条事件报告消息的原因。我们目前仅为事件触发这一种上报原因定义了对应的值 EVENT_TRIGGERED = 1。事件类型与事件 ID 部分为所上报事件的类型与事件 ID。事件报告的消息体部分包含所上报事件指定的统计信息，这部分消息的格式同样由具体的事件类型来指定。

4.1.4 OpenEvent 中两类事件的相关消息格式

端口统计事件使用的事件类型为 1。在添加事件请求与修改事件请求的消息体中，对于端口统计事件的描述的消息格式如图 4-5 所示。其中，端口编号表示该事件需要检测的端口编号，如果在交换机上，请求中给定的编号对应的端口不存在，交换机应在对该请求的回复中指出这一错误。时间间隔秒数与时间间隔毫秒数共同指定该事件被检查和触发的时间间隔，检查和触发的时间间隔等于秒数与毫秒数之和。事件触发条件这一属性用于指定需要检测的统计数据，其数值为

以下代表不同统计数据的常量的按位或的结果：1 表示检测发送的数据包数，2 表示发送的字节数，4 表示接收的数据包数，8 表示接收的字节数。对应位上为 1，表示这个统计数据如果在给定的时间间隔内超过下面给定的阈值，该事件就被触发。发送数据包数阈值、发送字节数阈值、接收数据包数阈值、接收字节数阈值均为 64 位整数，分别指定这些统计数据的阈值，或设定为 2^{64}-1 表示阈值未指定。

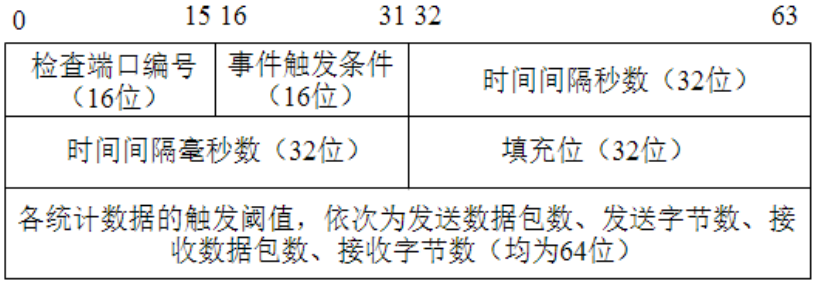


图 4-5 端口统计事件的事件请求描述格式
Fig.4-5 Format of description of events on port statistics

端口统计事件被触发后，交换机向控制器发送的事件报告的消息体部分如图 4-6 所示。上报的统计数据依次为：在事件给定的时间间隔发送的数据包数量、给定时间间隔内发送的数据字节数、时间间隔内接收数据包数量、时间间隔内接收数据字节数；端口发送的数据包总数、端口发送的数据总字节数、端口接收的数据包总数、端口接收的数据的总字节数。

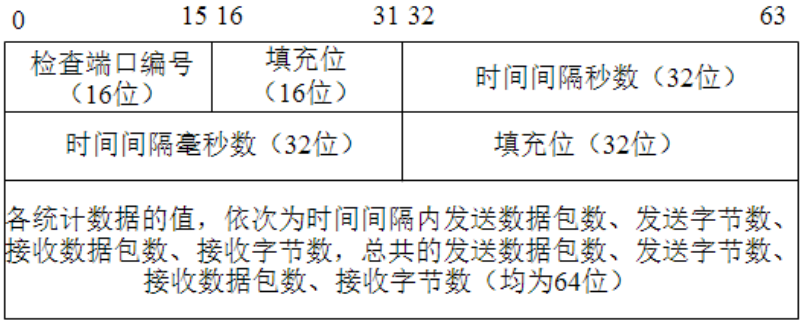
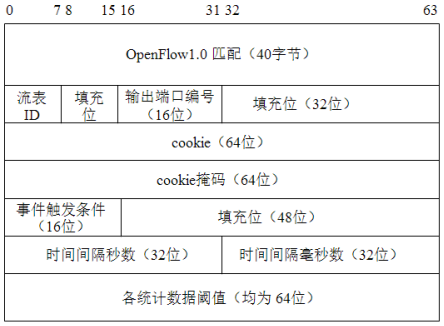


图 4-6 端口统计事件的事件报告格式
Fig.4-6 Format of event report messages of events on port statistics

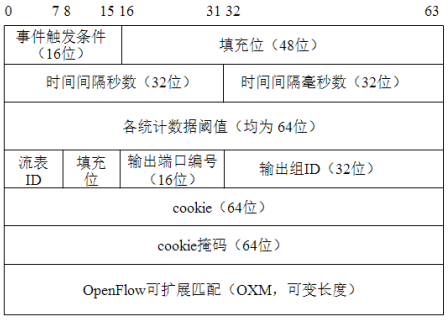
流统计事件使用的事件类型为 3。由于不同版本的 OpenFlow 协议中，对流表项的描述也不相同，因此，该类型的消息在不同版本的 OpenFlow 协议中也有不同的形式。在 OpenFlow1.0 与 OpenFlow1.1 以上的版本中，流统计事件的描述格式分别如图 4-7(a)与 4-7(b)所示。其中流表 ID 表示流表项所在的流表 ID，若该流表项 ID 被指定为某一特定 ID，则只有在 ID 所指定的流表中的流表项才在检测范

围内；输出端口表示流表项输出的端口，如果该项被指定为一个具体的端口，只有直接输出到该端口的流表项在检测范围内；cookie 与 cookie 掩码为流表项的特定标识，只有流表项的 cookie 和 cookie 掩码按位与的结果与事件中指定的 cookie 和 cookie 掩码的结果相同时，该流表项才属于检测范围。输出组表示流表项直接输出的 group 的 ID，当该域被指定为一个特定 group 的 ID 时，只有直接输出到该组的流表项属于检测范围。OpenFlow1.0 中的流表项匹配范围使用 OpenFlow1.0 中规定的标准格式来描述；OpenFlow1.1 以上版本中，流表项的匹配范围使用 OpenFlow 可扩展匹配（OXM）描述，以兼容 OpenFlow 协议。时间间隔秒数与时间间隔毫秒数共同指定该事件被检查和触发的时间间隔，检查和触发的时间间隔等于秒数与毫秒数之和。事件触发条件为以下四个代表关于不同统计数据的触发条件的按位或的结果：1 代表时间间隔内新匹配的数据包数，2 代表时间间隔内匹配的数据字节数，4 表示流表项存在期间内匹配的数据包总数，8 表示流表项存在期间内匹配的数据总字节数。若对应位上为 1，则表示如果一个在检测范围内的流表项在时间间隔内，该值表示的统计数据达到或超过了给定的阈值，该事件就被触发。



a) OpenFlow 1.0 下的 OpenEvent 中对流统计事件的描述格式

a) Description of events on flow statistics in OpenEvent on OpenFlow 1.0



b) OpenFlow 1.1 以上版本下的 OpenEvent 中对流统计事件的描述格式

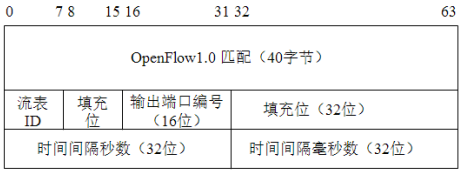
b) Description of events on flow statistics in OpenEvent on OpenFlow 1.1 and later

图 4-7 不同版本的 OpenFlow 协议下的 OpenEvent 中对流统计事件的描述格式

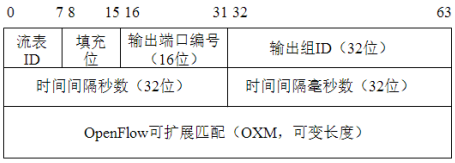
Fig. 4-7 Formats of description of events on flow statistics in OpenEvent on different versions of OpenFlow protocol

当流统计事件被触发时，交换机应在事件报告上报中所有在该事件检测范围内，并且满足了触发条件的流表项的统计数据。事件报告的消息体部分由头部和各流表项各自的统计数据两部分组成，头部与各流表项单独的统计数据在

OpenFlow1.0 与 OpenFlow1.1 及以后的版本中均有不同格式。流统计事件的事件报告的消息体头部的格式在 OpenFlow1.0 及 OpenFlow1.1 以上中分别如图 4-8(a)与 4-8(b)所示，各流表项的单独统计数据在 OpenFlow1.0 及 OpenFlow1.1 以上中分别如图 4-9(a)与 4-9(b)所示。

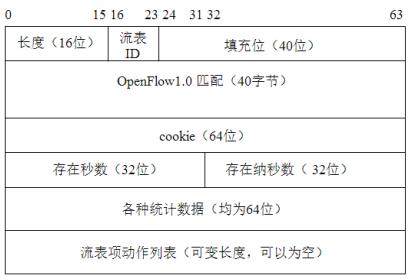


a) OpenFlow 1.0 下的 OpenEvent 中，流统计事件的事件报告头部
a) Header of event reports of events on flow statistics in OpenEvent on OpenFlow 1.0

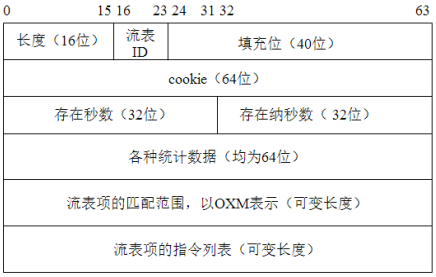


b) OpenFlow 1.1 以上版本下的 OpenEvent 中，流统计事件的事件报告头部
b) Header of event reports of events on flow statistics in OpenEvent on OpenFlow 1.1 and later

图 4-8 流统计事件的事件报告头部格式
Fig.4-8 Format of headers of event reports of events on flow statistics



a)OpenFlow 1.0 下的 OpenEvent 中，流统计事件的事件报告中单个流表项的统计数据描述格式
a)Format of statistics of a single flow entry in event reports of flow statistics in OpenEvent on OpenFlow 1.0



b) OpenFlow 1.1 以上版本下的 OpenEvent 中，流统计事件的事件报告中单个流表项的统计数据描述格式
b) Format of statistics of a single flow entry in event reports of flow statistics in OpenEvent on OpenFlow 1.1 and later

图 4-9 流统计事件的事件报告中单个流表项的统计数据描述格式
Fig.4-9 Format of statistics of a single flow entry in event reports of events on flow statistics

头部包括的以下数据：流表 ID、输出端口、输出组 ID、流表项匹配范围、时间间隔秒数、时间间隔毫秒数均与事件中指定的值相同。各流表项的单独统计数据中，每一条记录表示一个流表项的统计数据。最开头的长度表示该条流表项的统计数据的总长度；流表 ID 表示该流表项所在的流表 ID；cookie 表示该流表项

的特定标识符；持续秒数与持续纳秒数表述该流表项的存在时间，流表项的存在时间为两者之和；上报的统计数据中依次为以下四项，其长度均为 64 位：该流表项在事件给定的时间间隔内匹配的数据包数与字节数；该流表项在其存在时间内匹配的数据包总数与总字节数。在 OpenFlow1.0 的单独流表项统计中，流表项的匹配范围以 OpenFlow1.0 的标准格式表达，其最后部分为该流表项执行的动作列表，长度可变；OpenFlow1.1 以上版本的单独流表项统计中，最后部分依次是以 OXM 描述的流表项的匹配范围，以及流表项执行的指令，两者均为可变长度。

4.2 控制器端的实现

我们在 Ryu 控制器提供的框架下实现了 OpenEvent 的控制器端。Ryu 控制器是一个用 Python 语言实现的开源 OpenFlow 控制器，它提供了一个基本的 OpenFlow 消息处理框架与大量基于该框架的应用[19]。Ryu 的整体架构采用事件-处理者模型。Ryu 控制器运行时，最基本的 OpenFlow 消息处理模块以及用户指定的应用将会运行。Ryu 控制器中的每个应用都包含若干个事件处理函数，对某一类事件进行处理。应用也可以产生各种类型的事件，并发送给其他应用。一个事件会被运行中的所有对该类事件进行处理的应用处理。OpenFlow 处理模块是运行 Ryu 控制器时必然会运行的应用，该应用负责接收 OpenFlow 消息并根据 OpenFlow 协议解码这些消息，并产生相应的事件，同时负责编码并发送控制器向交换机发送的 OpenFlow 消息。OpenEvent 的控制器端实现为 Ryu 框架下的一个应用，负责处理 OpenEvent 框架与事件相关的 OpenFlow 消息，包括编码并发送事件请求和接收并解码事件请求回复与事件报告，并产生事件以通知其他应用两部分。

为实现接收并解码的功能，该应用注册了一个处理收到 OpenFlow 自定义类型消息事件的函数，该函数检查收到的自定义类型消息的 Experimenter ID 是否是我们指定的 0xEBCC3118，如果符合，则进行进一步的解码；如果不符合，则处理到此为止。进一步的解码中，该函数根据消息的子类型与消息中指定的事件类型进一步调用各类具体的消息的解码函数。对于事件请求回复，解码函数从消息中提取出事件类型、事件状态、事件 ID 以及消息体，并产生一个“收到事件请求回复”的事件，通知其他可能需要处理这个消息的应用。对于事件报告，解码函数从消息中提取出事件 ID、事件类型，以及该事件类型规定的各种交换机的统计数据。对于端口统计事件，统计数据包括该端口在指定的时间间隔内发送与接收的数据包数量与字节数，以及该端口发送和接收的总数据包数量和字节数；对于流表项统计事件，上报的统计数据包括满足触发条件的所有流表项分别的存在时长、

在指定时间间隔内匹配的数据包数量和字节数、流表项存在期间匹配的数据包总数和字节总数等。解码函数提取出事件 ID、事件类型和统计数据之后，产生“收到事件报告”的事件，发送给可能需要处理这一事件报告的其他应用。

对于编码并发送部分，该应用提供接口让其他的应用给定事件请求的类型、事件类型、事件 ID 以及消息体，对这些信息编码成为可以通过 OpenFlow 连接发送的消息。

由于 OpenFlow 协议的不同版本在 OpenFlow 头部的版本号、指定流表项的匹配范围的格式等方面存在区别，我们实现了在 OpenFlow 协议 1.0 版本和 1.3 版本下适用的两个不同版本的应用。

4.3 交换机端的实现

OpenEvent 的交换机端的功能除了对于事件相关的各种 OpenFlow 消息的编码与解码之外，还包括对自身已安装的事件的维护、监测和上报。为实现 OpenEvent 的交换机端，我们对 Open vSwitch[9]的多个模块做了改动，以实现 OpenEvent 所需要的功能。

Open vSwitch 解码 OpenFlow 消息时首先根据其头部的类型这一属性查找所属类型，再根据消息体的头部一些指明细分类型的属性查找到细分类型，再调用各细分类型的解码函数。编码 OpenFlow 消息时，Open vSwitch 也根据消息所属的细分类型调用对应的编码函数。对于自定义类型消息，Open vSwitch 将具有不同使用者 ID 与或同一使用者 ID，但子类型不同的消息视为不同细分类型，对于每一类细分类型的消息，我们需要在 Open vSwitch 中注册，并指明编码与解码的函数。为了让 OpenEvent 的消息能够正确地在 Open vSwitch 中被编码和解码，我们为 OpenEvent 的三类消息：事件请求、事件请求回复、事件报告注册了三种细分类型，以及相应的编码与解码函数。对于事件请求，Open vSwitch 收到控制器发来的 OpenEvent 事件请求后，首先根据消息的头部调用对应的事件请求的解码函数。该解码函数继续读取事件请求的消息体部分，根据请求的类型与请求的事件类型进一步调用针对各类具体事件的解码函数，完成对事件请求的解码之后，根据请求的内容调用请求处理模块中处理相应请求的功能。对于事件请求回复，我们实现了对事件请求回复消息的编码函数，当请求处理模块完成了对事件请求的处理之后，调用对应的编码函数。该编码函数读取请求处理的结果、对应请求的事务 ID、请求的事件类型与事件 ID 以及其他必要的信息编码成事件请求回复消息，发送给控制器。对于事件报告，事件被触发时，事件监测与上报模块首先监

测到，并调用被触发事件对应的编码事件报告消息体的编码函数。我们针对目前已经定义的端口统计事件与流统计事件定义并实现了对应的编码函数。然后，编码与解码模块将消息体与 OpenFlow 头部一同编码成完整的事件报告，发送给控制器。

Open vSwitch 中，每一台交换机中已安装的事件由一个列表维护，每一个事件在列表中有一条对应的记录，该记录包含其事件类型、事件 ID、安装时间、上次触发时间、已被触发的次数，以及由于事件类型不同而具有不同结构的事件配置。请求处理模块接收到添加事件的请求后，首先根据控制器对该事件的配置构建一条新的事件记录，如果该事件合法，可以成功添加，请求处理模块为该事件生成一个新的事件 ID，插入到事件列表中，并返回添加成功的结果，让编码与解码模块去回复给控制器；如果添加失败（例如，端口统计事件中指定了不存在的端口编号），请求处理模块返回相应的错误信息，由编码与解码模块在对请求的回复中报告这一错误信息。对于修改事件请求，请求处理模块首先检查控制器给定的事件类型与事件 ID 所对应的事件在列表中是否存在。如果事件 ID 对应的事件在列表中不存在，请求处理模块返回“事件不存在”错误；如果事件 ID 对应的事件在列表中存在，但其事件类型与控制器给定的事件类型不相同，请求处理模块返回“事件类型错误”的错误信息。然后，请求处理模块判断控制器对该事件给出的修改后配置是否合法，如果合法，则修改事件列表中的对应记录，将它的配置改为控制器指定的新配置，并返回修改成功的结果；若控制器给定的修改后配置不合法，则请求处理模块返回相应的错误信息，并且事件配置保持不变。对于删除事件请求，处理的原则与方法与修改事件请求类似。

Open vSwitch 中，一台交换机可以分为待机和唤醒两个状态，当交换机待机时，它运行 ofproto_wait 函数等待被唤醒。可能的唤醒原因包括收到 OpenFlow 消息、内核模块部分调用了一个 upcall、事先设定的定时器超时等。当交换机被唤醒时，运行 ofproto_run 函数来调用检查各个模块是否有需要处理的请求和事件。为了在 Open vSwitch 上实现对事件的检查和上报，我们在交换机的待机状态下添加了 event_wait 函数以在应当检查事件是否发生时唤醒交换机，以及 ofproto_event_run 函数以遍历事件列表，查看每一个事件是否应当被检查。对于应当检查的事件，交换机对事件的类型调用相应的检查函数，读取事件中规定的统计数据并检查是否达到或超过了规定阈值，满足了触发条件。对于满足了触发条件的事件，对事件报告的编码函数将被调用。交换机中，检查与触发事件的流程如图 4-10 所示。在交换机中，我们根据检查事件是否发生的时机，将事件分为

计数器触发与定时器触发两类。计数器触发的事件将在交换机中设定一个计数器，当此计数器的计数值达到一定极限时，交换机将被唤醒，并检查对应的事件是否满足了触发条件。定时器触发的事件将在交换机中设定一个定时器，当定时器达到其超时时间时，交换机被唤醒，并检查对应的事件是否被触发。检查完成后，无论该事件是否被触发，若该事件依然存在，计数器或定时器会复位，以等待下一次检查。当一个事件被删除时，其对应的计数器或定时器也被删除。目前定义的端口统计事件与流统计事件均为定时器触发。event_wait 函数负责等待事件对应的计数器的计数值到达极限时或定时器到达超时时间时唤醒交换机，让交换机调用 ofproto_event_run 函数。ofproto_event_run 函数在运行时，对事件列表进行遍历，检查每一个事件的计数器是否到达极限值或者定时器是否超时。对于计数器到达极限值或定时器超时的事件，ofproto_event_run 将对该事件调用其所属类型的检查函数，检查该事件是否满足了触发条件。如果该事件未满足触发条件，其计数器或定时器被复位以等待下次检查。如果该事件满足了触发条件而被触发，编码与解码模块中编码事件报告的功能将被调用，生成事件报告，向控制器上报该事件的发生，并上报该事件中指定的统计数据。如果该事件被设定为非周期性触发，则将该事件删除；若为周期性触发，其计数器或定时器应被复位，其触发次数增加一次，上次触发时间更新为当前时间。ofproto_event_run 遍历事件列表，对事件进行检查的流程可见图 4-10。对于目前已经定义的两类事件，我们在 Open vSwitch 上实现了检查者两类事件是否满足触发条件，并收集事件报告中所需要的统计数据的检查函数。

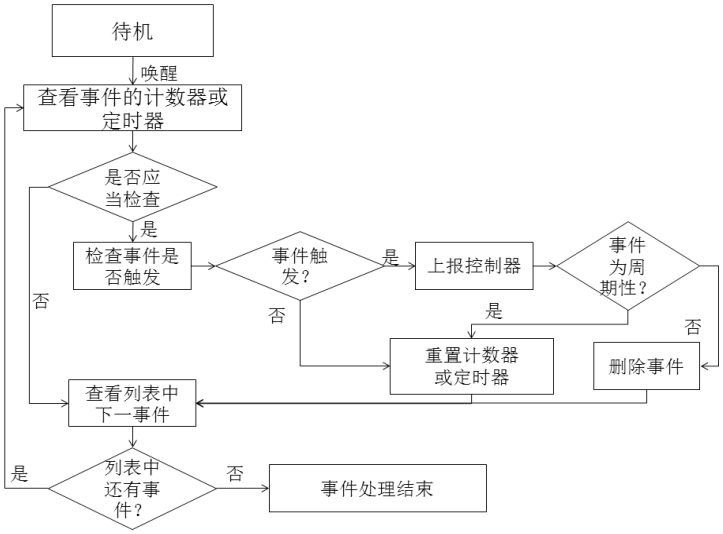


图 4-10 ofproto_event_run 函数检查事件是否触发并上报的流程图

Fig.4-10. Flowchart of function ofproto_event_run、

4.4 本章小结

本章介绍了 OpenEvent 在实现上的细节，包括通信协议上对 OpenFlow 的扩展、在 Ryu 上的控制器端的实现、在 Open vSwitch 上的交换机端的实现三个部分。

通信协议上，OpenEvent 用到的消息全部属于 OpenFlow 自定义消息，其使用者 ID 被规定为 0xEBCC3118。OpenEvent 中用到的消息包括三大类：事件请求、事件请求回复、事件报告，以不同的子类型（subtype）作为区分。事件请求由控制器向交换机发出，用于描述控制器添加、修改、删除交换机上事件的请求。事件请求回复由交换机向控制器发出，用于回复控制器下发的事件请求，反馈请求执行的结果。事件报告由交换机向控制器发出，用与交换机在自身的事件被触发时通知控制器，并上报相关的状态信息与统计数据。由于 OpenFlow 协议的不同版本对流表项的描述格式有较大的不同，针对流统计的事件请求与事件报告的消息体格式在 OpenFlow 的不同版本中也有不同。

控制器端的主要模块包括对 OpenEvent 中的消息进行编码与解码的模块。该模块负责与上层的 SDN 网络管理应用交互，并正确地发送与接收 OpenEvent 中用到的事件请求、事件请求回复、事件报告等 OpenFlow 消息。由于 Ryu 框架本身的原因，以及在不同版本的 OpenFlow 协议中通信协议也存在不同，我们基于 OpenFlow1.0 与 OpenFlow1.3 这两个版本的 OpenFlow 协议分别实现了对应的编码与解码的模块。

交换机端包括编码与解码、请求处理、事件监测与上报以及事件列表等模块。本章的第三节介绍了这些模块的实现。编码与解码模块在 Open vSwitch 中注册了通信协议规定的事件请求、事件请求回复、事件报告等类型的消息，并实现了对这些消息以及已定义的两类事件的消息体的编码与解码函数。请求处理模块负责处理控制器下发的事件请求，并在事件列表中维护事件的记录，对事件列表中的事件进行添加、删除和修改。事件监测与上报模块在需要检查事件是否被触发时遍历事件列表，并检查每一个需要检查的事件是否满足了其触发条件。对于满足了触发条件的事件，该模块收集事件所指定的统计数据，调用对应的编码函数以生成事件报告，向控制器上报。对于满足了触发条件的事件，该模块还会更新列表中的记录（对于周期性触发的事件）或删除该事件（对于非周期性，即一次性触发的事件）。

第五章 OpenEvent 在网络状况监控与流调度上的应用

利用 OpenEvent 这一框架以及在 OpenEvent 框架中定义的两类事件，我们在 Ryu 控制器上实现了网络状况监控与流调度方面的应用。在网络状况监控方面，我们实现了带宽利用状况监测与大象流探测两个应用。在流调度上，我们基于大象流探测的应用，实现了在 Fat tree 拓扑结构的网络中的流调度。本章将介绍这些应用的实现。

5.1 带宽利用状况监测

带宽利用状况监测一直是网络状况监控中的重要组成部分。在非 SDN 网络中，我们通常使用 SNMP 协议[57]来获取；在 OpenFlow 网络中，通常由控制器定期轮询，按照一定的时间间隔向交换机发送请求以读取交换机端口上的统计数据，包括发送的数据包数与字节数、接收的数据包与字节数等，通过端口接收或发送的数据量来得知对应的链路的带宽利用状况。

利用 OpenEvent 框架中定义的端口统计事件，我们实现了基于事件机制的链路带宽利用状况监测这一应用。OpenEvent 中的端口统计事件可以要求交换机在一定的时间间隔内，特定的端口发送或接收的数据包数量或字节数达到或超过给定阈值时，上报该端口的相关统计数据。进行带宽利用状况监测的应用可以通过 OpenEvent 在需要监测的链路的某一端口上安装相应的事件，让交换机在链路的数据传输速率达到一定限度时触发事件并上报该端口的统计数据，以减少控制器与交换机之间的网络通信，并增强控制器端所获取到的带宽利用状况的统计数据的准确性。

利用 OpenEvent 进行链路带宽利用状况的应用会在被监测的链路一端的端口上安装一个事件，要求该端口在给定的时间间隔内，发送或接收的数据的字节数达到或者超过一定阈值时触发事件，并上报统计数据。这一阈值根据所需要获取统计数据的时间间隔（例如，每 1 秒获取一次）与给定的数据传输速率（当该链路的数据传输速率达到或超过此值时，控制器需要获取该端口的统计数据，例如当链路上的数据传输速率达到最大数据传输速率的 1%时）计算。这个阈值也可以被设定为 0，即交换机无论链路的带宽利用状况如何，都以给定的时间间隔向控制器上报对应端口的统计数据。对于安装事件的端口，该应用以如下方式确定：如果被监测的链路两端都是连接同一控制器的 OpenFlow 交换机，则该应用将在具有较小的 DPID 的交换机上的相应端口上安装事件；如果被监测的链路一端是

OpenFlow 交换机，另一端是非 OpenFlow 设备（如主机或不支持 OpenFlow 的交换机、路由器等网络设备），或另一端并未连接同一控制器，则该应用在被其控制的 OpenFlow 交换机上安装检测对应端口的统计数据的事件。

通过在交换机上安装端口统计事件让交换机主动上报端口的统计数据以达成链路带宽利用状况监测的功能，消除了控制器向交换机发送请求时带来的网络开销。通过对事件设定阈值，让交换机仅在链路的带宽利用率达到一定水平时上报统计数据，可以进一步地降低控制器与交换机之间的网络开销。

5.2 大象流探测

在数据中心中，通常会有 10%左右的流，其传输的数据量占到总数据量的 90%[1]。这一类的流传输数据量大、持续时间长，一般称为“大象流”，相对的，持续时间短、总传输数据量较少的流称为“老鼠流”。如何探测出这些大象流，并持续监控这些大象流的统计数据，同样是网络状况监控中的重要问题。探测出大象流，并获取其统计数据也是进行流调度的前提之一。

在 OpenFlow 网络中，控制器通常需要按照一定时间间隔向交换机发送请求以读取所有的流表项，根据每个流表项的匹配数据包数量、匹配数据包的字节数等统计数据，从这些流中找出大象流，并获取与维护大象流的统计数据。由于大象流在所有流中只是少数，控制器读取和处理的大部分流表项的统计数据都来自老鼠流，这意味着用这种简单轮询的方法来探测大象流，消耗的大部分网络资源和控制器上的处理资源都是没有必要的。

利用 OpenEvent 提供的流统计事件，我们实现了基于事件机制的大象流探测功能。OpenEvent 中的流统计事件可以要求交换机在有一个在特定范围内的流表项在一定时间间隔内匹配的数据包数量或匹配的字节数达到或超过一定阈值时触发事件，并上报所有在此范围内，匹配的数据包数量或字节数达到或超过这一特定阈值的流表项的统计数据。流统计事件也可以让交换机在流表项的匹配数据包总数或匹配字节总数在特定时间间隔内跨越某特定阈值时触发事件并上报统计数据。控制器可以通过在交换机上安装流统计事件，让交换机主动上报一定时间间隔内传输速率超过一定值的流的统计数据，以此探测大象流，并持续监控这些大象流的统计数据。

大象流探测的应用在所有的边缘交换机，即至少有一个端口与主机（或其他非 OpenFlow 设备）连接的交换机上安装流统计事件，以达到探测大象流并持续监控的目的。由于所有的流都必须经过边缘交换机，所以只需要在边缘交换机上

安装流统计事件，让它们上报大象流的统计数据，就可以达到在所有流中探测出大象流，并持续监控其统计数据的目的，而不需要从所有的交换机上读取流表项的统计数据。大象流一般有两种界定标准：属于该流的总数据量超过一定范围（如1MB 或 10MB）的视为大象流，或数据传输速率超过一定限度（如 1 秒内的数据传输速率超过 100Mbps）的流视为大象流。 流统计事件可以要求边缘交换机上报在指定时间间隔内，数据的传输速率（单位时间内匹配的字节数或数据包数）超过一定范围的流表项的统计数据，用来获取在数据传输速率意义上的大象流；也可以要求交换机上报在一个特定时间间隔内总数据量（匹配的数据包总数或字节总数）跨越了某一特定阈值的流表项的统计数据，用来获取总数据量意义上的大象流。

通过在边缘交换机上安装流统计事件，让交换机主动上报大象流的统计数据，不但节省了控制器向交换机发送读取流表项统计数据的请求带来的网络上的开销，还可以让控制器仅获取所有大象流的统计数据，而不需要读取并非必需，但占据总数的 90%的老鼠流的统计数据。这样不仅节省了大部分由传输老鼠流的统计数据带来的网络开销，也节省了控制器处理流表项的统计数据，并从中选出大象流的处理开销。

5.3 Fat tree 网络中的流调度

Fat tree 网络是 Clos 网络中的一种特例，是对数据中心网络的研究中最经常使用的模型[59]。标准的 Fat tree 网络中，所有交换机上与其他设备的端口数量都相同，这个端口数代表了网络的规模。Fat tree 网络中，交换机分为三层：接入层、聚合层与核心层。接入层的交换机有一半端口直接与主机相连，另外一半的端口连接聚合层交换机。聚合层交换机一半端口连接接入层交换机，另外一半端口连接核心层交换机。核心层交换机的全部端口与聚合层交换机相连。在由具有 k 个端口的交换机组成的 fat tree 网络中，$k/2$ 个接入层交换机与 $k/2$ 个聚合层交换机组成一个 pod，每个 pod 内，每一台接入层交换机与每一台聚合层交换机两两相连，每一台接入层交换机剩余的 $k/2$ 个端口各连接一台主机。整个网络中一共有 k 个这样的 pod。整个网络中，核心交换机的数量为 $k^2/4$，每个 pod 中，每一台聚合层交换机的剩余与 $k/2$ 个端口各连接一台核心层交换机。一个交换机端口数为 k 的 fat tree 网络中，共有 $k^2/2$ 台接入层交换机，$k^2/2$ 台聚合层交换机，与 $k^2/4$ 台核心层交换机，共 $5k^2/4$ 台交换机，可以支持的主机数量为 $k^3/4$ 台。

Fat tree 网络的这种拓扑结构具有良好的对称性，具有分层结构，不同节点之

间一般存在多条可能的最短路径。由于数据中心的网络通常也属于分层的 Clos 拓扑结构，两节点间的最短路径不唯一，虽然数据中心的网络不一定是标准的 fat tree 网络，研究中也经常使用 fat tree 网络作为数据中心的网络拓扑模型。传统的非 SDN 网络一般采用在接入层及汇聚层交换机上部署 ECMP，让数据包以基本均等的概率选择不同的路径以充分利用 fat tree 网络中各条链路的带宽资源。但是，静态地部署 ECMP 无法根据网络的状态来进行调整转发路径。在存在大象流的数据中心网络中，静态地部署 ECMP 容易造成多个大象流被安排到同一路径上而带来拥塞的现象，从而无法充分利用网络带宽。OpenFlow 网络中，控制器可以读取每个流的统计数据以获取网络状况，并为每个流在交换机上安装对应的流表项以指定其路径。

Hedera 中提出了在 fat tree 网络中进行动态的流调度的算法[51]。该算法只关注网络中的大象流，其中，大象流的标准是数据传输速率达到或超过了链路最大带宽的 10%。对于一般千兆以太网而言，数据传输速率在一定时间内达到或超过 100Mbps（即每秒 12.5MB）的流就被界定为大象流。对于这些大象流，Hedera 首先根据每一台主机发送与接收的大象流数量来计算每一个大象流所需要的带宽资源，然后根据一定的算法为这些大象流重新安排路径。由于安排最优路径的问题是 NP 完全的，Hedera 只给出了可用的近似算法[51]。安排路径的算法包括全局最先匹配算法和模拟退火算法两种。全局最先匹配算法对于每一个大象流，寻找第一条能够满足其带宽需求的路径。对于大象流的探测以及其统计数据的收集，Hedera 采用了按照固定时间间隔向交换机发送请求以读取所有流表项的统计数据的方法。

我们的流调度的应用采用了上一节中的通过在接入层交换机上安装流统计事件以探测大象流并收集大象流的统计数据的方法来实现对大象流的统计数据的读取，对于大象流的带宽需求估计与路径安排，我们的流调度应用采用了 Hedera 中的全局最先匹配算法[51]来为大象流计算合适的转发路径以避免多个大象流被安排到同一路径上带来的拥塞。

我们的流调度应用在控制器端包括三个相对独立的模块：拓扑发现与转发路径计算、大象流探测、流调度，以及这几个模块公用的一些信息，包括两部分：拓扑信息集合与大象流记录。控制器端各个模块的关系以及这些模块与交换机端的交流如图 5-1 所示。这三个模块在我们的实现中是三个独立的 Ryu 控制器上的应用。拓扑信息集合中包含了所有交换机之间的链路，每条链路信息表示一条单向的链路，包括源端的交换机 DPID、源端的端口编号、目的端交换机 DPID、目

的端的端口编号。大象流记录中的每一条记录代表一个已经发现的大象流，该条记录中包括该大象流的源端主机 IP 地址、目的端主机 IP 地址，以及当前已经传输的数据包数量、数据总字节数、流的建立时间和统计数据更新时间等信息。

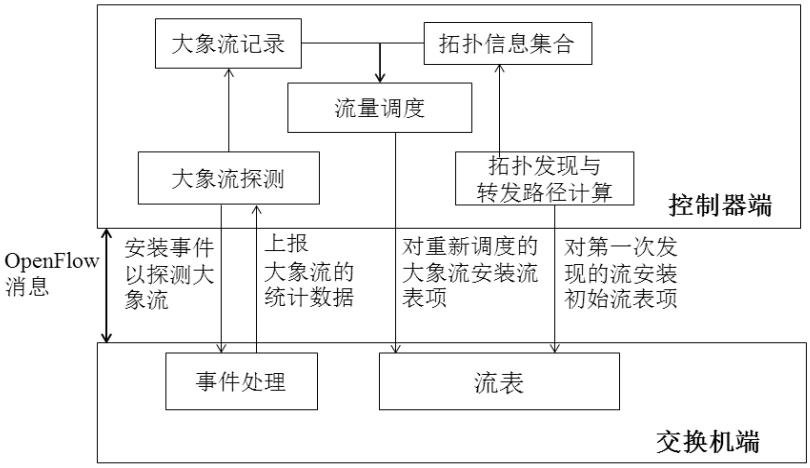


图 5-1 流调度应用架构
Fig.5-1 Architecture of flow scheduler

拓扑发现与转发路径计算模块负责探测并维护网络的拓扑结构，并对每个流根据拓扑结构计算初始的转发路径。这个模块通过向交换机发送 LLDP（链路层发现协议，Link Layer Discovery Protocol，一种链路层协议，用于探测邻居设备）包来探测交换机之间直接相连的链路，进而通过交换机之间的连接关系以确定拓扑。交换机连接到控制器时，该模块会安装一条优先级最高的流表项，要求交换机把所有 LLDP 包通过 packet in 消息转发给控制器。当这个模块发现新的交换机，或者交换机向控制器报告端口状态变化时，该模块向交换机的每一个端口依次发送一条 packet out 消息，要求交换机从该端口上发送一个包含本交换机的 DPID、本端口的端口号、端口的名称等信息的 LLDP 包。如果该端口连接的对端端口也是一个 OpenFlow 交换机，收到这一 LLDP 包的交换机会通过 packet in 消息将这一 LLDP 包转发给控制器。packet in 消息中包含了收到这一 LLDP 包的交换机的 DPID 与收到这一 LLDP 包的端口的编号，而这一 LLDP 包本身含有发出此包的交换机的 DPID 与发出此包的端口的编号。因此，控制器收到这一 LLDP 包之后，就找到了一条连接两台交换机之间的单向链路。拓扑发现与转发路径计算模块在发现一条链路后，就会把这条链路加入拓扑信息集合。

拓扑发现与转发路径计算模块还负责对所有流的初始转发路径的计算。当有从未出现过的流到达一个交换机时，交换机由于匹配失败，会把初始的数据包通过 packet in 消息转发给控制器。控制器接收到 packet in 消息后（除 LLDP 包外），

交由拓扑发现与转发路径计算模块进行处理。该模块根据数据包的头部，查找出需要转发到的最终与目标主机直接相连的接入层交换机的对应端口，并计算发出此 packet in 包的交换机到最终目标的最短路径以及最短路径上所有可能的输出端口，如果可能的输出端口唯一，控制器直接将该流的输出端口设定为这一唯一的输出端口，并安装对应的流表项。如果的可能的输出端口有多个，控制器通过设定一个 group，其每一个可能的动作是向这些端口中的某一个输出，然后安装流表项使得数据包以等概率从这一 group 中选择一个动作。对于 fat tree 网络，在确定的源主机与目的主机之间，从接入层到聚合层以及从聚合层到核心层这一“向上”的路径上，输出端口的选择不唯一，而且所有向上的路径都是等价的；而从核心层到聚合层、聚合层到接入层以及从接入层到主机这一“向下”的路径则是唯一的。因此，对于 fat tree 网络，该模块预先在接入层与聚合层交换机上建立 group，该 group 中包含所有连接上一层交换机的端口。对于需要“向上”转发的流，初始时控制器让交换机向这一 group 输出，让这个流以等概率向上一层各个的交换机转发。

大象流探测这一模块如上一节所叙述，会在所有的接入层交换机安装流统计事件以探测大象流，并持续监控大象流的统计数据。大象流探测模块定义了 Ryu 控制器中的“大象流探测请求”事件，该事件中包含需要探测的交换机的 DPID。拓扑发现与转发路径计算模块通过发送这一事件要求大象流探测模块开启对某一边缘交换机的大象流探测。在我们的实现中，拓扑发现与转发路径计算模块在找到了 fat tree 网络中的所有链路之后，会向大象流探测模块发送探测请求，要求大象流探测模块在所有接入层交换机上安装流统计事件。对于大象流的界定标准，我们采用了 Hedera[51]中的标准，即当一个流表项在 1 秒内匹配的数据总字节数达到或超过 1Gbps 链路的 10%，即 12.5MB 时，该流被认为是大象流，事件就被触发，此时交换机上报所有该交换机上的大象流在这 1 秒内的统计数据。大象流探测模块收到交换机发来的流统计事件的事件报告时，根据事件报告中的各大象流的统计数据，来更新控制器中的大象流记录。

流调度模块按照固定的时间间隔定期运行，调度网络中的大象流。流调度模块从大象流记录中读取网络中存在的大象流，根据其统计数据的更新时间判断该大象流是否仍然活跃（如果其最近一次的统计数据更新时间距离当前时间超过了流统计事件规定的时间间隔的两倍，就认为这个大象流不再活跃）。流调度模块对所有活跃的大象流，估算每一个大象流所需要的带宽资源，估算方法如下：对于每一台主机，假设以它为源的活跃大象流有 n_1 个，以它为目的的活跃大象流有 n_2

个，该主机与接入层交换机之间的链路带宽为 B，那么以它为源和以它为目的的所有活跃大象流的带宽需求为

$$\min(\frac{1}{n_1}, \frac{1}{n_2}) \times B \quad (5\text{-}1)$$

这一估算方法基于如下假设：所有大象流会尽可能的占满所有带宽；同一链路上的所有大象流会平分带宽资源；大象流的传输速率只会被主机到接入层交换机的链路带宽限制，交换机之间的连接的带宽是无限的。基于这些假设，每个大象流的传输速率就会为式（5-1）中的值。然后，流调度模块根据所有活跃大象流的带宽需求，使用全局最先匹配算法为这些流安排新的转发路径，以最小化多个大象流经过同一链路带来的拥塞，并在交换机上安装新的流表项。

在流调度的整个应用中，为了减少交换机中流表项的数量，我们把源 IP 地址和目的 IP 地址均相同的所有数据包视为同一个流，并安装一条匹配源 IP 地址和目的 IP 地址的流表项来匹配这些数据包。例如，从 10.0.0.1:54321 到 10.0.0.2:12345 和从 10.0.0.1:22222 到 10.0.0.2:33333 的两个 TCP 连接上传输的数据包，会被视为同一个流，在所有的交换机上均只有一条源 IP 地址为 10.0.0.1，目的 IP 地址为 10.0.0.2 的流表项来匹配这属于两个 TCP 连接的数据包。

由于拓扑发现与转发路径计算模块中，为首次发现的流设置初始转发路径时用到了 OpenFlow1.1 以上才能支持的 group 功能，而 OpenEvent 的控制器端目前支持 OpenFlow1.0 与 OpenFlow1.3 这两个版本，流调度的应用需要的 OpenFlow 协议为 OpenFlow1.3。

这一流调度应用克服了传统网络中采用 ECMP 实现多路径转发不能根据网络状况动态调整的缺点，实现了根据网络状况动态地调整转发路径以优化网络带宽的利用。在大象流探测方面，该应用使用了 OpenEvent 中的流统计事件来探测并监控大象流，相比原先的轮询方法，该应用不但大大减少了探测大象流的网络开销，还能使得控制器更快地发现网络中的大象流，在更早的时间上对这些大象流安排路径，以优化调度效果。

5.4 本章小结

本章介绍了我们在 OpenEvent 事件框架下定义的两类事件的基础上，实现的网络监控与流调度方面的应用。在端口统计事件基础上，我们实现了基于事件机制的链路带宽利用状况监测；在流统计事件的基础上，我们实现了基于事件机制的大象流探测。在我们的大象流探测应用的基础上，我们实现了 fat tree 网络中的

流调度。

链路带宽利用状况监测通过在每条链路的一端安装端口统计事件，让交换机主动上报该端口接收与发送的数据包数与字节数，从而获取与此端口相连的链路上的带宽利用状况。相比基于轮询的传统方式，基于事件机制的监测方式可以显着地降低网络开销。

大象流探测的应用通过在与主机直接相连的边缘交换机上安装流统计事件，让这些边缘交换机主动上报大象流的统计数据，从而实现对大象流的探测与对其统计数据的持续监控。相比定期轮询以获取所有流表项的统计数据的传统方式，基于事件机制的探测方式不仅减少了大量的网络开销，还减少了控制器处理大量流表项的统计数据带来的处理上的开销。

Fat tree 网络中的流调度利用到了以上的大象流探测应用作为一个组成模块，另外还包括拓扑发现与转发链路计算和流调度两个模块。拓扑发现与转发链路计算负责探测与维护整个网络的拓扑，并为每一个流计算和设定初始的转发路径。大象流探测模块负责探测并维护大象流的统计信息。流调度模块负责为所有的大象流重新安排转发路径，以减少拥塞，增加网络带宽的利用率。这一流调度的应用相比 ECMP，能够动态地根据网络状况调整数据包的转发路径，更好地利用了网络带宽资源。其中的大象流探测是基于事件机制的，相比基于轮询方式的原有方法，基于事件机制的方法不仅减少了网络开销，还能够更快地发现网络中的大象流，更加及时地对这些大象流进行调度。

第六章 实验设计、结果与分析

为验证 OpenEvent 框架以及在这个框架下实现的，基于事件机制的网络状况监控与流调度的应用的效果，以及基于事件机制的方法对于 OpenFlow 中的原有方法的优势，我们进行了一系列的实验。对于上一章中的每一个应用，我们分别做了一组实验以验证这些应用对于 OpenFlow 中现有的这些应用的实现的优势。实验结果表明，基于事件机制的这些应用相比原有的实现，不仅大大减少了控制器与交换机之间的网络通信上的开销，还提升了统计数据的准确性和流调度的效果。

6.1 实验环境介绍

在所有的实验中，我们采用 Ryu 3.21[19]作为控制器端，承载我们的 OpenEvent 框架和相关的控制器端应用；源代码经过修改并重新编译的 Open vSwitch 2.3.90[9] 版作为交换机端。所有实验在 Mininet 构建的虚拟网络环境下运行，Mininet 的版本为 2.21[60]。在 Mininet 构建的虚拟环境下，每一台交换机是 Open vSwitch 维护的一个 ofproto 结构体；每一台主机是拥有自己的网络设备命名空间的一个 shell。连接各交换机与主机的链路是由一对虚拟以太网设备来模拟的，一对虚拟以太网设备中，每一个设备只能把数据包发送到它的同伴的接收端。控制器与交换机的连接通过本地环回端口（lo）。实验中使用的 Ryu 控制器、Open vSwitch 与 Mininet 均为进行实验时的最新版本。所有的主机、交换机与控制器均运行在一台安装 Ubuntu 14.04 操作系统，CPU 为英特尔 i5-2400 四核处理器，内存大小为 4GB 的 PC 机上。

所有实验采用的网络拓扑均为交换机端口数为 4 的 fat tree 网络。该网络中含有 16 台主机，8 台接入层交换机，8 台聚合层交换机，4 台核心层交换机，共 20 台交换机。每台主机到连接同一接入层交换机的另一主机只有唯一最短路径，到同一 pod 不同接入层交换机上的主机有 2 条不同的最短路径，到不同 pod 的主机上有 4 条不同的最短路径。除另有说明外，主机与交换机的之间的链路带宽以及交换机之间的链路带宽均设定为 1000Mbps，不另外设置延迟。

在我们的实验中，每一台主机通过 TCP 连接向一台或多台特定的主机持续地发送数据。每一台主机在起始时间相同的 180 秒内向不同的一台或者多台主机不断通过 TCP 连接发送数据，然后维持 20 秒空闲。在持续发送数据的 180 秒内，主机按照平均 1 毫秒的时间间隔建立新的连接，这个时间间隔服从指数分布。在

所有的连接中，有 1%的连接会成为大象流，另外的所有连接都是老鼠流。所有大象流传输的数据量的平均值为 1GB，所有老鼠流传输的数据量的平均值为 1MB。所有大象流与老鼠流传输的数据量都服从指数分布，当一个连接传输的数据量达到了事先设定的值时，该连接终止，以平均 1 毫秒的，服从指数分布的时间间隔发起新的连接。当时间到达 180 秒时，所有连接终止。网络中每一台主机依次编号为 0~15。在实验中，我们按照不同的流量模式，为每一台主机分配传输数据的目标主机（一台或多台）。这些流量模式包括：

Stride(n)：编号为 i 的主机向编号为$(i+n)$ mod 16 的主机传输数据。

Random(n)：每台主机向随机选出的 n 台主机传输数据。

Same-pod：每台主机向同一 pod 中，但不在同一台接入层交换机的另一台主机传输数据。

6.2 大象流探测的相关实验设计、结果与分析

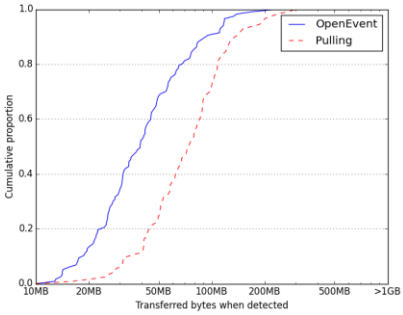
为检验基于事件机制的大象流探测的应用的性能，我们进行了一系列的实验，对比了基于事件机制的大象流探测与基于轮询的现有大象流探测方法在探测的及时性，与探测带来的控制器与交换机之间的网络通信开销这两方面的表现。

在这一组实验中，具有不同的（源 IP 地址，目的 IP 地址，传输层协议，源传输层端口号，目的传输层端口号）五元组的数据包被归类为不同的流，也就是每一个 TCP 连接的两个方向的数据包分别视为一个流。交换机中所有的流表项都精确匹配如上所述的五元组，控制器对于每一个由于匹配失败通过 packet in 转发到控制器的数据包，在交换机上安装一条精确匹配其上述五元组的流表项。实验中，调度应用不会启动，以防止由于调度而重新安排大象流的路径造成交换机上流表的变动。

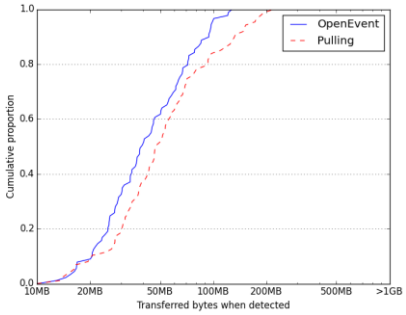
实验中，我们采用的大象流界定标准如下：如果一个流在 1 秒内的数据传输速率达到链路带宽的 10%，该流就被视为大象流。在我们的实验环境下，链路带宽为 1000Mbps，因此在 1 秒内传输的字节数大于等于 12.5MB（即传输速率大于等于 100Mbps）的流被视为大象流。

我们在数种不同的流量模式下进行了实验。对于每一种流量模式，我们重复进行了 5 次实验，每次实验中，基于事件机制的大象流探测方法与基于现有 OpenFlow 的轮询机制的探测方法同时运行，探测并监控同一个网络中的大象流。基于事件机制的探测方法在所有的接入层交换机上安装一个流统计事件，该事件要求交换机每 1 秒主动上报在这 1 秒内匹配数据包的字节数大于等于 12.5MB 的，

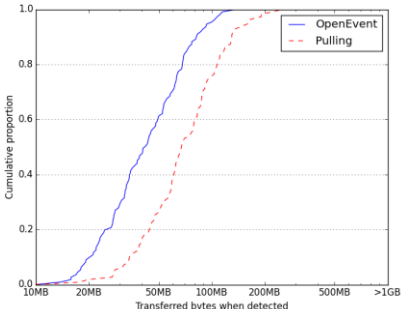
且其网络层协议为 IPv4（0x0800）的流表项的统计数据。若没有满足条件的流表项，交换机不会通过这一事件上报。基于轮询机制的探测方法每 1 秒向接入层交换机发送一条读取流表项统计数据的请求，要求交换机在回复中报告所有网络层协议为 IPv4 的流表项的统计数据，控制器通过计算找出流表中对应于大象流的流表项。我们通过大象流被发现时已经传输的数据量来比较两种探测方法的准确性与及时性，其结果如图 6-1 所示；通过为大象流探测在控制器与交换机之间传输的消息的数据包数与字节数来对比两种方法的网络开销，结果如图 6-2 所示。



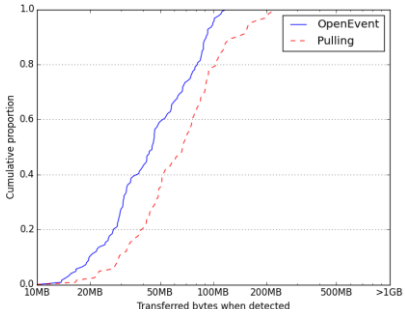
a) 流量模式 Random(2)下的发现时已传输字节数的累积分布图
a) CDF of bytes transferred when detected under traffic pattern Random (2)



b) 流量模式 Random(4)下的发现时已传输字节数的累积分布图
b) CDF of bytes transferred when detected under traffic pattern Random (4)



c) 流量模式 Stride(4)下的发现时已传输字节数的累积分布图
c) CDF of bytes transferred when detected under traffic pattern Stride (4)



d) 流量模式 Stride(8)下的发现时已传输字节数的累积分布图
d) CDF of bytes transferred when detected under traffic pattern Stride (8)

图 6-1 不同流量模式下大象流发现时已传输字节数的累积分布
Fig.6-1 Cumulative proportion of bytes already transferred when detected of elephant flows under different traffic patterns

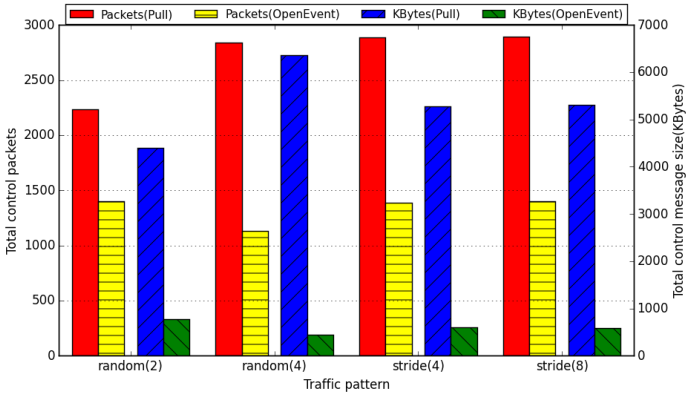


图 6-2 两种方法中大象流探测中的网络开销

Fig. 6-2 Network consumption of elephant detection by two methods

图 6-1(a)~(d)展示了通过两种方法探测到大象流时，这些流已经传输的字节数的累积分布。累积分布曲线上的点(*x*,*y*)表示有占全部被发现大象流比例为 *y* 的流，在已经传输的字节数小于等于 *x* 时被探测到了。因此，该曲线越偏左侧，说明该方法能够越及时地探测到网络中的大象流。从图 6-1 中，我们可以看出，每一种流量模式下，使用 OpenEvent，基于事件机制的大象流探测方法对应的曲线相比基于轮询方式的现有方法对应的曲线明显偏向左侧，这说明基于事件机制的探测方法在发现大象流的及时性上明显优于现有的基于轮询的方法。在同一已发现比例的水平线上，我们的基于事件机制的探测方法对应的已传输数据量大约为基于轮询的现有方法的一半。在更早的时间上发现大象流，有利于更加及时地对这些大象流进行调度，优化网络的整体性能。实验中，我们还发现对于同样的网络，同时运行的两种探测方法，基于轮询的方法探测到的大象流的数量少于基于事件机制探测到的大象流的数量。这说明基于轮询的方法可能由于某些原因漏掉了小部分的大象流。

两种探测方法在不同流量模式下，在控制器与交换机的连接上的网络开销如图 6-2 所示。该图的每种流量模式下的四项数据依次为基于轮询的方法和基于事件机制的方法传输的数据包数量，以及基于轮询的方法和基于事件机制的方法传输的数据总量，单位为 KB。从传输的数据包数量的对比来看，基于事件机制的方法减少了 50%的数据包数量。这是由于基于事件机制的方法中，交换机主动上报大象流的统计数据，而无需像在基于轮询的方法中一样由控制器发出请求，然后交换机回复。这样，占总数据包数量一半的请求就可以节省下来。从传输的数据总量来看，基于事件机制的探测方法相比基于轮询的现有方法减少了 82%~93%的数据量。这表明，使用基于事件机制的方法来探测大象流，可以大大减少控制器

与交换机之间的网络开销。在实验中，我们还发现通过轮询机制收集到的流表项中，只有 5%~15%的流表项满足大象流的界定标准。这个比例与两种方法中传输的数据总量的比例相符。这说明，通过安装流统计事件，控制器可以要求交换机仅上报所有大象流的统计数据，而忽略老鼠流。基于事件机制的方法中，交换机仅上报大象流的统计数据不仅减少了网络开销，也减少了控制器处理流表项的统计数据的处理开销。

综合以上的结果，基于事件机制的大象流探测方法具有更强的及时性，能更早地发现网络中的大象流，而且可以非常显著地减少控制器与交换机之间的网络开销与控制器的处理开销。对于交换机来说，不论是通过控制器定时轮询还是自身定时检查事件是否触发，处理的开销基本相同。因此，基于事件机制的大象流探测方法在性能和开销上全面优于基于轮询的现有方法。

6.3 Fat tree 网络中流调度的相关实验设计、结果与分析

这一节我们将展示利用了基于事件机制的大象流探测的流调度应用在 fat tree 网络中的调度效果。作为对比，我们在同一流量模式下对比了以下四种调度方法达成的网络总体带宽：(1) 静态 ECMP 路由：当一个流到达其目的地有多条路径时，随机选择一条固定的路径，以作为基准；(2) 非阻塞网络：所有交换机之间的连接的带宽不受限制，主机到接入层交换机的连接带宽仍然为 1000Mbps，网络整体带宽仅受主机与接入层交换机的连接的带宽限制，路由仍然使用静态 ECMP 决定，以作为理想情况；(3) 使用我们的流调度应用进行调度，以检测该调度应用的效果与可行性；(4) 用基于轮询的大象流探测方法来代替流调度的应用中的大象流探测部分，以比较基于事件机制的大象流探测与基于轮询的大象流探测对调度效果的影响。对于未经调度的流，其转发路径仍然由 ECMP 决定。在实验中，ECMP 实际上利用 OpenFlow 的 group 功能实现，我们通过在一个 select 类型的 group 中，为 ECMP 中的每一个可能的输出端口设定一个相等的权重来实现其功能。在我们的调度应用中，流调度模块使用全局最先匹配算法为大象流重新安排路径。

我们在多种流量模式下进行了实验。对于每种流量模式，我们使用前文中(1)~(4)的路由与调度方法，运行 5 次实验，通过每台主机在传输数据的 180 秒内传输的数据量的总和来计算网络的总体带宽，并取 5 次总体带宽的平均值。各种路由与调度方法下的网络总体带宽如图 6-3 所示。每一种流量模式下从左到右依次是在 ECMP 下、使用基于轮询的大象流探测方法的调度下、使用基于事件机制的大象流探测方法的调度下，以及非阻塞网络下的网络总带宽。

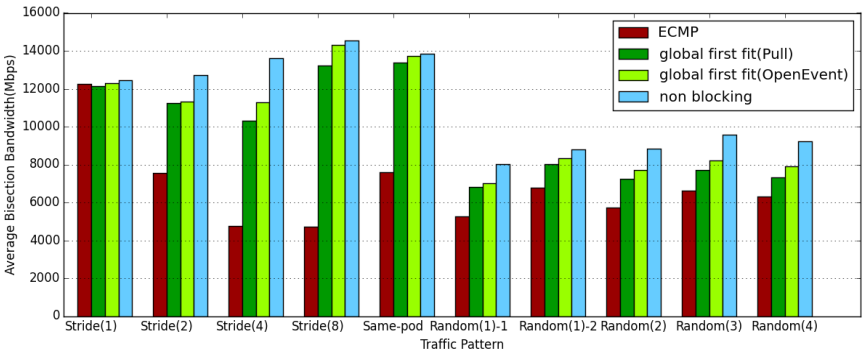


图 6-3 各种流量模式下不同路由与调度方法达成的网络总带宽

Fig.6-3 Total bandwidth achieved by different routing and scheduling methods under different traffic patterns

在图 6-3 中，我们可以看出流调度应用在提高网络整体带宽上的显著效果。经过我们的流调度应用的调度，网络整体带宽在最好的情况下达到了 ECMP 下网络整体带宽的 3 倍。与非阻塞的理想情况相比，经过调度达成的总带宽已经达到了理想情况的至多 99%，即使在效果最差的一组实验中，调度后的网络总带宽也达到了理想情况下网络总带宽的 83%。

通过在图 6-3 中使用同一调度算法，但使用的大象流探测方法不同的两种情况的对比，我们可以看到，我们的使用基于事件机制大象流探测方法的应用相比使用基于轮询的方法探测大象流的方法，在最终达成的调度效果上有些许的提升，在调度后的网络总带宽上提升了约 10%。结合上一章的实验结果说明，我们实现的基于事件机制的大象流探测的方法可以有效地运用在流调度中，并且能够提升流调度的效果，同时减少网络开销。

通过进一步地对比图 6-3 中不同流量模式下各种路由与调度算法在网络整体带宽上的表现，可以看出不同接入层交换机之间与不同 pod 之间的流量所占比例越高，ECMP 的表现越差。在 stride(1)这一流量模式下，50%的流量在同一接入层交换机上，此时 ECMP 的表现接近理想情况。在 stride(2)与 same-pod 两种流量模式下，大部分或全部的流量在同一 pod 的不同接入层交换机之间，ECMP 下的网络总带宽仅有理想情况的 60%，而经过我们的调度应用调度后的网络总带宽仍然能达到理想状况的 90%；在流量模式 stride(4)与 stride(8)中，所有流量均在不同 pod 之间，这时 ECMP 下的网络总带宽下降到了理想状况的 40%以下。这说明静态的 ECMP 路由并不能准确地将网络中的流量平衡地分配到各条路径上；经过的 ECMP 选择次数越多，这种不平衡越明显，由流量在各条路径上分配不均带来的总体带宽下降也越严重。因此，在数据中心中，根据网络的当前状态进行动态的

流调度也是必需的。

6.4 带宽利用状况监测的相关实验设计、结果与分析

这一节将对比基于事件机制的带宽利用状况监测与基于轮询的带宽利用状况监测方法在准确性与网络开销上的对比实验结果。在实验中，两种方法均会对交换机与交换机之间的所有链路的带宽利用状况进行监测。基于事件机制的监测方法对每一条交换机之间的链路，在 DPID 较小的交换机一端上为对应的端口安装一个事件，要求交换机在每 1 秒内，如果这个端口发送或接收数据的速率达到或超过最大带宽的 1%，即发送或接收的数据量在这 1 秒内达到或超过 1.25MB 时触发事件，上报端口的统计数据。基于轮询的监测方法对于每一条交换机之间的链路，每 1 秒向 DPID 较小的交换机一端发送一条读取对应端口的统计数据的请求，让交换机通过回复上报统计数据。

我们在几种不同的流量模式下分别重复进行了 5 次实验，每次实验中，两种监测方法同时运行，监控同一网络中所有交换机之间的链路的带宽利用状况。我们对比了一次实验中在 stride(8)的流量模式下，两种方法测得的从某一台聚合层交换机到一台核心层交换机的链路的带宽利用状况，以对比两种方法的准确性，结果如图 6-4 所示；图 6-5 中，两种方法在不同流量模式下进行监测，在实验进行的 200 秒内所传输的数据包数量与数据总字节数的对比说明了基于事件机制的方法在减少网络开销上的效果。

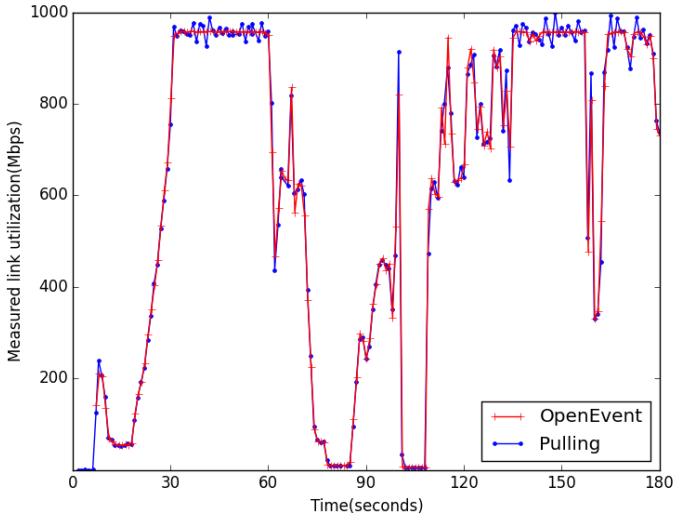


图 6-4 两种方法测得的带宽利用状况

Fig.6-4 Measured link utilization by the two methods

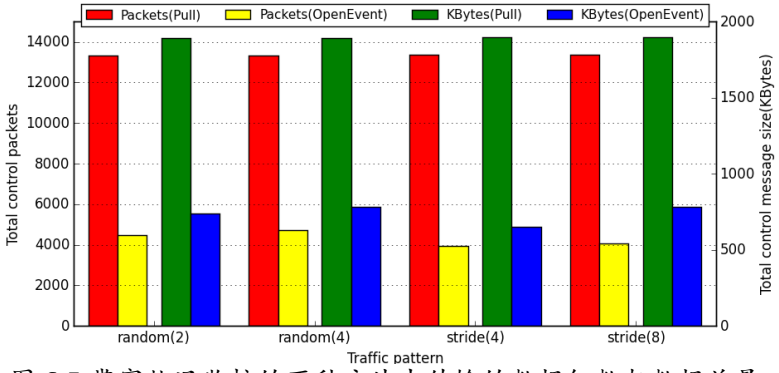


图 6-5 带宽状况监控的两种方法中传输的数据包数与数据总量

Fig.6-5 Number of packets and total amount of data transferred for link utilization monitoring by the two methods

图 6-4 中，两条线接近重合，这说明通过基于事件机制的方法与基于轮询的方法测得的带宽利用状况差别不大。虽然两种方法测得的链路上的数据传输速率的数值上稍有差异，其相差在 10%以内，这说明基于事件机制测得的带宽利用状况的精确度是可以保证的。图中，我们可以看到基于轮询的方法测得的数据传输速率的数值波动较大。据此，我们认为，基于事件机制的方法测得的数值更为精确。因为两种方法测得的带宽利用状况都来自交换机的端口统计数据，其计算方法如下：对于某一端口，从该端口出发的链路的数据传输速率由两个不同时刻测得的该端口发送的字节数计算得出。若在时刻 t_0 测得该端口已传输 B_0 字节的数据，而在时刻 t 测得该端口已传输 B 字节的数据（$B>B_0,t>t_0$），则测得的 t_0 到 t 这段时间内，该链路的数据传输速率为

$$\frac{B - B_0}{t - t_0} \qquad (6\text{-}1)$$

在基于轮询的方法中，B 与 B_0 来自从交换机读取的统计数据，而 t 与 t_0 则来自控制器收到统计数据的时刻，两者来源不一致，误差较大；而在基于事件机制的方法中所有数据都由交换机测得，误差的来源较少。尤其当控制器与交换机，或者控制器与交换机之间的网络比较繁忙时，基于轮询的方法的误差更大。

图 6-5 中表明，使用基于事件机制的监测方法相比基于轮询的方法可以减少 66%~70%的数据包数量与 56%~65%的传输数据总量。理论上来说，用事件机制代替轮询来进行带宽利用状况监测可以减少控制器下发的请求的那一部分，占数据包数一半的数据；从实验中我们可以看出，如果设定一定阈值，让交换机仅在链路的数据传输速率足够大时上报统计数据，可以进一步减少网络开销。

6.5 本章小结

我们为验证基于 OpenEvent 事件框架与框架内定义的事件实现的网络状况监控与流调度的应用的性能，在 Mininet 搭建的虚拟 OpenFlow 网络环境下进行了一系列的实验。实验结果表明，我们实现的基于事件机制的网络状况监控的应用在准确性与及时性方面优于基于轮询的现有方法，网络开销方面显著少于基于轮询的现有方法。在流调度上，我们的应用能够有效地支持流调度功能，明显提高网络的总体带宽利用率。

所有的实验都在 Mininet 虚拟的交换机端口数为 4 的 fat tree 网络中进行，网络中共有 16 台主机。在每一次实验中，每台主机向其他特定的一台或多台主机通过 TCP 连接传输数据，这些连接中，有约 1%的连接会成为大象流。

在大象流探测方面，我们通过在同样的网络上运行基于事件机制与基于轮询两种不同的探测方法来探测大象流，以对比这两种方法的性能差异。实验结果表明，我们的基于事件机制的探测方法相比现有的基于轮询的方法，能够更早地发现网络中的大象流，并减少 80%以上的网络开销。

在流调度方面，我们在多种不同的流量模式下对比了以下路由与调度方法下的网络总带宽，以验证我们的流调度应用的有效性：静态 ECMP，作为基准；非阻塞网络，作为理想情况；采用基于事件机制的大象流探测方法的流调度；采用基于轮询的大象流探测方法的流调度。实验结果表明，ECMP 并不能有效地利用网络带宽，尤其在不同 pod 间流量大时，ECMP 效果较差；使用我们的流调度应用以调度大象流，可以显著提升网络的总体带宽，能够达到理想状况的 83%~99%，而且使用基于事件机制的探测大象流方法的调度能够达到更好的效果。

在带宽利用状况监测上，我们在各种流量模式下，运行基于事件机制与基于轮询的两种方法以监测交换机之间的所有链路的数据传输速率，通过比较测得的链路数据传输速率来比较两种方法的准确性，以及两种方法在控制器与交换机之间传输的数据量来比较网络开销。从多次实验的结果可以看出，两种方法测得的链路数据传输速率差别不大，并且基于事件机制的方法带来的网络开销明显较少。并且，我们有理由相信基于事件机制测得的链路数据传输速率更为接近其真实值。

综合以上的实验结果，OpenEvent 事件框架以及该框架内定义的事件能够有效地应用在网络状况监控与流调度上，并相对现有的方法显著提升性能与降低网络开销。

第七章 结束语

7.1 主要工作与创新点

本文提出了 OpenEvent 这一事件框架作为 OpenFlow 的一个扩展，以允许控制器让交换机在一定条件下主动上报自身的统计数据，满足网络状况监控与流调度的需要。在 OpenEvent 框架下可以定义各种类型的事件以要求交换机上报不同类型的统计数据。我们定义了端口统计事件以让交换机上报端口的发送与接收的数据量等统计数据，以及流统计事件以让交换机上报流表项的匹配数据量等统计数据。

OpenEvent 的实现包括通信协议、控制器端与交换机端三个部分。通信协议部分，OpenEvent 中所有关于事件的消息都定义为 OpenFlow 自定义类型消息，分为事件请求、事件请求回复、事件报告三大类。事件请求用于控制器向交换机发送请求，以添加、修改或删除交换机上安装的事件；事件请求回复用于交换机向控制器回复请求执行的结果；事件报告用于交换机在事件被触发时向控制器上报自身的统计数据。控制器端在 Ryu 控制器[19]提供的框架下实现，主要包括对通信协议中各类消息的编码与解码，以及给上层的网络管理方面的应用提供的接口。交换机端包括以下几个模块：编码与解码模块，用于对通信协议中的各类消息进行编码与解码，正确地接收和发送 OpenEvent 中的消息；请求处理模块，用于处理控制器发来的请求；事件列表维护模块，用于维护自身已安装的事件；事件监测与上报模块，用于检查自身已安装的事件是否被触发以及在触发时上报所需统计数据。我们在 Open vSwitch[9]源代码的基础上做出了一些改动，集成了如上所述的几个模块，实现了交换机端。

在 OpenEvent 框架以及该框架下定义的两类事件的基础上，我们实现了在 OpenFlow 网络中，基于事件机制的进行网络状况监控与流调度的控制器端应用。在端口统计事件的基础上，我们实现了监测链路带宽利用状况的应用，该应用通过在被监测链路的一个端口所属的交换机上安装对应端口的端口统计事件，要求该交换机上报该端口发送与接收的数据包数量与字节数，以获取对应链路上的数据传输速率。在流统计事件的基础上，我们实现了大象流探测的应用。该应用通过在与主机直接相连的边缘交换机上安装流统计事件，要求交换机主动上报大象流的统计数据。在大象流探测的基础上，我们实现了在 fat tree 网络中进行动态流调度的应用。

为验证以上基于事件机制的应用对于原有的基于轮询的方法的优势，我们在Mininet 模拟的 OpenFlow 网络环境下进行了一系列的对比实验。所有的对比实验结果表明，基于事件机制的网络监控方法不仅能有效地降低控制器与交换机之间的网络开销，还提高了监控的准确性与及时性，能够有效地应用在流调度上。在大象流探测方面，基于事件机制的方法能够更早地发现网络中的大象流，并且相比基于轮询的原有方法，减少了 83%~92%的网络开销。使用了基于事件机制的方法探测大象流的流调度应用有效地提升了 fat tree 网络中的网络总体带宽，达到了理想情况的 80%以上甚至 99%，显著优于 ECMP，并且比较使用基于轮询的方法探测大象流的调度方法，取得了更好的调度效果。在链路带宽利用状况监测方面，基于事件机制的监测方法测得的结果与基于轮询的方法测得的结果差别较小，并且，基于事件机制的监测方法减少了一半以上的网络开销。这些实验的结果表明，OpenEvent 框架与该框架下定义的事件能够有效地应用在网络状况监控与流调度上。基于事件机制的方法与基于轮询的原有方法相比，在监控的及时性、准确性上更优，而且能够显著地降低网络开销。

本文的主要创新点在于，提出并实现了具有通用性、可扩展性，与 OpenFlow 兼容的 OpenEvent 事件框架，以支持交换机通过事件机制主动上报各类统计数据。以往的 OpenFlow 网络中，虽然也存在 FlowSense[44]等使用 OpenFlow 现有的异步消息来收集统计数据的方法，但这些方法局限性较大，不能有效地收集网络状况监控与流调度所需要的统计数据。对比实验的结果证明，我们实现的 OpenEvent 框架，以及在这个框架下定义并实现的两类事件能够有效地应用在 OpenFlow 网络中的网络状况监控与流调度上，有效地收集应用中需要地各类统计数据，并且在统计数据的及时性、准确性和网络开销方面都优于现有的基于轮询的方法。

7.2 后续研究

OpenEvent 这一框架以及该框架下的事件，以及它们在 OpenFlow 网络中的应用仍然存在很多值得进行后续研究的方面。

在 OpenEvent 框架本身的方面，在 Open vSwitch 以外的其他软硬件交换机上如何高效地实现交换机端的功能仍然是值得研究的问题。随着事件种类的增加，在交换机上维护并检测事件带来的开销会如何变化，以及这一开销相对于统计数据的准确性提升与网络资源消耗的减少而言，代价是否合适也需要进一步的研究。在 OpenEvent 框架下定义的事件方面，如何定义并实现其他类型的事件以有效地从交换机上收集网络状况监控与流调度所需要的各种统计数据，并且尽量减少包

括交换机与控制器在内的处理开销，也是需要进行后续研究的一个方面。

在 OpenEvent 的应用方面，除了我们已经实现的链路带宽利用状况监测与大象流探测两种应用之外，还有更多基于事件机制的网络状况监控方面的应用有可能在这个基础上实现。

7.3 本章小结

本章总结了我们在本文中介绍的主要工作与这些工作的创新之处，以及在本文基础上可能的后续研究的方面。

本文的主要介绍的工作是我们在 OpenFlow 上实现的 OpenEvent 这一事件框架，以及在此框架下实现的基于事件机制的，进行网络状况监控与流调度的应用。OpenEvent 可以支持控制器要求交换机主动上报统计数据的各类事件。实验表明，在 OpenEvent 框架下实现的基于事件机制的网络状况监控方法在监控的准确性与及时性，以及网络开销方面均优于基于轮询的现有方法。本文的主要创新之处在于提出并实现了可以在其基础上定义各类事件的，通用、可扩展的事件框架。在这个框架的基础上，我们提出了框架本身的实现方面与框架的应用方面的可能的后续研究的一些方向。

参 考 文 献

[1] Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010: 267-280.

[2] Kandula S, Sengupta S, Greenberg A, et al. The nature of data center traffic: measurements & analysis In Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference. ACM, 2009: 202-208.

[3] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks[J]. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74.

[4] Greene K. TR10: Software-defined networking[J]. Technology Review (MIT), 2009.

[5] Nadeau T D, Gray K. SDN: software defined networks. USA, O'Reilly, 2013,7-8

[6] Open Networking Foundation. OpenFlow switch specification version 1.0.0[EB/OL].[2015-12-25]. . https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf.

[7] Open Networking Foundation. Software-defined networking: the new norm of networks[EB/OL].[2015-12-25]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[8] Internet Assigned Numbers Authority(IANA). Service name and transport protocol port rumber registry[EB/OL].[2015-12-25]. http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[9] Open vSwitch. Production Quality, Multilayer Open Virtual Switch[EB/OL]. [2015-12-25]. http:// openvswitch.org/.

[10] Project floodlight. Indigo virtual switch[EB/OL]. [2015-12-25]. http://www.projectfloodlight.org/indigo-virtual-switch/

[11] Cpqd group. OpenFlow software switch[EB/OL]. [2015-12-25]. http://cpqd.github.io/ofsoftswitch13/.

[12] Naous J, Erickson D, Covington G A, et al. Implementing an OpenFlow switch on the NetFPGA platform In Proceedings of the 4th ACM/IEEE Symposium on

Architectures for Networking and Communications Systems. ACM, 2008: 1-9.
[13] Cisco Systems, Inc. Release Notes for the Catalyst 4500E Series Switch, Cisco IOS XE 3.7xE[EB/OL].[2015-12-25]. http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst4500/release/note/ol-37xe-4500e.html.
[14] Pica8, Inc. Product reference guide[EB/OL].[2015-12-25]. http://www.pica8.com/wp-content/themes/twelve-theme/documents/pica8-datasheet-64x10gbe-p3922-p3930.pdf.
[15] Gude N, Koponen T, Pettit J, et al. NOX: towards an operating system for networks[J]. ACM SIGCOMM Computer Communication Review, 2008, 38(3): 105-110.
[16] NOX Repo. POX About POX[EB/OL].[2015-12-25]: http://www.noxrepo.org/pox/about-pox/.
[17] Erickson D. The beacon openflow controller In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013: 13-18.
[18] Project floodlight. Floodlight OpenFlow controller[EB/OL].[2015-12-25]. http://www.projectfloodlight.org/floodlight/.
[19] Ryu SDN framework community. Ryu SDN framework[EB/OL].[2015-12-25]. http://osrg.github.io/ryu/.
[20] IBM. IBM programmable network controller 3.0[EB/OL].[2015-12-25]. http://www-01.ibm.com/common/ssi/rep_ca/0/897/ENUS212-310/ENUS212-310.pdf.
[21] NEC Corporation of America. ProgrammableFlow PF6800 Controller [EB/OL].[2015-12-25]. http://www.necam.com/sdn/doc.cfm?t=PFlowController
[22] Tootoonchian A, Ganjali Y. HyperFlow: A distributed control plane for OpenFlow In Proceedings of the 2010 internet network management conference on Research on enterprise networking. USENIX Association, 2010: 3-3.
[23] Sherwood R, Gibb G, Yap K K, et al. Flowvisor: A network virtualization layer[J]. OpenFlow Switch Consortium, Tech. Rep, 2009.
[24] Drutskoy D, Keller E, Rexford J. Scalable network virtualization in software-defined networks[J]. Internet Computing, IEEE, 2013, 17(2): 20-27.
[25] Koponen T, Casado M, Gude N, et al. Onix: A Distributed Control Platform for Large-scale Production Networks In OSDI. 2010, 10: 1-6.
[26] Mattos D M F, Fernandes N C, Da Costa V T, et al. Omni: Openflow management infrastructure In Network of the Future (NOF), 2011 International Conference on

the. IEEE, 2011: 52-56.

[27] Project Floodlight. OFTest—Validating Openflow Switches[EB/OL].[2015-12-25]. http://www.projectfloodlight.org/oftest.

[28] Rotsos C, Sarrar N, Uhlig S, et al. OFLOPS: An open framework for OpenFlow switch evaluation In Passive and Active Measurement. Springer Berlin Heidelberg, 2012: 85-95.

[29] Lantz B, Heller B, McKeown N. A network in a laptop: rapid prototyping for software-defined networks In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010: 19.

[30] Wette P, Draxler M, Schwabe A, et al. Maxinet: Distributed emulation of software-defined networks In Networking Conference, 2014 IFIP. IEEE, 2014: 1-9.

[31] Handigol N, Heller B, Jeyakumar V, et al. Reproducible network experiments using container-based emulation In Proceedings of the 8th international conference on Emerging networking experiments and technologies. ACM, 2012: 253-264.

[32] Wang S Y, Chou C L, Yang C M. EstiNet openflow network simulator and emulator[J]. Communications Magazine, IEEE, 2013, 51(9): 110-117.

[33] Jain S, Kumar A, Mandal S, et al. B4: Experience with a globally-deployed software defined WAN In ACM SIGCOMM Computer Communication Review. ACM, 2013, 43(4): 3-14.

[34] Hong C Y, Kandula S, Mahajan R, et al. Achieving high utilization with software-driven WAN In ACM SIGCOMM Computer Communication Review. ACM, 2013, 43(4): 15-26.

[35] Vissicchio S, Tilmans O, Vanbever L, et al. Central control over distributed routing In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. ACM, 2015: 43-56.

[36] Handigol N, Seetharaman S, Flajslik M, et al. Plug-n-Serve: Load-balancing web traffic using OpenFlow[J]. ACM SIGCOMM Demo,ACM, 2009, 4(5): 6.

[37] Wang R, Butnariu D, Rexford J. OpenFlow-based server load balancing gone wild In Proceedings of USENIX HotICE, 2011. USENIX, 2011:21-26.

[38] Cisco Systems, Inc. OpFlex: An Open Policy Protocol[EB/OL].[2015-12-25]. http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.pdf.

[39] Tootoonchian A, Ghobadi M, Ganjali Y. OpenTM: traffic matrix estimator for OpenFlow networks In Passive and active measurement. Springer Berlin

Heidelberg, 2010: 201-210.

[40] Phemius K, Bouet M. Monitoring latency with openflow In Network and Service Management (CNSM), 2013 9th International Conference on. IEEE, 2013: 122-125.

[41] Chowdhury S R, Bari M F, Ahmed R, et al. Payless: A low cost network monitoring framework for software defined networks In Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014: 1-9.

[42] Van Adrichem N L M, Doerr C, Kuipers F. Opennetmon: Network monitoring in openflow software-defined networks In Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014: 1-8.

[43] Lin C Y, Chen C, Chang J W, et al. Elephant flow detection in datacenters using OpenFlow-based hierarchical statistics pulling In Global Communications Conference (GLOBECOM), 2014 IEEE. IEEE, 2014: 2264-2269.

[44] Yu C, Lumezanu C, Zhang Y, et al. Flowsense: Monitoring network utilization with zero measurement cost In Passive and Active Measurement. Springer Berlin Heidelberg, 2013: 31-41.

[45] Open Networking Foundation. OpenFlow switch specification version 1.5.0[EB/OL].[2015-12-25]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[46] Yu M, Jose L, Miao R. Software Defined Traffic Measurement with OpenSketch In NSDI,2013. USENIX. 2013, 13: 29-42.

[47] Curtis A R, Mogul J C, Tourrilhes J, et al. DevoFlow: Scaling flow management for high-performance networks In ACM SIGCOMM Computer Communication Review. ACM, 2011, 41(4): 254-265.

[48] Rasley J, Stephens B, Dixon C, et al. Planck: millisecond-scale monitoring and control for commodity networks In Proceedings of the 2014 ACM conference on SIGCOMM. ACM, 2014: 407-418.

[49] Suh J, Kwon T T, Dixon C, et al. OpenSample: A low-latency, sampling-based measurement platform for commodity In Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on. IEEE, 2014: 228-237.

[50] sFlow. About sFlow[EB/OL].[2015-12-25]. http://www.sflow.org/about/index.php.

[51] Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera: Dynamic Flow Scheduling for Data Center Networks In NSDI. 2010, 10: 19-19.

[52] Curtis A R, Kim W, Yalagandula P. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection In INFOCOM, 2011

Proceedings IEEE. IEEE, 2011: 1629-1637.

[53] Benson T, Anand A, Akella A, et al. MicroTE: Fine grained traffic engineering for data centers In Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies. ACM, 2011: 8.

[54] Trestian R, Muntean G M, Katrinis K. MiceTrap: Scalable traffic engineering of datacenter mice flows using OpenFlow In Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on. IEEE, 2013: 904-907.

[55] Li Z, Shen Y, Yao B, et al. OFScheduler: a dynamic network optimizer for MapReduce in heterogeneous cluster[J]. International Journal of Parallel Programming, 2015, 43(3): 472-488.

[56] Guo Y, Wang Z, Yin X, et al. Traffic Engineering in SDN/OSPF Hybrid Network In Network Protocols (ICNP), 2014 IEEE 22nd International Conference on. IEEE, 2014: 563-568.

[57] Open Networking Foundation. OpenFlow notification framework version OpenFlow management[EB/OL].[2015-12-25]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-notifications-framework-1.0.pdf.

[58] NetSNMP. Simple Network Management Protocol[EB/OL].[2015-12-25]. http://www.net-snmp.org/.

[59] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture[J]. ACM SIGCOMM Computer Communication Review, 2008, 38(4): 63-74.

[60] Mininet Team. Mininet an instant virtual network on your laptop [EB/OL].[2015-12-25]. http://mininet.org/.

致　谢

本文的完成首先需要感谢我的导师沈耀老师。沈耀老师带领我进入了软件定义网络与 OpenFlow 的研究这个领域，在我攻读硕士研究生的期间对我的学业与科研十分关心，并给予了悉心的指导。在文献的查阅、论文的选题与实验的设计上，沈老师给予了我充分的建议、指导和帮助，让我能够顺利进行这个课题的研究工作，完成论文。其次，我要感谢上海交通大学与嵌入与普适计算研究中心实验室给我提供了优良的科研环境，让我有机会阅读领域内最前沿的文献，在 OpenFlow 网络中进行与课题相关的一系列实验。我还要感谢实验室的张绍宇同学与李炯炯同学，在课题研究的过程中，他们曾经和我一起合作，帮助我解决了一些问题，克服了一些难点。此外，实验室的曾鹏程学长、郑利明学长、刘敛学长和黎寒宇同学在我攻读硕士期间的学习、科研与生活上也给予了各个方面的帮助，在这里一并表示感谢。

攻读硕士学位期间已发表或录用的论文

[1]（第一作者） Yi T, Li H. Flow-split: an approach to reduce flow establish time and invoking of controller in OpenFlow networks. In 2016 IEEE information technology, networking, electronics and automation control conference （IEEE ITNEC2016，已录用）