

学校代码：10126  
分类号： TP393.01

学号： 31409033  
编号：                     

# 论文题目

## 基于 SDN 的可扩展转发设备架构设计及关键 技术实现

学    院： 计算机学院  
专    业： 软件工程  
研究方向： 软件服务计算与测试  
姓    名： 胡新磊  
指导教师： 李华教授

2017 年 5 月 3 日



## 原创性声明

本人声明：所呈交的学位论文是本人在导师的指导下进行的研究工作及取得的研究成果。除本文已经注明引用的内容外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得内蒙古大学及其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：\_\_\_\_\_ 指导教师签名：\_\_\_\_\_

日 期：\_\_\_\_\_ 日 期：\_\_\_\_\_

## 在学期间研究成果使用承诺书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：内蒙古大学有权将学位论文的全部内容或部分保留并向国家有关机构、部门送交学位论文的复印件和磁盘，允许编入有关数据库进行检索，也可以采用影印、缩印或其他复制手段保存、汇编学位论文。为保护学院和导师的知识产权，作者在学期间取得的研究成果属于内蒙古大学。作者今后使用涉及在学期间主要研究内容或研究成果，须征得内蒙古大学就读期间导师的同意；若用于发表论文，版权单位必须署名为内蒙古大学方可投稿或公开发表。

学位论文作者签名：\_\_\_\_\_ 指导教师签名：\_\_\_\_\_

日 期：\_\_\_\_\_ 日 期：\_\_\_\_\_

# 基于 SDN 的可扩展转发设备架构设计及关键技术实现

## 摘 要

网络技术正在随着网络需求的多样性而变得越来越多样化，SDN 以其独特数控分离和灵活可编程特性吸引了人们越来越多的关注。SDN 当前主要关注和解决了控制平面的可扩展性和动态部署问题，对数据平面的可扩展性和动态部署问题的研究才刚刚起步，数据平面正越来越成为限制 SDN 发展的一个瓶颈。解决数据平面的可扩展性和动态部署问题的关键之一在于合适的转发设备体系架构，因此本文将关注的重点放在了转发设备的新型架构设计和该架构中关键技术的设计与原型实现上。本文的主要研究内容包括：

(1) 针对当前转发设备可扩展性和动态部署能力不足的问题，结合转发设备方向的最新研究内容提出了一种新型的转发设备架构，对该架构中语言部件、系统构建部件、构件库管理部件、运行环境部件和网络环境部件进行了详细的定义与描述。

(2) 对五个部件中的关键技术进行详细设计与原型实现，主要包括基于 Java 实现的网络功能开发类库、以图论知识为基础的系统构建部件的实现方案、基于 OSGI 框架的运行环境部件实例设计与实现、基于扩展协议解析树模型的数据包解析方案和扩展了 OpenFlow 使支持对运行环境部件的网络功能进行重构。

(3) 通过特定应用场景分析本文架构在可扩展性和动态部署方面的优势，并以原型系统为基础部署了本文设计的 UUID (Universally Unique Identifier) 网络证明了该架构的灵活性和可行性。

**关键词：**SDN；转发设备架构；可扩展性；动态部署；扩展协议解析树模型

# **DESIGN OF EXTENSIBLE FORWARDING ELEMENT ARCHITECTURE AND ITS KEY TECHNOLOGY VERIFICATION**

## **ABSTRACT**

Network technology is becoming more and more diverse with the diversity of network requirements, and SDN attracts more and more attention with its unique data plane and control plane separation and flexible programmability. SDN currently focuses on and resolves the scalability and dynamic deployment of control plane, but researches on the scalability and dynamic deployment of data plane has just started. So the data plane is increasingly becoming a bottleneck to limit the development of SDN. The key technology to solve the scalability and dynamic deployment of data plane is the architecture of the forwarding element (FE), so this paper focus on the design of FE architecture and the design and prototype implementation of the key technologies. The main research contents of this paper include:

(1) Aiming at the problem that the scalability and dynamic deployment capability of current forwarding elements is insufficient. The forwarding element architecture is proposed based on the latest research of the data plane and it includes language element, system construction element, component library management element, runtime environment element and network environment element. They are defined and described in detail.

(2) The key technologies of the five elements are designed and the prototype

systems are implemented, they mainly include network function development library based on java as a language element instance, a system construction element solution based on graph theory, the design and implementation of runtime environment element instance based on OSGI, the packet parsing solution based on the parsing method of extended protocol parsing tree model (EPPTM) and network environment element instance which OpenFlow is extended to support the restructure of network functions of runtime environment element.

(3) Analyze the advantages of our proposed FE architecture in the scalability and dynamic deployment through the specific application scenarios, and a UUID network is designed and implemented based on prototype system.

**KEYWORDS:** SDN; FE architecture; scalability; dynamic deployment; EPPTM

# 目 录

摘 要.....	II
目 录.....	V
图表目录.....	VII
第一章 引言.....	1
1.1 背景简介.....	1
1.2 可扩展转发设备的重要意义.....	1
1.3 研究工作.....	2
1.4 论文组织结构.....	2
第二章 相关工作.....	4
2.1 SDN 研究现状.....	4
2.2 传统网络环境下转发设备研究现状.....	5
2.3 SDN 环境下转发设备研究现状.....	6
2.4 协议数据包解析方法研究现状.....	7
2.5 前导知识.....	8
2.6 本章小结.....	9
第三章 基于 SDN 的可扩展转发设备架构设计.....	10
3.1 架构概述.....	11
3.2 语言部件.....	14
3.3 系统构建部件.....	15
3.4 运行环境部件.....	17
3.5 网络环境部件.....	20
3.6 构件库管理部件.....	21
3.7 本章小结.....	24
第四章 可扩展转发设备架构关键技术设计与原型实现.....	26
4.1 基于 Java 的语言部件设计与实现 .....	26
4.2 系统构建部件设计与实现.....	27

4.3 运行环境部件设计与实现.....	30
4.3.1 基于 OSGI 框架的运行环境部件设计 .....	30
4.3.2 基于 OSGI 框架的运行环境部件原型实现 .....	32
4.4 网络环境部件设计与实现.....	34
4.5 构件库管理部件设计与实现.....	36
4.5.1 构件库管理部件的详细设计 .....	36
4.5.2 构件库管理部件的原型实现.....	37
4.6 基于扩展协议解析树模型解析模块设计与实现.....	40
4.6.1 扩展协议解析树模型.....	40
4.6.2 扩展协议解析树描述语言 .....	43
4.6.3 基于扩展协议解析树模型解析模块设计与实现.....	44
4.6.4 配套部署机制设计与原型系统实现.....	46
4.6.5 解析模块和部署机制性能评估.....	47
4.7 本章小结.....	50
第五章 应用场景分析与原型系统验证.....	51
5.1 应用场景分析.....	51
5.2 原型系统部署 UUID 网络.....	52
5.3 本章小结.....	55
第六章 结论及未来工作.....	57
6.1 论文工作总结.....	57
6.2 下一步工作.....	57
参考文献.....	58
致谢.....	62
攻读硕士期间发表的学术论文.....	63
主持与参加项目.....	64



## 图表目录

图 2.1 传统转发设备架构.....	5
图 2.2 SDN 转发设备架构.....	6
图 2.3 OSGI 框架图 .....	9
图 3.1 基于 SDN 的可扩展转发设备架构.....	10
图 3.2 网络功能运行时和可扩展转发设备架构组成.....	11
图 3.3 整体架构物理场景分布图.....	12
图 3.4 转发设备功能构建工作流程图.....	13
图 3.5 构件依赖关系矩阵图示例.....	16
图 3.6 运行环境部件图.....	18
图 3.7 构件库管理部件整体结构图.....	22
图 3.8 转发设备服务功能重构图.....	23
图 4.1 抽象转发设备处理流程模型.....	26
图 4.2 系统构建部件描述.....	28
图 4.3 运行环境部件实现结构图.....	31
图 4.4 Bundle 状态转化图 .....	33
图 4.5 运行环境部件原型系统效果图.....	34
图 4.6 扩展 OpenFlow 协议消息格式.....	35
图 4.7 构件库管理部件实例结构图.....	37
图 4.8 构件库管理部件关键数据结构类.....	38
图 4.9 构件管理界面效果图.....	39
图 4.10 转发设备管理效果图.....	39
图 4.11 系统构建部件描述管理界面效果图.....	40
图 4.12 Ethernet 协议的扩展协议解析树 .....	40
图 4.13 IP、TCP 和 UDP 组成的扩展协议解析树 .....	42
图 4.14 IP 协议的扩展解析树描述语言描述 .....	44
图 4.15 扩展协议解析树更新过程图.....	45

图 4.16 解析器模块结构图.....	45
图 4.17 部署机制整体架构图.....	47
图 4.18 传统解析与扩展协议解析树解析对比.....	48
图 4.19 传统解析与选定字段解析对比.....	48
图 4.20 部署机制实验模拟场景图.....	49
图 5.1 应用场景 1.....	51
图 5.2 应用场景 2.....	52
图 5.3 UUID 数据包格式.....	53
图 5.4 UUID 部署拓扑图.....	53
图 5.5 UUID 协议的扩展解析树描述语言描述.....	54
图 5.6 数据平面 Bundle 功能部分代码 .....	54
图 5.7 系统构建部件描述.....	55
图 5.8 主机接收数据包效果图.....	55
表 4.1 转发设备功能开发接口列表.....	27
表 4.2 系统构建部件执行方式表.....	28
表 4.3 扩展的 MANIFEST.MF 文件属性 .....	32
表 4.4 Bundle 操作命令集 .....	33
表 4.5 扩展 OpenFlow 协议消息表.....	35
表 4.6 第一组实验结果汇总.....	50
表 4.7 第二组实验结果汇总.....	50

# 第一章 引言

## 1.1 背景简介

计算机网络技术随着互联网时代的到来已经变得越来越重要，而互联网的发展也对计算机网络技术提出了新的需求和期望。以以太网和 TCP/IP 协议为事实标准的传统网络经过三十年的发展已经成为了开启互联网时代的底层基础，可以说正是以太网加 TCP/IP 协议的网络体系结构带来了繁荣的互联网时代，这足以证明该网络体系结构的强大和成功。但是以太网加 TCP/IP 协议的网络体系结构并不是没有缺陷，而且随着网络规模的不断扩大和网络功能需求的不断增加，这种经典网络体系结构也正在面临着越来越多的挑战，如扩展性和易管理等，由此诞生了软件定义网络（SDN），它为解决这些传统网络体系结构中存在的问题提供了可能。

目前 SDN 仍处于发展期，在协议和处理机制上还有很多要改进和完善的地方，特别是数据平面的可扩展性和动态部署问题仍然有很多问题需要进行深入的探索与研究。由于 SDN 中控制平面的重要性导致长期以来大多数研究者关注的重点放在了控制平面的扩展与完善上，关于数据平面的研究相对起步较晚，而且现在关注的重点在于如何提供数据平面的高转发速率和灵活可编程性上，对适应于 SDN 的转发设备架构并没有给出非常优秀的解决方案，适应于 SDN 的转发设备架构研究的欠缺为本文研究开展提供了很好的契机。

## 1.2 可扩展转发设备的重要意义

SDN 的发展非常迅速，其中控制平面的发展更是日新月异，而转发设备一般是基于硬件实现的，在设计和实现上很难跟上控制平面的更新与发展，这就造成了 SDN 技术规范虽然已经更新至最新而转发设备的设计与实现仍然停留在已经实现的 SDN 技术规范上。即使是软件实现的转发设备也存在类似的情况，因为新规范出现之后程序员要有一个开发测试周期和一个更新部署周期。这种情况带来的最大问题便是原有的硬件转发设备和软件转发设备并不支持最新的 SDN 技术规范，转发设备又不提供对新规范的扩展能力，因此会造成在 SDN 的发展过程中大量的原有转发设备还没充分使用就已经过时的尴尬局面。转发设备的可扩展性和

动态部署问题正在成为限制 SDN 发展的瓶颈之一。

本文从网络体系架构的角度提出了一种可行的可扩展转发设备架构来解决转发设备的可扩展性和动态部署问题。该架构使转发设备的功能可以伴随 SDN 技术规范的发展而发展，在转发设备上动态灵活部署控制平面需要的新功能，灵活支持新协议的解析与扩展，这样不但可以减少大量老旧转发设备投入的浪费，而且可以在不改变现有架构的情况下快速实现对 SDN 新规范的支持。本文提出的可扩展转发设备架构可以很好解决 SDN 在数据平面面临的挑战，最大限度地释放 SDN 强大的可扩展能力和可编程能力，以促进 SDN 的快速发展与应用。本文架构可以使网络用户根据自己的需求构建更加合适的网络环境去承载自己的业务需求，同时可以根据需求变化轻松实现网络环境的演化。

### 1.3 研究工作

本文的研究工作主要包括以下三部分：

（1）针对当前转发设备可扩展性和动态部署能力不足的问题，结合转发设备方向的发展现状提出了一种新的转发设备架构，并对该架构中语言部件、系统构建部件、构件库管理部件、运行环境部件和网络环境部件进行了详细的定义与描述。

（2）采用 Proof Of Concept 思想对五个部件中的关键技术进行详细设计与原型实现，主要包括基于 Java 实现的网络功能开发类库、以图论知识为基础的系统构建部件的实现方案、基于 OSGI 框架的运行环境部件实例设计与实现和扩展了 OpenFlow 协议使其可以支持对运行环境部件的网络服务功能进行重构。同时为了解决数据包解析协议无关性问题，本文提出了基于扩展协议解析树模型的解析方案并进行了实现。

（3）通过特定应用场景分析了本文架构在可扩展性和动态部署方面的优势，以原型系统为基础部署了本文设计的 UUID 网络证明了该架构的灵活性和可行性。

### 1.4 论文组织结构

本文组织结构如下：

第一章介绍研究背景和研究意义，从而引出了本文的研究内容。

第二章从 SDN 研究现状、传统网络环境下转发设备研究现状、SDN 环境下转发设备研

究现状、协议数据包解析方法研究现状和前导知识五个方面进行阐述，对本文要解决问题的研究现状进行了总结。

第三章是本文的理论核心，提出了由五个部件组成的适应于 SDN 的可扩展转发设备架构，对语言部件、构件库管理部件、系统构建部件、运行环境部件和网络环境部件进行了详细的定义与描述，其中该架构的核心是运行环境部件和语言部件，运行环境部件就是可扩展转发设备的具体实现。

第四章介绍了本文架构中各个部件的一些关键技术的详细设计与原型实现。其中包括使用 Java 语言实现的网络功能开发类库；以图论知识为基础提出了一种可行的系统构建方案；对构件库管理部件进行了详细设计并进行了原型实现；对基于 OSGI 框架的运行环境部件进行了详细设计与原型实现；网络环境部件主要是扩展了 OpenFlow 协议使其可以支持网络功能构件灵活部署能力；同时为解决协议解析无关性问题提出了基于扩展协议解析树模型的数据包解析方案。

第五章以应用场景为基础，分析本文提出的转发设备架构在可扩展性和动态部署方面的优势，同时利用第四章实现的原型系统对本文设计的 UUID 网络进行开发和部署以验证转发设备架构的可行性与灵活性。

第六章是对本文工作进行总结并对下一步工作进行规划和展望。

## 第二章 相关工作

### 2.1 SDN 研究现状

SDN 随着 Nick McKeown 教授等在 2008 年提出 OpenFlow 协议而诞生<sup>[1]</sup>, 随后 SDN 技术以其独特的数控分离和可编程性开始了快速发展, 特别是近些年数据中心的快速发展将 SDN 从研究推向实际应用, 其中 Google 的 B4 系统<sup>[2]</sup>的良好表现更是将 SDN 发展推向了高潮。SDN 并不是突然产生的网络技术, 它是网络体系架构经历了长期探索和积淀的产物, 在 SDN 之前已经存在数控分离的体系架构 ForCES<sup>[3]</sup>和可编程性较强的主动网络<sup>[4-5]</sup>, 4D 架构<sup>[6-7]</sup>将可编程性较强的决策平面(即控制平面)从数据平面分离, 该设计思想正是 SDN 技术的雏形。主动网络技术由于它的复杂性和安全性问题渐渐退出了历史舞台, ForCES 由于和 SDN 极其相似正在转向 SDN 网络的一种实现, 4D 架构则由 SDN 架构所取代, 但是它们和 SDN 一样对转发设备实现提出了新的需求, 是本文转发设备架构设计的重要参考。

SDN 控制平面的发展非常迅速, 其中已经实现的控制器就不下十几种, 比较著名有 OpenDayLight<sup>[45]</sup>、ONOS<sup>[46]</sup>、FloodLight<sup>[47]</sup>和 NOX<sup>[48]</sup>等。研究者和从业人员在不断完善 SDN 对 TCP/IP 网络功能支持的同时, 正在将 SDN 扩展到网络功能虚拟化和移动网络领域, 由于 SDN 的数控分离和可编程特性可以很好地支持网络功能虚拟化和移动网络的扩展, 因此 SDN 在这两个领域的研究逐渐成为了新的热点。SDN 另一个研究热点是北向 API 的标准化, 但是由于各方对 SDN 的理解和期望不同造成了北向 API 标准化实现比较困难, 因此基于北向 API 的编程语言开始出现并不断发展, 它们的设计者期望可以利用编程语言屏蔽控制器北向接口实现的差异性, 为网络用户提供一套灵活统一的开发接口<sup>[8-10]</sup>。然而, 相对于控制平面而言发展比较缓慢的数据平面并不能很好地支持控制平面的快速创新, 由此数据平面正在成为限制控制平面创新和 SDN 发展的一个瓶颈。其中比较突出的问题就是数据平面的可扩展性和动态部署问题。例如 OpenFlow 协议<sup>[11]</sup>作为 SDN 南向接口事实上的标准正在变得越来越复杂, 对数据平面要求也越来越高, OpenFlow 协议中很多新功能在用户的网络需求中有可能不是必须的, 但是为了规范实现的完整性数据平面却必须要实现才能支持, 无形中造成了不必要的资源浪费, 更严重的问题是数据平面当前的体系架构并不能动态支持 SDN 的功能扩展, 因而造成控制平面的创新总是会受制于数据平面的更新。可编程网络芯片的设计与开发为数据平

面的动态更新提供了契机,  $\text{POF}^{[12]}$ 和  $\text{P4}^{[13]}$ 的提出正是这种大环境中产生的, 它们都致力于解决数据平面的可编程能力, 但是它们都没有提出适应于  $\text{SDN}$  环境的转发设备架构。文献[14]提出了一种可扩展的架构可以支持将代码块动态部署到交换机中的能力, 但是该架构是针对特定芯片提出的, 对底层芯片可编程的差异性没有考虑, 并且该架构不具有通用性。文献[15]是我们之前提出的一种适应于  $\text{SDN}$  环境的可扩展转发设备架构, 其中该架构的核心是三层转发设备架构, 转发设备操作系统构成了转发设备动态可扩展的核心, 但是它存在的问题是转发设备操作系统的扩展与开发困难较大, 动态部署灵活性较差。

## 2.2 传统网络环境下转发设备研究现状

传统网络环境下的转发设备架构模型如图 2.1<sup>[16]</sup>一般将控制平面和数据平面物理上紧密耦合在一起, 控制平面负责数据包转发决策并生成转发表, 数据平面负责对数据包进行快速转发。传统转发设备一般是基于  $\text{ASIC}$  芯片实现的, 它的优点是可以提供较高的转发速率, 缺点是不具备可编程性, 动态扩展困难。

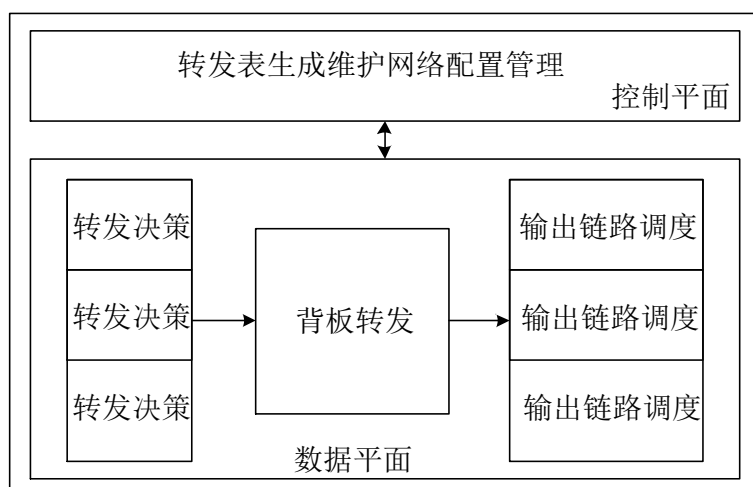


图 2.1 传统转发设备架构

Figure 2.1 Traditional forwarding element architecture

在传统网络环境下的转发设备架构模型研究中, 比较优秀的代表有  $\text{NetFPGA}^{[49]}$ 、 $\text{Click}^{[18]}$ 、 $\text{XORP}^{[19]}$ 和  $\text{Pearl}$  架构<sup>[20]</sup>。NetFPGA 是一个完全可编程的硬件平台, 它有多个可编程的硬件模块组<sup>[17]</sup>。Click 是一个模块化的软件实现的路由器, 第三方网络开发者可以通过添加子功能模块实现对 Click 的编程, 可以使得路由器软件更加灵活且易于扩展以支持第三方网络开发者自定义的新的网络协议。XORP 与 Click 的设计类似, 只是 XORP 对 Click 进行了扩展, 在内核

功能模块之外扩展出了用户空间功能模块，增加编程的灵活性<sup>[18-19]</sup>。Pearl 架构是一种软硬结合的转发设备可扩展解决方案，Pearl 架构既可以利用了软件平台处理来保证对新协议的支持，又利用了硬件平台来保证数据流转发的性能<sup>[20]</sup>。这些转发设备可扩展方案最大的问题是针对传统 TCP/IP 协议架构提出的，并不能与 SDN 环境配合解决 SDN 数据平面的可扩展性和动态部署问题。

## 2.3 SDN 环境下转发设备研究现状

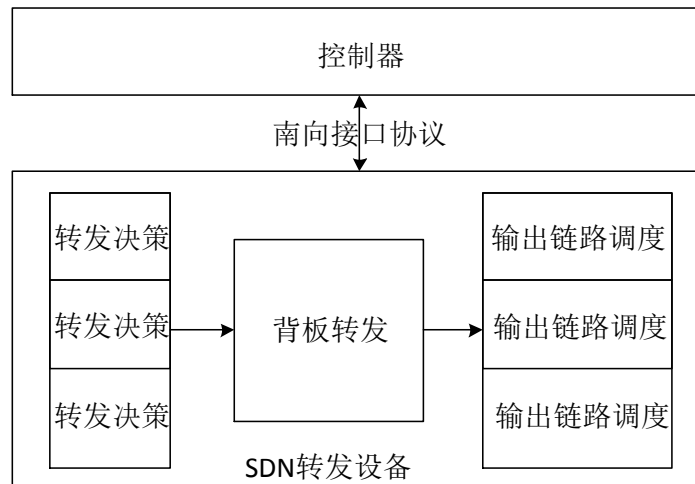


图 2.2 SDN 转发设备架构

Figure 2.2 SDN forwarding element architecture

SDN 转发设备架构模型如图 2.2<sup>[16]</sup>将控制平面和数据平面完全解耦，控制器实现了控制平面的功能，同时通过南向协议将数据包转发决策下发到数据平面从而控制数据包的转发，网络的配置管理也可以由控制器完成，由此大大提高了网络管控的效率。转发设备因此可以只关注数据平面的功能，降低转发设备实现的复杂度。尽管如此，SDN 的快速发展对转发设备的可编程能力要求反而不断增强，这直接促进了可编程网络芯片的发展，通用 CPU、FPGA(Field Programmable Gate Array)和 NP(Network Processor)等可编程网络芯片的代表也因此得到了充分的发展。随着这些可编程网络芯片性能的不断提升，在转发设备的设计与实现中正在被逐步采用。这也为本文可扩展转发设备架构的可行性提供了底层芯片可行性的基础。在工业界思科、华为和惠普等一批传统转发设备厂商都已经生产出了商业化的 SDN 转发设备产品，并且提供了不俗的转发效率<sup>[16]</sup>。特别是华为基于自研 END 芯片生产的 S12700 系列敏捷交换机<sup>[21]</sup>提供了非常强大的可编程能力，支持 POF 技术可以做到协议无关转发，同时拥有



非常良好的数据包处理性能，为本文提出架构的性能可行性提供了重要依据。

相对于硬件开发成本的高昂和开发周期长的问题，软件交换机的开发成本较低、配置更为灵活，同时又可以满足中小规模实验网络的要求，因此 SDN 软件交换机也随着 SDN 发展而迅速发展起来。其中比较著名的是 Open vSwitch<sup>[22]</sup>，它已经被广泛地应用在各种网络应用场景之中。在国内，北京邮电大学的周通等人提出了支持多协议的 SDN 交换机，通过添加协议解析表组件的方式使得第三方开发者可以在 SDN 的控制器端以基于 XML 的 NetPDL 语言的形式配置自定义的网络协议到交换机的协议解析表中<sup>[23]</sup>。OpenFlow1.3<sup>[24]</sup>已经关注到了协议处理无关性的重要性，因此添加了 TLV 格式的流表匹配项的支持，使从数据包中提取匹配字段信息更加灵活，可以支持更多协议，但是也因此使转发设备的设计与实现变得更加复杂。POF 提出通过将交换机初始化成一个不提供任何管理功能的白盒，控制器端可以通过使用 POF 提供的操作指令进行编程实现对数据包的解析、分析和转发等功能的扩展<sup>[12]</sup>。P4 提出了抽象编程语言的思想来解决 SDN 交换机对特定协议版本依赖难以扩展的问题，控制器端可以通过 P4 语言编程实现对交换机的扩展<sup>[13]</sup>。POF 的目标是提供类似于 x86 体系的转发设备底层指令集，通过底层指令集可以完成对数据平面的动态重构以支持 SDN 控制平面的创新与发展。P4 的目标是提供类似于传统 PC 高级编程语言的高级转发设备编程语言，通过该语言可以屏蔽底层转发设备厂商原始指令集，开发能力更强，效率更高。POF 为数据平面的扩展提供了底层指令集，由硬件厂商进行实现和维护；P4 为数据平面的扩展提供了更方便的开发工具，由语言开发商进行实现和维护，P4 要实现具体网络功能必须编译成底层指令集才能完成，由此可以看出 POF 和 P4 相互配合才能使转发设备的高度可扩展性成为现实。它们没有解决与 SDN 控制器的协作问题，但是它们为数据平面的可扩展性和动态部署问题的解决提供了基础，这些研究为本文架构的提供了非常重要的铺垫。

## 2.4 协议数据包解析方法研究现状

协议解析无关性是数据平面可扩展问题中非常核心的问题之一，很多文献都针对该问题进行了研究。文献[25]中作者对通用网络数据包格式进行了研究，提出了协议字段之间相关度的概念，本文借鉴了该文的思想。文献[26]对网络数据包解析工作进行了深入的分析，提出了一种基于 XML 技术的网络协议描述语言 NetPDL，实现了一套完整但复杂的网络数据包解析描述语言，但该语言主要关注数据包格式和解析流程的描述，与本文中的组织协议字段

便于解析的思路存在很大不同。文献[27]为基于 FPGA(Field Programmable Gate Array)的高速网络系统设计了一种解析网络数据包的新架构,在该架构中引入了基于 XML 网络协议描述标签,这套标签只关注协议包格式的描述。文献[28]的作者在无线网络协议解析过程中应用了基于 XML 网络协议包格式描述方法。文献[29]提出了一种用于描述网络数据包解析算法的语言,该语言可基于 FPGA 的解析器进行编译实现对相关协议的解析。文献[30]的作者提到了解析树的概念,同时也提出了 Kangaroo 系统以及如何实现一次对多个数据字段进行解析的方法。文献[31]的作者提出了 packetC 语言,该语言抽象封装了网络数据解析涉及到的数据类型和操作,并将 packetC 解析器的设计重点放在了异构的并行数据包解析处理模型上。文献[32]将二叉 trie 树引入到数据包解析工作中,通过建立协议二叉 trie 树来支持报文协议解析的灵活性。文献[33]中提出了通用协议解析接口的概念,通用协议解析接口可以向解析器中送入要解析协议的数据包和该协议包对应协议的字段偏移信息,从而实现对数据包的解析。文献[34]研究了基于 WindSock 技术的数据包解析技术,实现方式是为每种协议开发配套的解析程序。文献[35]研究了如何通过插件方式扩展网络数据包解析的功能,当有新的协议需要解析时可以通过开发新协议的解析插件的方式实现对网络数据包的解析,然而它们的本质仍然是为每种协议开发配套的解析程序。

## 2.5 前导知识

在解决软件危机的过程中人们逐渐认识到实现软件开发的工业化是解析危机的唯一途径,符合标准的构件生产以及基于标准构件的软件生产是软件产业发展的必经之路,其中,构件是核心和基础,重用是必须的手段<sup>[36]</sup>。基于构件的系统开发以其良好的可重用性、高效性在网络体系架构的实现中也有非常广泛的应用。文献[37]提出了一种基于构件的层次化可重构网络体系结构并给出了基于进程代数的重构模型、原则、功能实体合作方法。文献[38-39]在他们的系统设计与实现中采用了基于构件的重构思想。文献[40]中详细分析了基于构件的可重构路由平台结构模型并进行了原型系统实现。文献[41]提出了基于构件运算的可重构系统代数模型,从理论上提供了对基于构件的可重构网络体系结构的支持。OSGI 框架是一种支持灵活扩展和动态部署的开发框架,它的核心标准是定义了一个动态化构件化的应用架构。该框架提供了一个通用安全可管理的 Java 框架,能够支持动态部署构件程序<sup>[42]</sup>。

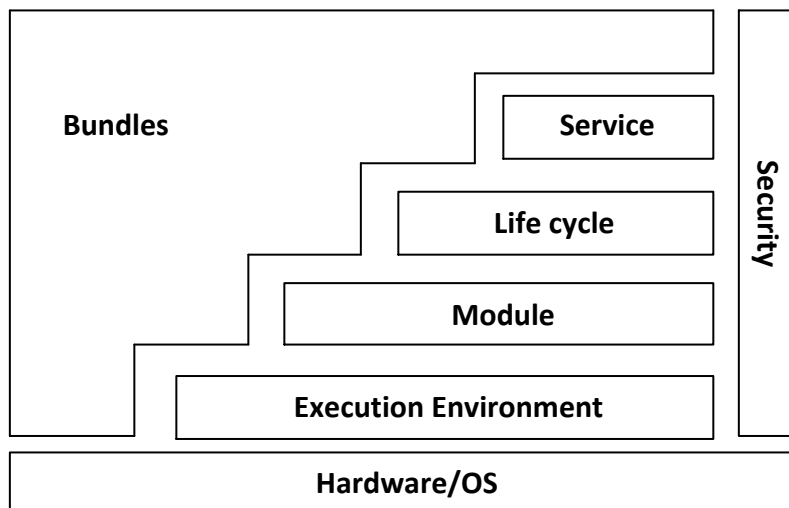


图 2.3 OSGI 框架图

Figure 2.3 OSGI Framework Overview

从图 2.3<sup>[42]</sup>中可以看出 OSGI 框架主要由服务层、生命周期层、模块层、安全层和执行环境组成，其中 Bundles 代表了构件集合，通过构件 Bundle 的动态安装、卸载和修改完成系统功能的升级与重构。文献[43]提出了一种软件体系结构演化模型，为软件体系结构演化的管理、控制、量化描述和评价奠定了基础。文献[44]针对当前 OSGI 框架只应用于 Java 平台的局限性尝试将 OSGI 框架移植到兼容 POSIX（可移植操作系统接口）的操作系统上并提供了原型实现，该原型系统证明了移植的可行性，但是相对于基于 Java 的 OSGI 框架，如 Equinox 和 Felix，它的功能完善性较差还不适合实际应用场景的需要。基于构件的可重构网络的设计思想和 OSGI 框架的强大的动态重构特性为本文框架的提出提供了重要的知识和技术准备。

## 2.6 本章小结

本章从 SDN 研究现状、传统网络环境下转发设备研究现状、SDN 网络环境下转发设备研究现状和协议数据包解析方法研究现状和前导知识五个方面对本文要解决问题的研究现状进行了总结。从以上参考文献可以看出 SDN 的数据平面在可扩展性和动态部署问题上仍存在不足，而以软件体系结构知识为基础的解决方案在 SDN 网络的设计中目前没有出现，虽然在传统互联网环境中研究者有所尝试，但只局限在模块可扩展上，如何灵活部署和重构网络的考虑并不是很完善。本文正是针对当前存在的这些问题，以网络体系架构的角度提出了自己的解决方案，以解决本文待解决的问题。

### 第三章 基于 SDN 的可扩展转发设备架构设计

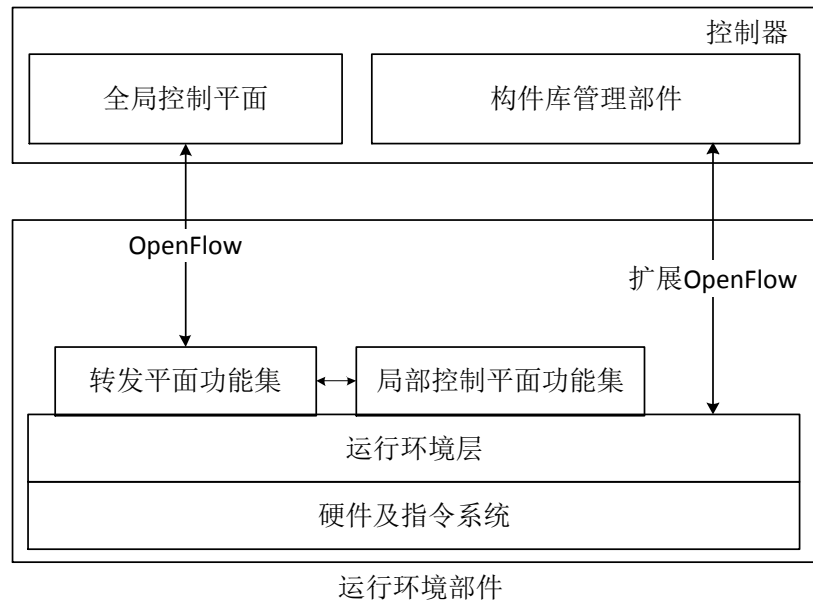


图 3.1 基于 SDN 的可扩展转发设备架构

Figure 3.1 The extensible forwarding element architecture based on SDN

从图 2.1 中可以看出传统转发设备架构将控制平面和数据平面紧密耦合在转发设备中，它的优势是控制平面与数据平面物理上紧密结合在一块，处理速度快，缺点是维护扩展困难，整个网络是分布式的，需要大量分布式协议协调才能正常工作，从而也造成了转发设备的设计与实现比较复杂。从图 2.2 中可以看出在 SDN 转发设备架构将控制平面与数据平面在物理上进行分离，控制平面放在控制器中，数据平面则由转发设备实现，它的优势是集中化管理，容易扩展，数据平面设计相对传统转发设备较简单，缺点是对控制器要求较高，控制器有可能成为整个网络转发速率的瓶颈且控制器崩溃时整个网络将不能工作，而转发设备只负责转发，需要与控制器通信决策数据包处理动作，效率较低。通过对比可以发现传统转发设备架构和当前 SDN 转发设备架构都存在优点和缺点，有没有方式可以同时利用这两种架构的优势。本文架构给出了一种可行的解决方案，本文架构不仅可以提供转发平面功能和控制平面功能的开发能力，而且可以提供对新协议的动态支持和对转发设备功能集合的动态管理。通过构件库管理部件将功能构件集合动态部署到运行环境部件中，从而实现转发平面和控制平面服务功能的重构，由此也可以实现如图 3.1 控制平面功能的调节能力。如图 3.1 当控制器中的全局控制平面功能集合为空时，转发设备（运行环境部件）就成为了传统转发设备架构，当运行环境部件中的局部控制平面功能集合为空时，转发设备（运行环境部件）就成为了现在的

SDN 转发设备架构。网络管理人员可以根据自己的业务需求决定控制器中全局控制平面功能集合和运行环境部件中局部控制平面的功能集合的分配，从而可以提供更加灵活高效的网络服务，而不再局限于架构的形式是分布式的，还是集中式的。

### 3.1 架构概述

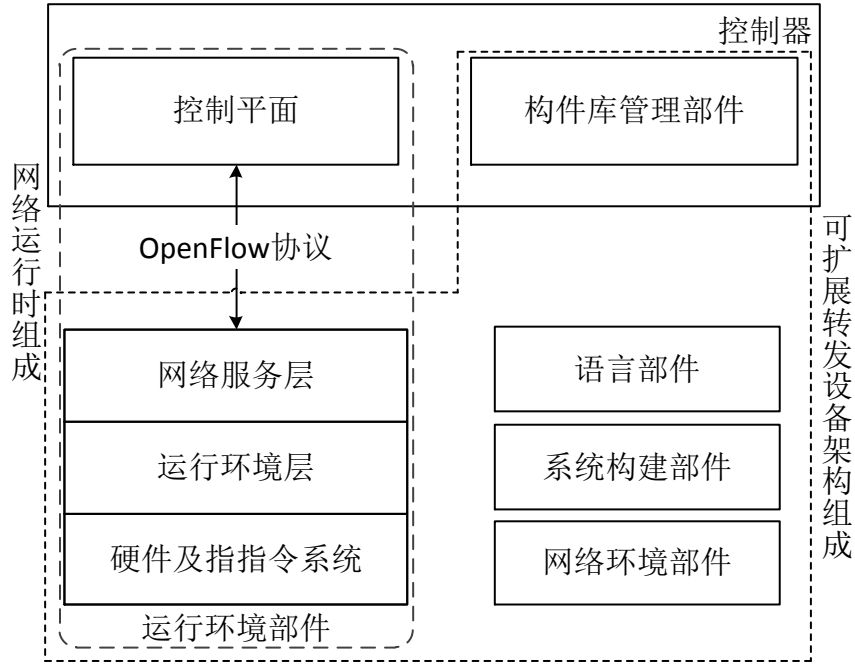


图 3.2 网络功能运行时和可扩展转发设备架构组成

Figure 3.2 The constitute of network runtime and extensible forwarding element architecture

参考 SDN 依据功能逻辑将网络划分为控制平面和数据平面的思想，本文站在网络运行与维护的角度将网络服务功能的实现分成两部分：网络功能运行时和可扩展转发设备架构，网络功能运行时负责使用可扩展转发设备架构构建的网络服务功能为网络用户提供服务，可扩展转发设备架构则负责提供构建和维护网络服务功能。为了解决 SDN 数据平面的可扩展性和动态部署问题，本文将网络功能运行时和可扩展转发设备架构的划分作为 SDN 架构的扩充应用于 SDN 网络转发设备的设计与实现。从图 3.2 中可以看出网络功能运行时可以用当前的 SDN 技术划分方案进行划分，分为控制平面和运行环境部件；可扩展转发设备架构可以用本文提出的架构进行描述和刻画，该架构包括语言部件、构件库管理部件、系统构建部件、运行环境部件和网络环境部件五部分。当前 SDN 控制器主要关注控制平面的逻辑与功能的设计与实现，而在本文中对 SDN 控制器的能力进行了扩充，添加了对转发设备网络服务的构建能力。在本文的可扩展转发设备架构功能设计中，不仅包括转发设备架构本身的设计，而且还

包括一组用于方便转发设备进行网络服务功能开发与维护的配套机制。扩充后的 SDN 控制器将由控制平面的控制管理功能模块和转发设备网络服务构建功能模块两大部分组成，控制平面的功能负责在网络运行时对整个网络进行转发决策和管理，转发设备网络服务构建功能则在网络需求发生变化时通过可扩展转发设备架构中的相关部件配合实现对网络功能的重构和升级。控制器的控制平面管理功能和转发设备网络服务构建功能两大模块的密切配合将会使得网络服务的构建能力和动态部署能力变得非常强大。

网络功能运行时是相对于可扩展转发设备架构提出的，可扩展转发设备架构可以利用五个部件完成转发设备所需要的功能集合的开发与部署，然后启动这些功能集合实现用户需要的网络运行服务，控制平面和数据平面协调配合的目的正是为用户提供网络运行服务，网络功能运行时因此得名。可扩展转发设备架构不仅对转发设备本身架构进行了新的定义与描述，同时也对配套扩展部署机制进行了详细的定义与描述，且应该指出本文中的可扩展是指转发设备的服务功能可以灵活修改和重构。在具体展开对各部件的详细定义与描述之前，在此对整个可扩展转发设备架构的工作原理进行概括性的介绍，然后再依次进行详细介绍，目的是避免大家对整个架构的工作原理产生不必要的误解。

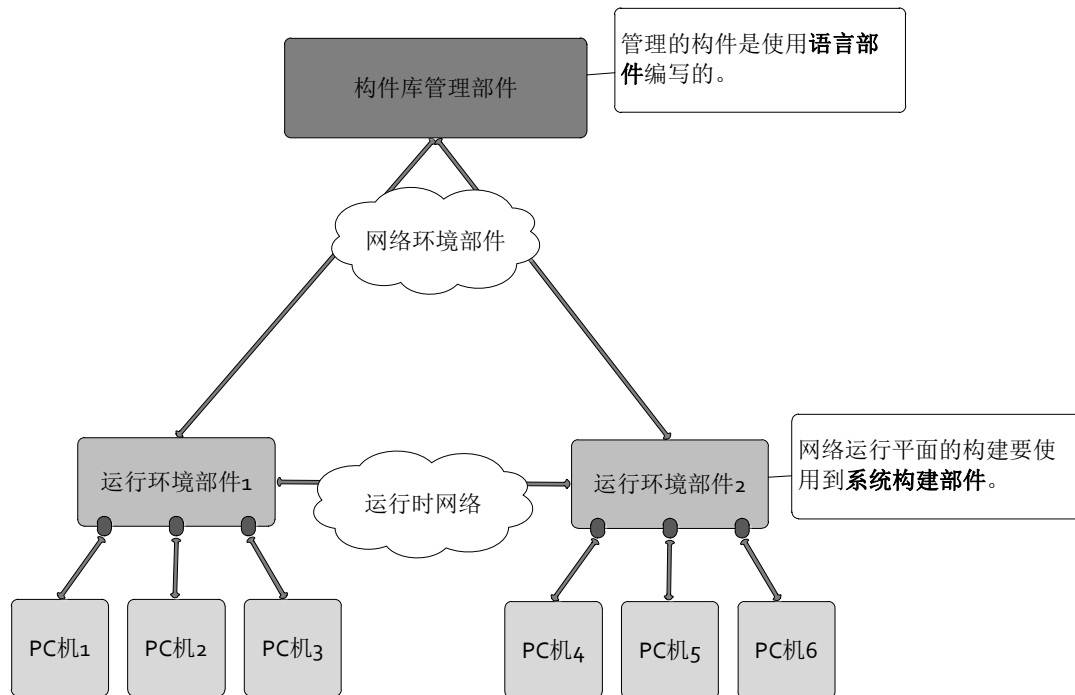


图 3.3 整体架构物理场景分布图

Figure 3.3 The overview of the scalable FE achitecture

可扩展转发设备架构充分利用了 SDN 网络可以对转发设备进行集中化管理的优势来实现转发设备的灵活扩展和动态部署功能，该架构更像一个有中央控制器的分布式网络管理系统

统，不同的是本文架构将 POF 和 P4 的思想融入了该架构设计中，为未来网络架构的发展提出一个新的方向，而且简化了转发设备本身的管理模型，将构件化编程思想引入了 SDN 转发设备的实现中。整个架构管理的对象是以转发设备功能构件为基础，而转发设备本身需要实现和支持的只是对构件进行本地化编译、维护和管理。语言部件提出的初衷是利用 P4 等高级抽象语言的易用性和屏蔽底层转发设备细节的优势，降低转发设备功能构件开发与维护的难度，语言部件是该架构可以实现的核心基础之一。构件库管理部件是管理功能构件的中央控制器，构件库管理部件通过网络环境部件（如图 3.3 中所示）与运行环境部件进行通信实现对网络服务功能的维护和管理，构件库管理部件同时维护着转发设备功能构件集合和每个转发设备相关的系统构建部件描述，构件库管理部件在整个架构中是一个集中控制管理平台。运行环境部件是该架构的核心部件，它是整个架构可以正常工作的基础，该部件的具体实现需要提供对转发设备功能构件进行本地化编译、维护和管理的功能，运行环境部件要支持与构件库管理部件进行通信的功能，以便运行环境部件可以根据构件库管理部件发送过来的系统构建部件描述和功能构件集合实现对网络服务功能集合的维护与更新，该部件是一种添加了运行环境层的转发设备。系统构建部件用于解决运行环境部件在模块化构建转发设备网络服务时告诉运行环境部件应该如何正确安装各个构件模块，解决各个构件之间的相互依赖关系和启动顺序的问题。网络环境部件是构件库管理部件与运行环境部件实现通信的基础网络。

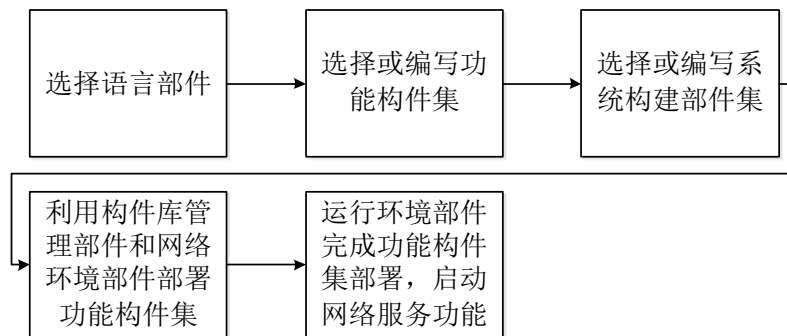


图 3.4 转发设备功能构建工作流程图

Figure 3.4 The work flow of FE function construction

在介绍了各个部件的功能与职责之后，依照图 3.3 和图 3.4 来介绍一下整个架构的工作流程以及如何实现转发设备的灵活扩展和动态部署。第一步，网络管理人员可以在不清楚网络转发设备底层细节的情况下利用语言部件进行网络功能构件程序的开发，在完成了所需网络服务功能构件集合的开发后可以进入第二步操作；第二步，网络管理人员根据网络服务功能构件之间的依赖关系和启动顺序编写系统构建部件描述，系统构建部件的不同描述会指导运行环境部件生成提供不同服务功能的转发设备实例，比如交换机或深度包检测设备；第三

步，在完成了转发设备功能构件集的开发和系统构建部件描述的编写之后将构件集合和系统构建部件描述提交到构件库管理部件中进行统一管理和维护，以实现构件集合和系统构建部件描述的部署和重用，构件库管理部件同时也维护着所有与该构件库管理部件连接的运行环境部件的基本信息和通信通道，通过构件库管理部件可以直接对每个运行环境部件进行管理维护和状态查询。网络管理人员可以找到自己的部署这些新功能的运行环境部件将构件集合和系统构建部件描述发布到指定的运行环境部件中进行转发设备功能的构建与生成；第四步，应该指出，在构件库管理部件和运行环境部件进行通信的过程中隐藏着我们的第四步动作，就是利用网络环境部件进行指令或者数据通信；第五步，运行环境部件在获得转发设备功能构件集合和系统构建部件描述之后，根据系统构建部件描述的安装启动过程实现对转发设备功能构件集合的安装与启动，从而实现转发设备功能的部署或重构。网络管理人员可以根据业务的需要开发新的功能构件，也可以利用已经开发和验证的功能构件来完成自己的转发设备功能构件集合的构建。当某个构件不再满足要求的时候可以通过本文框架进行运行时更新，不需要重启转发设备，也不会影响其他与该模块不相关的功能模块的正常工作。介绍完了可扩展转发设备架构的组成和工作原理之后，接下来我们将对各个部件进行比较详细的定义与描述。

### 3.2 语言部件

语言部件在我们的架构中是高度抽象了网络转发设备工作模型的编程语言的统称，在架构具体实现时代表一种或多种转发设备编程语言，比如 P4 语言。构件是本文架构管理和维护的核心，语言部件是编写功能构件的基础，我们相信构件化设计与部署是网络转发设备将来灵活扩展和动态部署的发展方向之一，因为只有构件化的开发方式才能保证转发设备始终可以维持比较合理的转发设备功能构件集合，对于不需要的转发设备功能构件可以即时拆除以节省系统资源；而且构件化的开发方式可以非常方便网络开发人员对已经存在的成熟的功能构件进行最大化的重用，从而大大提高开发效率。构件的实现离不开语言部件，因此可以看出语言部件是本文架构设计的核心部件之一。语言部件实现对网络转发设备的编程不可以绕开的一个话题是编译器，运行环境部件对编译器的支持是必不可少的，只有支持了语言部件的编译器才能实现对语言部件编写的构件进行本地化翻译与安装，屏蔽底层硬件指令系统的差异。不同的抽象转发设备编程语言对编译器的实现也有不同的要求，因此运行环境部件在



对抽象编程语言编译器的实现上要有充分的考虑。

本文并没有将语言部件等价于 P4 语言, 因为高度抽象的转发设备编程语言现在处于发展初期, P4 语言可以作为其中突出的代表, 但是就像 C 语言、C++ 语言和 Java 语言等传统 PC 机编程语言一样将来应该会有更多不同的转发设备编程语言出现, 因此 P4 语言只可以作为语言部件具体实现的一种, 而不能取代语言部件在本文架构中的职责和作用。利用传统 PC 机编程语言也可以用来开发网络转发设备的服务功能构件, 与 P4 语言不同, 传统 PC 机编程语言可以将 P4 文章中抽象出的转发设备工作模型以转发设备编程类库的方式提供给网络开发人员, 这样网络开发人员不但不用再重新学习 P4 语言来开发网络服务功能构件, 而且传统 PC 机编程语言已经非常成熟, 只要稍加改造就可以投入使用, 这一点相比 P4 语言来说还是有一定优势的, 当然转发设备编程语言的发展仍然是网络转发设备可编程性的大趋势, 利用传统 PC 机编程语言实现的方式只能是一种过渡方案。本文在原型系统的实现过程中采用了这种过渡方案。

### 3.3 系统构建部件

系统构建部件在我们架构中是对构件之间的依赖关系与部署顺序进行描述的一类描述方案的统称, 在架构具体实现时是一种按一定规则与格式描述构件之间的依赖关系和部署顺序的配置文件。系统构建部件配置文件将会和功能构件集合一起通过网络环境部件发送到运行环境部件中进行解析, 运行环境部件可以解析系统构建部件配置文件, 然后按照系统构建部件中确定的构件之间的依赖关系和部署顺序实现功能构件集合的安装与部署。原则上系统构建部件只要实现了对功能构件之间的依赖关系与部署顺序的正确描述并指导运行环境部件进行功能构件集合的安装便是一种比较优秀的部件实例, 但是有比较完善的形式化定义作为系统构建部件设计的依据将会带来更多的优势, 比如避免二义性, 可以进行正确性验证等。因此在参考了前人关于构件及软件体系结构的相关知识之后, 同时结合实际编程经验, 站在构件依赖关系和部署顺序的角度给出了关于构件和部署方案的形式化定义, 在定义的过程中简化了构件与构件之间的语义关系, 而将关注的重点放在了构件实际部署时更加在意的信息, 比如依赖关系以及部署的先后顺序。具体定义如下:

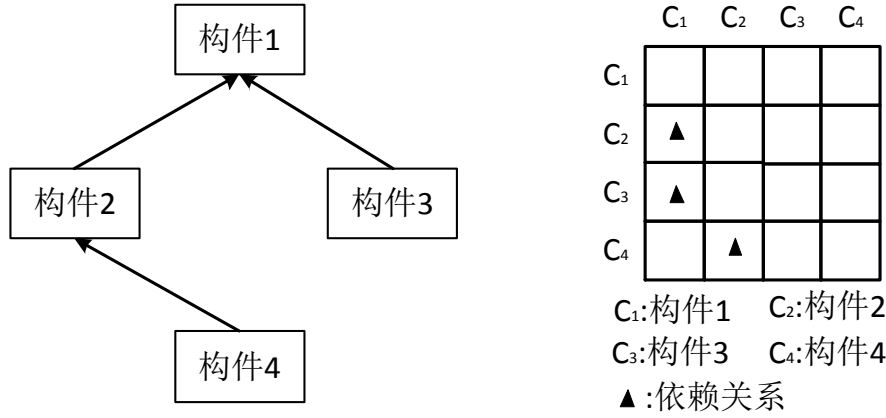


图 3.5 构件依赖关系矩阵图示例

Figure 3.5 component dependency matrix

**定义 1** 构件 Comp 是网络系统中承担一定功能的数据或计算单元且可以用于网络系统的构建过程中去。Comp= $\langle$ ID, Imports, Exports, Level $\rangle$ ，其中 ID 为网络系统中构件标识，推荐实现方案为“名称+版本号+时间戳”表示法。其中，Imports 是该功能构件依赖的功能接口子集的集合，Imports= $\{\text{PortGroup}_1, \text{PortGroup}_2, \dots, \text{PortGroup}_n\}$ ，每个 PortGroup 又可以表示为一个被依赖的功能接口集合，即 PortGroup =  $\{\text{Port}_1, \text{Port}_2, \dots, \text{Port}_n\}$ ，每一个 Port 表示一个被依赖的功能接口。其中，Exports 是该功能构件可以提供给其他构件的功能接口子集的集合，Exports= $\{\text{PortGroup}_1, \text{PortGroup}_2, \dots, \text{PortGroup}_n\}$ ，每个 PortGroup 又可以表示为一个提供给其他构件的功能接口集合，即 PortGroup =  $\{\text{Port}_1, \text{Port}_2, \dots, \text{Port}_n\}$ ，每一个 Port 表示一个提供给其他构件的功能接口。其中 Level 表示该构件在部署时的优先等级，等级越低将被越早部署，取值范围为正整数。

**定义 2** 构件 Comp A 在实现时使用或者依赖于构件 Comp B 提供的功能接口时称为构件 Comp A 依赖于构件 Comp B，即 Comp A 与 Comp B 之间存在依赖关系。如图 3.5 所示构件 1 的实现依赖于构件 2 提供的功能接口，即构件 1 与构件 2 之间存在依赖关系，图中构件 2 到构件 1 的边的箭头表示依赖关系的方向性，即箭头指向端构件依赖于非箭头指向端构件。

**定义 3** 如图 3.5 中右侧图所示是由左侧所有构件之间的依赖关系构成的依赖关系矩阵。一个依赖关系矩阵在实际描述中代表一种部署方案，由此可以推出形式化的部署方案定义。

**定义 4** 由定义 3 可以推出每种部署方案本质是一个有向无环图  $G_{\text{deploy}} = \langle V(G_{\text{deploy}}), E(G_{\text{deploy}}) \rangle$ ，其中  $V(G_{\text{deploy}})$  代表该种方案中所有功能构件； $E(G_{\text{deploy}})$  代表了功能构件之间依赖关系的集合。部署方案  $G_{\text{deploy}}$  中功能构件集合和依赖关系集合保持不变时就可以表示一种稳定的部署方案  $g_1$ ，当然当用户的需求发生变化时可以通过修改功能构件集合和依赖关系集合

达到另外一种稳定的部署方案  $g_2$ ，随着用户需求的不断变化而引起部署方案不断发生变化，从而产生部署方案域 Deploys，由此推出了部署方案域 Deploys 的形式化定义。

**定义 5** 部署方案域 Deploys 是一个部署方案的集合，即  $\text{Deploys}=\{G_1, G_2, \dots, G_n\}$ ，其中每一个  $G$  表示一个由功能构件集合和依赖关系集合构成的有向无环图。有了部署方案域 Deploys 的定义之后，用户的部署方案的变化将可以被限定在某个部署方案域中进行演化，由此可以在部署方案域 Deploys 中定义两种基本的演化操作，在定义基本演化操作之前，首先来定义一下演化操作因子。

**定义 6** 通过对实际中构件系统的演化过程进行观察与分析可以发现部署方案的变化不外乎是添加或删除一些功能构件和一些依赖关系，因此部署方案的本质可以看成是一种稳定的部署方案  $g_1$  经过添加或者删除一个  $G_{\text{factor}}=\langle V(G_{\text{factor}}), E(G_{\text{factor}}) \rangle$  之后到达另一种稳定的部署方案  $g_2$ 。其中， $G_{\text{factor}}$  被称为一个演化操作因子， $V(G_{\text{factor}})$  代表要添加或者删除的功能构件集合， $E(G_{\text{factor}})$  代表要添加或者删除的依赖关系集合。有了演化操作因子的定义之后，接下来定义两个基本演化操作。

**基本演化操作 1 加操作：**在部署方案域 Deploys 中一种稳定的部署方案  $g_1$  经过添加一个演化操作因子  $g_{\text{factor}}$  之后到达部署方案  $g_2$  的过程称为加操作，即  $g_2 = g_1 + g_{\text{factor}}$ 。

**基本演化操作 2 减操作：**在部署方案域 Deploys 中一种稳定的部署方案  $g_1$  经过删除一个演化操作因子  $g_{\text{factor}}$  之后到达部署方案  $g_2$  的过程称为减操作，即  $g_2 = g_1 - g_{\text{factor}}$ 。

有了上述的形式化定义和基本演化操作的定义之后，可以很容易地设计实现一种系统构建部件描述方案，本文将在原型系统实现中详细讲解我们的系统构建部件描述的实现。

### 3.4 运行环境部件

运行环境部件在我们架构中是对一类实现了下列功能的转发设备的统称，基本功能包括：与构件库管理部件通过网络环境部件进行交互的功能，可以根据系统构建部件描述编译、维护和动态部署用语言部件编写的功能构件集合的功能，启动功能构件集合提供网络服务的功能。运行环境部件必须包括列出的基本功能，但是并不限于这些基本功能，设备提供商可以根据自身需求扩展更多实用的功能，比如基于 WEB 的运行环境部件管理系统等。运行环境部件的实现依赖于硬件提供商的支持，在文献 POF 中提出的基于标准化的指令集的“白盒交换机”是一种非常不错的解决方案，然而该方案也有它的弊端。首先，POF 中的 FIS 指令系

统对硬件提供厂商的硬件设计与实现要求太过严格，很难在硬件提供厂商中间进行推广；其次，POF“白盒交换机”对硬件提供商已经开发和实现的硬件设备产品不兼容，不能重用之前已经非常成熟和稳定的硬件产品；最后，POF“白盒交换机”的初衷是提供类似 PC 机的 X86 指令集的转发设备指令集标准而不是提供类似于传统 PC 操作系统的资源管理平台，这也意味着 POF“白盒交换机”的灵活扩展能力仍然有限。

应该指出的是在 POF“白盒交换机”中假定 SDN 控制器可以提供类似于传统 PC 操作系统的资源管理功能，但是目前 SDN 控制器并不提供类似功能。SDN 控制器中控制平面的实例已经正在承担着越来越多的控制平面相关功能的运行任务，如果把数据平面的构建任务交给 SDN 控制器中控制平面模块实现不仅会加大 SDN 控制器实现的复杂性，同时也会重新混淆控制平面与数据平面的划分，因此本文在 SDN 控制器中扩展出了构件库管理部件模块来负责转发设备的重构功能将更容易扩展且不影响控制平面的维护与扩展。本文架构中提出了构件库管理部件与运行环境部件结合的方式来实现类似于传统 PC 操作系统的资源管理功能，用于完成对转发设备功能的动态维护和重构。构件库管理部件将在 3.6 小节进行详细的讲述，而本小节的运行环境部件是本文所提架构和构件库管理部件可以正常工作的核心基础之一。

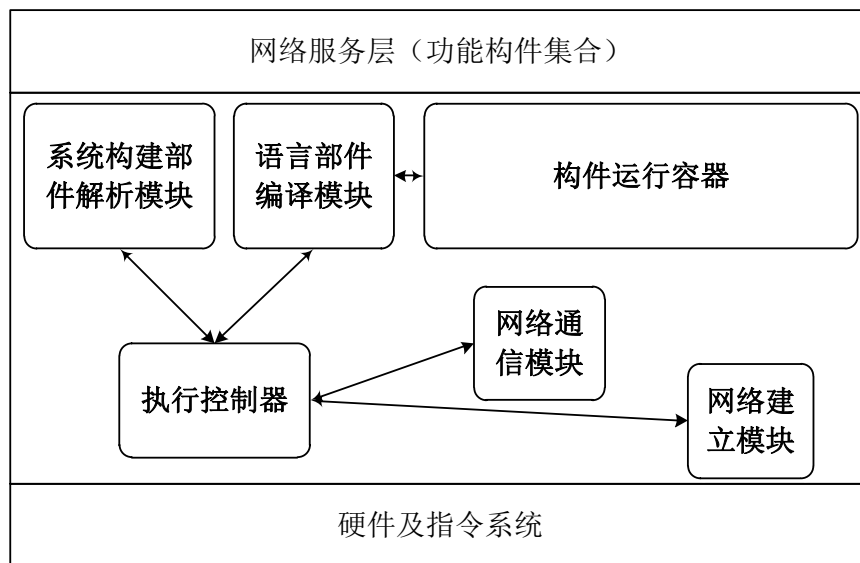


图 3.6 运行环境部件图

Figure 3.6 The architecture of runtime environment element

如图 3.6 所示，运行环境部件由三层组成。其中最底下的一层是硬件及指令系统层，POF 可以说是在硬件及指令系统层所做尝试的方案中比较优秀的一种，但是 POF 并没有提供与控制器协作对转发设备功能进行维护和动态部署的能力，硬件指令集编程效率低且对硬件实现依赖过强，因此 POF 只可以作为本文架构实现的一种硬件及指令系统解决方案，而不是

人们期望的“白盒转发设备”。在本文架构设计中将硬件及指令系统的设计与实现交给了硬件提供商而没有做过多限制，当然如果硬件提供商可以采用 POF 等大家公认的解决方案将会为系统功能维护与扩展带来非常大的优势。如图 3.2 中所示，本文推荐的解决方案是在硬件及指令系统之上添加一个**运行环境层**，该层可以与构件库管理部件通过网络环境部件进行交互，根据系统构建部件描述编译、维护和动态部署用语言部件编写的功能构件集合，可以启动功能构件集合提供网络服务。通过引入运行环境层，网络服务功能开发人员可以通过语言部件、系统构建部件、构件库管理部件和网络环境部件协完成功能构件集合的动态部署与更新而不用关注转发设备的底层硬件及指令系统的具体细节，硬件提供商也因此可以从对复杂转发设备功能的设计与实现中解脱出来，而将关注重点放在硬件及指令系统层和运行环境层的设计与实现，从而硬件提供商只关注由硬件及指令系统层和运行环境层组成的“白盒转发设备”的设计与实现，转发设备功能开发人员可以关注利用语言部件进行转发设备功能的开发、测试和部署，类似于传统 PC 机的软件产业与硬件产业模式将在计算机网络这一领域中成为一种趋势。最上面的一层是网络服务层，该层中的服务功能正是通过语言部件、系统构建部件、构件库管理部件和网络环境部件在运行环境层中完成功能构件集合的部署之后提供的功能服务集合。需要指出的是语言部件是屏蔽底层硬件及指令系统的编程方式可以实现的核心，因此运行环境层只有提供语言部件编译功能才可以将语言部件编写的程序编译成本地可执行的硬件指令。类似于传统 PC 系统中的高级语言一样，一种语言部件可以满足所有转发设备功能开发需求的可能性比较低，因此运行环境部件可以提供对多种语言部件进行编译的能力。由此可以给出，在本文架构中定义的“**白盒转发设备**”，是基于特定的硬件及指令系统提供了标准化的、规范化的运行环境层的转发设备。

在运行环境部件中运行环境层包含如图 3.6 中所示的执行控制器、构件运行容器、系统构建部件解析模块、语言部件编译模块、网络通信模块和网络建立模块六个功能模块。其中，执行控制器负责协调控制其他模块共同完成功能构件集合的维护与部署；构件运行容器是负责功能构件集合的安装、卸载、启动、停止等维护工作的容器；系统构建部件解析模块负责对系统构建部件描述进行解析并通知执行控制器按照解析的安装维护指令指导语言部件编译模块管理功能构件集合的编译和安装；语言部件编译模块负责按照执行控制器发送的命令和功能构件集合进行功能构件集合本地化编译部署和对构件运行容器发送维护命令的功能；网络通信模块和网络建立模块一起构成运行环境部件的网络功能模块，该模块提供网络通信功

能。接下来对运行环境层的各个模块提供的功能和服务进行更加详细讲解。

**执行控制器**是运行环境层中的控制模块,主要功能包括调用网络建立模块跟构件库管理部件通信获得可靠稳定网络协议的配置信息并完成将配置信息部署到运行环境部件中;调用网络通信模块同构件库管理部件进行通信获得系统构建部件描述和功能构件集合;调用系统构建部件解析模块将从构件库管理部件中获得的系统构建部件描述进行解析;可以根据系统构建部件描述解析的反馈信息指导语言部件编译模块完成功能构件的本地化编译与部署安装。

**构件运行容器**是运行环境层为 SDN 控制平面提供转发设备功能的基础,构件的安装、运行及卸载都是要在该容器中进行。构件运行容器本质上就是一个构件运行与维护的操作系统,但是它的功能和结构比操作系统的要求低很多,实现难度和成本将会大大降低。构件运行容器应该提供的功能包括提供构件的安装功能,构件的卸载功能,构件的启动功能,构件的停止功能,这些功能的基础是可以调用底层指令系统完成构件的资源调用功能。

**系统构建部件解析模块**主要的功能是将执行控制器发送过来的系统构建部件描述解析成功能构件部署命令并反馈到执行控制器,执行控制器将按照解析出的部署命令控制语言编译模块进行工作。**语言部件编译模块**主要的功能是按照控制器的部署命令将功能构件进行本地化编译,编译成本地可执行硬件指令并调用构件运行容器进行安装部署的功能;对构件运行容器中构件的启动、停止和卸载命令的执行也在语言部件编译模块中实现,这样做的目的是可以将构件的安装、卸载、启动和停止功能放到一起进行实现和维护。**网络通信模块**负责运行环境部件到构件库管理部件之间网络通信功能的实现。**网络建立模块**负责了在网络通信模块可以正常工作之前通过利用初始协议与构件库管理部件通信获取可靠稳定协议的配置信息并在运行环境部件中进行设置的功能。

### 3.5 网络环境部件

**网络环境部件**在我们架构中是对一类为转发设备功能构建过程提供初始网络服务功能的**通信协议及相关硬件的统称**,在架构具体实现时是一种由物理网络和基本通信协议构成的网络系统,比如以太网,其中包括物理网络和以太网协议。在转发设备功能构建过程中网络环境部件的主要作用是为构件库管理部件和运行环境部件提供通信服务,一般由物理网络和基本通信协议构成的网络系统不能提供可靠稳定的数据通信能力,因此在网络环境部件的实现时提出了**两阶段协议通信模型**来解决这一问题。两阶段协议通信模型的实质就是首先利用

物理网络和基本通信协议来建立可靠稳定的高层通信协议通道，当可靠稳定的高层通信协议通道建立之后，构件库管理部件和运行环境部件将利用高层通信协议通道进行数据和控制命令的传输。比如可以利用以太网和以太网协议实现构件库管理部件和运行环境部件之间的基本通信，然后从构件库管理部件获取运行环境部件的 TCP/IP 协议配置信息并进行设置，当构件库管理部件与运行环境部件之间的 TCP/IP 协议通道建立之后将会以 TCP/IP 协议进行数据和控制命令的通信，这样可以解决以太网协议不能提供可靠数据传输的问题。

需要指出运行时为用户提供的网络和网络环境部件逻辑上并不是同一个网络，网络环境部件是可扩展转发设备架构的组成部分，而可扩展转发设备架构主要的作用是重构运行时为用户提供的网络。任何网络功能都是要基于一定的物理网络和网络端口监听发送技术才可以实现的，因此在硬件设备提供商提供运行环境部件的实例时应该提供网络端口监听发送技术接口以便网络开发人员可以快速进行网络功能的开发与部署。

### 3.6 构件库管理部件

构件库管理部件在我们的架构中是对实现了对功能构件、运行环境部件、系统构建部件描述等网络资源进行统一管理的集中控制管理系统的统称。构件库管理部件主要解决语言部件编写的构件、系统构建部件描述、网络环境部件和运行环境部件如何协作进行网络功能运行时中转发设备服务功能的构建问题，它是可扩展转发设备架构的控制中心，负责构件、系统构建部件描述、网络环境部件和运行环境部件的管理。如图 3.7 所示设备管理模块负责对运行环境部件进行管理与维护，构件管理模块负责对语言部件编写的功能构件进行管理与维护，构造件管理模块负责对系统构建部件描述文档进行管理与维护，网络管理模块负责对网络环境部件相关设置进行管理与维护。用户管理是网络维护人员执行网络管理时进行必要安全认证使用的权限管理模块，部署管理模块则是将功能构件集合和系统构建部件描述发布到运行环境部件进行实际部署时使用的功能模块，其中包括对运行环境部件中已经部署的功能构件集合的管理与维护功能。网络通信模块和网络建立模块主要负责处理网络环境部件中的初始网络建立过程和网络通信过程，在这里应该指出初始网络环境建立和网络通信过程采用了两阶段协议通信模型，接下来进行详细介绍。

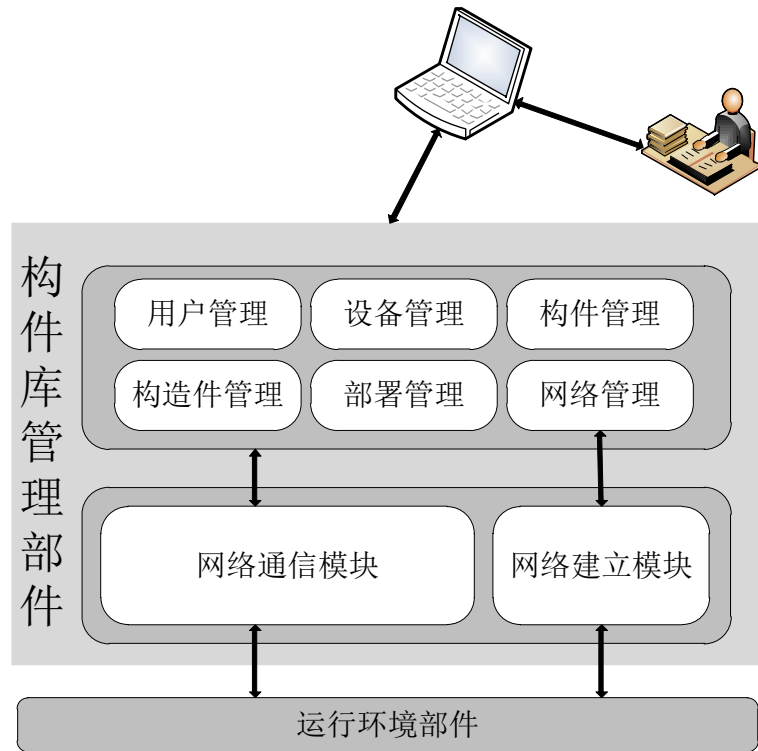


图 3.7 构件库管理部件整体结构图

Figure 3.7 The architecture of component library management element

**构件管理模块**是构件库管理部件中进行构件添加、删除、更新和查询管理维护的入口，构件管理模块可以维护着大量自己开发的功能构件和第三方经过严格验证的功能构件，这些功能构件构成了运行环境部件灵活构建网络服务的基础。当网络管理人员需要实现新的网络服务时，首先可以从构件管理模块中查看是否存在可以直接使用的功能构件集合，如果存在可以通过修改系统构建部件描述实现对运行环境部件的网络服务的重构；如果现有的功能构件集合不能满足新的网络服务需求时，网络管理员可以利用语言部件编写新的功能构件同时将构件添加到构件库部件中，然后再修改系统构建部件描述实现对运行环境部件的网络服务的重构，这样做不但可以完成新的网络服务部署，也可以使构件的重用更加方便易用。图 3.8 展示了运行环境部件利用系统构建部件描述进行网络服务重构的过程，从图中还可以看出构件库管理部件应该提供系统构建部件描述的版本控制以方便用户在需要时进行网络服务的还原。

**设备管理模块**主要是负责对运行环境部件的管理，其中包括监听新设备的接入、修改设备的基本信息、查询感兴趣的设备等功能。监听新设备的接入功能是其中比较复杂的一种，它需要网络建立模块的支持，实现流程主要包括当新的设备接入网络时会主动广播构件库管理部件查询包，构件库管理部件在接收到构件库管理部件查询包时会检查该设备是否已经在



设备管理模块进行了注册，如果没有该设备的注册信息则认为是新的设备，对该设备在设备管理模块进行注册，同时发送构件库管理部件查询包的应答包，通知该设备它已经注册成功。新接入网络的设备在没有接到构件库查询包时会一直发送查询包直到接到构件库管理部件的应答包为止，当转发设备接到应答包后会停止构件库管理查询包的发送，进入网络环境建立阶段，经过网络环境建立阶段之后便可以与构件库管理部件利用高层协议进行正常通信时便可以正常工作了。

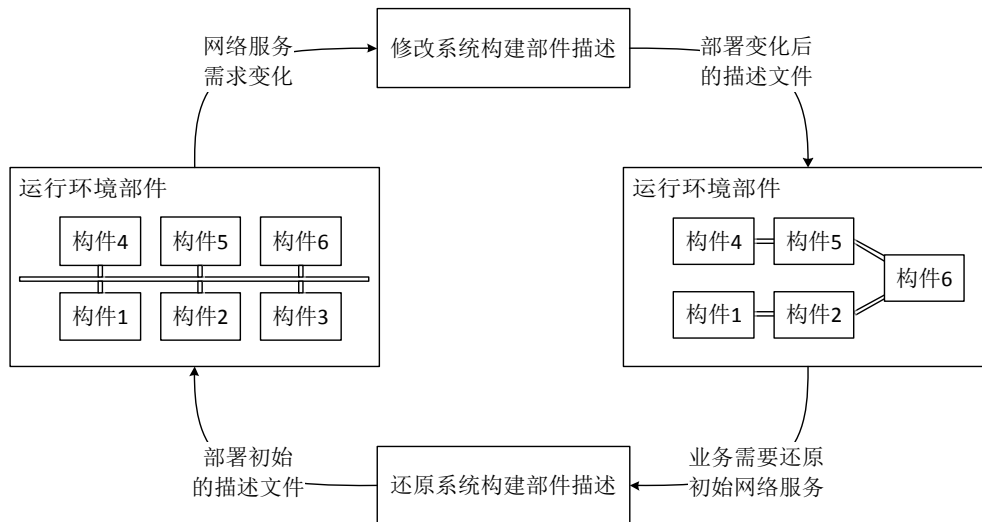


图 3.8 转发设备服务功能重构图

Figure 3.8 The restructure of FE services

**构造件管理模块**是以系统构建部件描述为管理对象，主要实现了对系统构建部件描述的添加、删除、更新和查询等功能。其中不得不特别指出的是对系统构建部件描述的版本管理功能，在此推荐的方案是系统构建部件描述的名称以构造件名称+版本号+时间戳的方式进行维护和管理以便版本控制功能的实现，即构造件管理模块可以同时维护一个网络转发设备的多个系统构建部件描述，它们通过“名称+版本号+时间戳”系统构建部件描述名称进行区分和维护。

**部署管理模块**主要负责将系统构造部件描述发布到运行环境部件中指导运行环境部件进行网络服务功能的重构，这个过程利用网络通信模块提供的通信功能将系统构建部件描述传输到运行环境部件，同时也要将系统构建部件描述中使用到的功能构件集合利用网络传输到运行环境部件中进行构件的部署工作。构件集合的传输过程可以采用两种方式进行实现，一种方式是在发送系统构建部件描述时将要使用到的构件集合同时发送到运行环境部件中，当系统构建部件描述和构件集合全部准备就绪时再进行部署安装。另一种方式是先把系统构

建部件描述发送到运行环境部件中，运行部件根据系统构建部件描述中确定的安装顺序依次通过同构件库管理部件通信下载部署安装。

**网络管理模块**主要是用来确定运行环境部件接入网络运行环境部件时如何确定网络通信模块的建立方式，一种是自动分配，另一种是网络管理员通过设备管理模块进行直接指定。在构件库管理部件和运行环境部件之间进行网络通信时一般要使用成熟稳定的网络协议才可以保证通信过程的稳定性和有效性，比如 SDN 控制器与数据平面进行通信时使用的是 TCP/IP 协议，考虑到类似情形，将类似 TCP/IP 等协议的配置信息的维护纳入了网络管理模块，其中自动配置方案是将类似于 TCP/IP 协议中地址和端口的配置信息通过网络管理模块自动进行配置；另一种是网络管理员可以通过设备管理模块对这些配置信息直接进行管理和维护。**两阶段协议通信模型**的提出正是为了解决类似于以太网的底层网络一般不提供稳定可靠的网络通信能力问题。两阶段协议通信模型的实质就是首先利用类似于以太网的底层网络通信对运行环境部件的类似于 TCP/IP 协议配置信息进行配置，完成稳定可靠通信环境的初始化后即可进入利用类似于 TCP/IP 等可靠通信协议进行构件库部件与运行环境部件之间的通信。如图 3.7 中的网络通信模块使用的是类似于 TCP/IP 等稳定可靠的网络协议，而网络建立模块使用的则是类似于以太网等没有提供可靠服务的网络协议。

**用户管理模块**主要是用来提供对网络管理员进行权限认证的，是架构中提供安全类服务开始的第一步，随着万物互联时代的启幕，网络安全的重要性正在变得越来越重要，因此在架构设计的开始就将安全性问题考虑进来，但是目前的重点仍然是解决架构的功能性问题，所以只考虑了网络管理员的权限认证问题，而没有涉及其他方面，安全相关内容将会在今后的工作中进行不断的完善和扩充。

本小节对构件库管理部件的整体架构和职责进行了比较全面的定义与描述，可以看出构件库管理部件是以构件和转发设备为基础的网络资源集中化管理系统的统称，它的正常运作是离不开语言部件、系统构建部件、运行环境部件和网络环境部件的支持与配合的。

### 3.7 本章小结

可扩展转发设备架构由语言部件、构件库管理部件、系统构建部件、运行环境部件和网络环境部件五部分，它们之间协调工作共同完成网络运行时的转发设备服务功能的构建和维护。这样在网络用户有了新的需求或者网络环境发生大的变化时都可以通过本文所提架构进

行转发设备服务功能的重构和升级，而且当新的模型或者新的技术出现后，通过丰富和完善构件库管理部件中的功能构件集就可以实现支持，可以大大减少了硬件设备因为不能适应新技术而被浪费的情况。

## 第四章 可扩展转发设备架构关键技术与原型实现

### 4.1 基于 Java 的语言部件设计与实现

语言部件和运行环境部件一起构成了本文架构的核心部件集，语言部件可以屏蔽底层硬件实现细节，运行环境部件为语言部件编写的功能构件的执行提供运行环境。语言部件的设计与实现是一件非常复杂的事情，比如 P4 语言从提出到现在一直停留在理论验证阶段，成熟稳定的编译器仍然没有出现。相对而言，传统 PC 机编程语言也可以用于转发设备功能构件的编写，同时又拥有成熟稳定的编译器，因此在该架构原型系统实现时为了简化语言部件实现的复杂性，本文采用 Java 语言作为语言部件的实例。采用 Java 语言的原因有两个，一个原因是 Java 语言有非常良好的跨平台特性，以便原型系统运行在不同系统平台上；另一个原因是 Java 语言由于其非常良好的开源特性在运行环境部件中构件运行容器的实现方面提供了成熟的框架基础（OSGI 框架）。考虑到 Java 语言并不是为了转发设备功能的开发而设计的语言，由此相对于 P4 语言，Java 语言需要网络管理人员花费更多时间学习更多语法和知识完成网络服务功能的开发，而一些网络底层网络知识和 Java 语言知识对网络管理人员来说不是必须的，因此在参考了 P4 的基础上本文提供了一套转发设备功能开发类库。

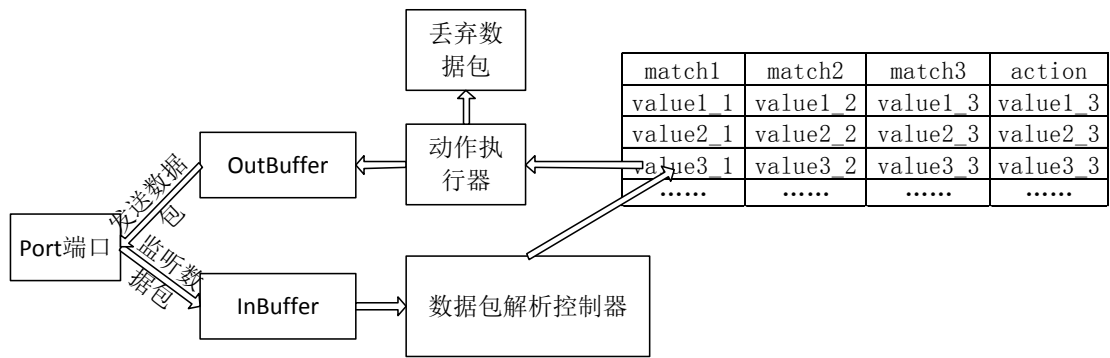


图 4.1 抽象转发设备处理流程模型

Figure 4.1 The abstract process flow model of FE

图 4.1 展示了我们基于 P4 文章中的抽象转发模型提出的抽象转发设备处理流程模型，基于该模型得出了转发设备功能开发类库中的主要功能类，Port 端口处理类主要负责监听网络中发送的数据包并将监听到的数据包放进数据包缓存里，同时还负责将该端口上注册的数据包缓存类中的缓存数据包发送到相应设备上去；数据包缓存类是存储待处理数据包的重要

数据结构，它以队列为基础实现，可以从队首获取要处理的数据包，从队尾放入最新要处理的数据包；图 4.1 中的表格是流表类的展示，流表类是存储匹配项、统计信息和动作指令集等处理信息的数据结构，提供了添加、删除、查询流表项等功能；动作执行器是对通过流表匹配之后产生的动作集进行执行的功能模块，当前阶段实现只支持丢弃和转发动作，丢弃动作将会把数据包直接销毁，转发动作会将数据包送到待发送端口的输出缓存队列里；数据包解析控制器的实现比较复杂，本文是以一个解析模块的方式进行实现的，该模块是基于扩展协议解析树模型实现的，在 4.6 小节会详细讲解。

表 4.1 展示了本文已经实现的转发设备功能开发类库，在运行环境部件实现时作为一个基础功能构件被部署于构件运行容器中。该转发设备功能开发接口库的实现底层依赖于 Pcap4J 底层协议处理程序包，Pcap4J 使用 Jna 技术封装了 WinPcap 和 LibPcap 为 Java 语言提供了操作底层网络接口的能力，为本文以太网数据包的监听和发送功能的实现提供了基础。

表 4.1 转发设备功能开发接口列表

Table 4.1 The development interface list of FE function

Port 类	端口类，负责监听网络中的数据包和发送数据包到直连的转发设备端口。
BasePacket 类	基础数据包类，在整个处理过程中负责传递数据包和动作指令集。
PacketBuffer 类	数据包缓存类，负责缓存数据包，在端口类监听到数据包和发送数据包时用来暂存数据包。
Action 类	动作指令类，存储动作指令信息，现在只包含转发和丢弃两种指令。
ActionSet 类	动作指令集类，存储动作指令列表。
ActionExecutor 类	动作指令执行器类，图 4.1 中动作执行器的实现类。
TableField 类	匹配字段类，用于记录匹配字段基本信息和组成流表项。
TableItem 类	流表项类，由匹配字段列表和动作指令列表组成，构成流表类的基础。
Table 类	流表类，图 4.1 中流表的实现类，用于记录转发规则信息，用于确定数据包如何处理。
TableMatcher 类	流表匹配处理器类，负责接受数据包解析控制器解析后的数据包并执行流表匹配动作。

## 4.2 系统构建部件设计与实现

系统构建部件是基于图论知识进行设计与实现的，可以利用 XML 格式进行描述，也可以利用 JSON 格式进行描述，在本文实现中采用了基于 XML 格式进行实现。如图 4.2 中展示的是图 3.5 中功能构件集合的系统构建描述。

```

<depgraph>
  <components>
    <component ID="element1" Level="1" ></component>
    <component ID="element2" Level="1" ></component>
    <component ID="element3" Level="1" ></component>
    <component ID="element4" Level="1" ></component>
  </components>
  <relations>
    <relation sID="element1" dID="element2"></relation>
    <relation sID="element1" dID="element3"></relation>
    <relation sID="element2" dID="element4"></relation>
  </relations>
</depgraph>

```

图 4.2 系统构建部件描述

Figure 4.2 System construction element instance

从图 4.2 中可以看出本文中系统构建部件主要由五个 XML 标签进行描述,其中<depgraph>标签表示一个系统构建部件描述的根节点,在该标签中包含着功能构件集合标签<components>和构件之间的依赖关系集合标签<relations>,由<depgraph>、<components>、<relations>三个标签功能构成了构件依赖关系图的描述框架,其中<components>标签由功能构件列表组成,<relations>标签由构件之间的依赖关系列表组成,标签<depgraph>包含一个属性 method,用于指示系统构建部件描述解析模块如何执行部署功能,它可选的参数如下表 4.2。一个功能构件利用<component>标签进行表示,component 标签包含两个属性,ID 属性是功能构件的唯一标识符,Level 属性是表示启动级别用于指示运行环境部件应该启动功能构件的顺序。构件与构件之间的依赖关系可以使用<relation>标签进行表示,其中,sID 属性表示需要依赖 dID 标识的功能构件的功能构件的 ID,dID 属性表示被 sID 标识的功能构件依赖的功能构件的 ID。

表 4.2 系统构建部件执行方式表

Table 4.2 The execution modes of system construction element

INITIAL	参数 $G_{\text{deploy}} = \langle V(G_{\text{deploy}}), E(G_{\text{deploy}}) \rangle$	指示解析器按 $G_{\text{deploy}}$ 重新部署服务功能。
ADD	参数 $G_{\text{factor}} = \langle V(G_{\text{factor}}), E(G_{\text{factor}}) \rangle$	指示解析器按 $G_{\text{factor}}$ 添加新的服务功能。
DELETE	参数 $G_{\text{factor}} = \langle V(G_{\text{factor}}), E(G_{\text{factor}}) \rangle$	指示解析器按 $G_{\text{factor}}$ 删除原来的服务功能。

系统构建部件是运行环境部件进行转发设备服务功能重构的依据,重构的过程应该包括初始化功能构件集合和依赖关系集合,维护的过程应该包括添加新的功能构件集合和依赖关系集合和删除原来不满足需要的功能构件集合和依赖关系集合。在初始化执行方式过程中功能构件集合和依赖关系集合应该是一种有效的部署方案,在添加和删除执行方式中功能构件

集合和依赖关系集合没有这一要求，因此系统构建部件描述解析模块应该提供如表 4.2 三种执行方式：

根据定义 1 可以发现在系统构建部件的描述中并没有依赖功能接口集合和提供功能接口集合，这是由于在标签中引入依赖功能接口集合和提供功能接口集合将会事标签的维护和使用变得非常困难，而且运行环境部件实现时是基于 OSGI 框架，该框架中已经提供了依赖功能接口集合和提供功能接口集合的实现说明方式，再引入依赖功能接口集合和提供功能接口集合是非常多余的。

```

Input: System Construction Element Instance;
Output: Correct/Error Msg;

Initial output message variable msg;
Read the method attribute of depgraph label, store the value in v1;
Chcek is there a loop in the instance, store the result in flagwarning;
if(flagwarning==true && v1 != INITIAL){
    msg = circle warning message; return msg;
}else if(flagwarning == false&& v1 != INITIAL){
    msg = correct message; return msg;
}
if(v1 == INITIAL){
    check is there a invalid dependency relation, store the result in flagerror;
    if(flagerror == true&&flagwarning==true){
        msg = circle warning and error message;
    }else if(flagerror == true&&flagwarning==false){
        msg = error message;
    }else if(flagerror == false&&flagwarning==true){
        msg = circle warning message;
    }else{
        msg = correct message;
    }
    return msg;
}

```

算法 1 系统构建部件正确性检查算法

基于图论知识的系统构建部件描述可以提供基本的正确性检查，因此本文提出一种正确性检查算法用于检查系统构建部件中可能存在的描述错误，其中  $G_{\text{deploy}}$  和  $G_{\text{factor}}$  检查算法是不同的，对于  $G_{\text{deploy}}$  来说描述中可以只存在功能构件节点而不存在构件之间的依赖关系，且构件之间的依赖关系中两端的功能构件必须存在，否则依赖关系是没有意义的。而对于  $G_{\text{factor}}$

来说不存在上述限制，因为  $G_{\text{factor}}$  本身是用于修改原有系统构建部件描述的内容，要添加和删除的边的两端的功能构件可能已经存在。因此该算法应该将  $G_{\text{deploy}}$  和  $G_{\text{factor}}$  分情况考虑，算法描述如算法 1 所示。算法 1 解决了系统构建部件描述最基本正确性的检查，该算法主要操作是遍历依赖关系图，时间和空间复杂度和图的遍历算法相同。

系统构建部件最重要的作用是为运行环境部件生成正确有效的构件安装序列，从而指导运行环境部件正确完成对转发设备功能的重构。由定义 4 可知每一个有效的部署方案应该是一个有向无环图，经过拓扑排序之后就可以生成一个可行的构件安装序列，但是拓扑排序没有考虑构件定义中的启动级别，因此我们基于拓扑排序提出了构件安装序列生成算法，用于生成满足要求的构件安装序列。该算法只是在删除图中顶点时首先删除可以删除顶点中启动级别最低的顶点，即添加了一步寻找可以删除顶点中启动级别最低的顶点的过程，对拓扑排序算法复杂度的影响不大，因此构件安装序列生成算法的时间复杂度和空间复杂度与拓扑排序算法相同。构件安装序列生成算法描述如算法 2 所示。

```

Input: System Construction Element Instance(SCEI);
Output: Available Installed Component List(AICL);

Initial AICL;
Find vertexs without precursor;
While (vertexs is no empty) {
    Find vertex of minimum startup level;
    Vertex insert into AICL;
    Delete vertex and the arcs which their tail is vertex;
    Find vertexs without precursor;
}
Return AICL;
    
```

算法 2 构件安装序列生成算法

## 4.3 运行环境部件设计与实现

### 4.3.1 基于 OSGI 框架的运行环境部件设计



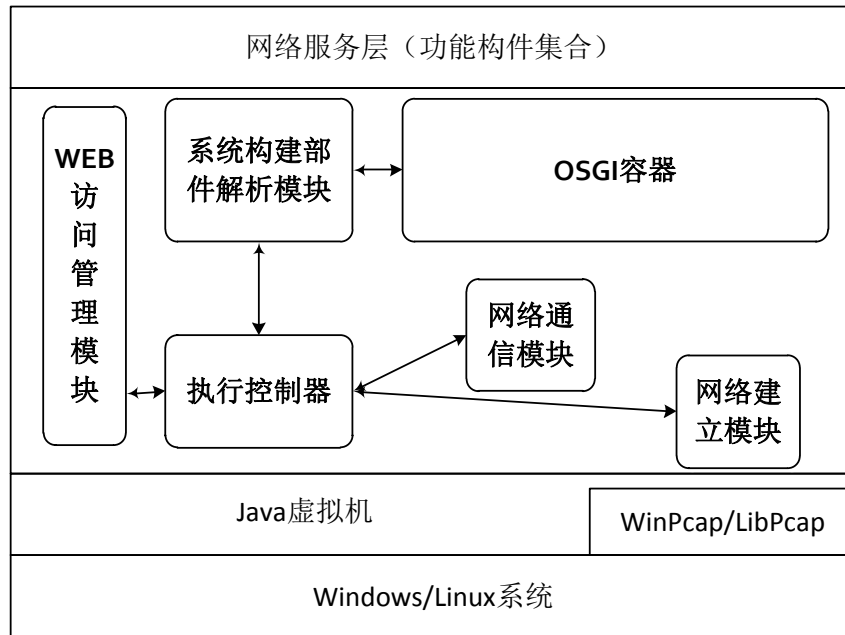


图 4.3 运行环境部件实现结构图

Figure 4.3 The implementation of runtime environment element

运行环境部件是转发设备的实例，同时也是本文所提架构的核心。为了验证本文架构中运行环境部件设计与实现的可行性，我们采用 Proof Of Concept 思想对运行环境部件进行了原型实现。可编程硬件及指令系统的实现比较困难，但是华为等厂商已经给出了非常优秀的解决方案，由此已经可以证明运行环境部件中可编程硬件及指令系统实现的可行性，所以我们将关注的重点放在了运行环境层的实现与设计上。

本文借鉴了 Open vSwitch 软件交换机的设计与实现，基于传统 PC 机设计实现了运行环境部件的原型系统。如图 4.3 所示原型系统中主要包括 OSGI 容器、系统构建部件解析模块、执行控制器模块、WEB 访问管理模块、网络通信模块和网络建立模块六部分。该原型系统的实现依赖于 Java 虚拟机、WinPcap/LibPcap 网络编程工具包和操作系统的支持，其中，WinPcap/LibPcap 网络编程工具包是 Pcap4J 底层网络访问程序包可以正常工作的基础。对比图 3.6 和图 4.3 会发现，在原型系统设计与实现时缺少了语言部件编译模块，而添加了 WEB 访问管理模块。去掉语言解析模块是因为 OSGI 框架作为构件运行容器的实例可以安装的功能构件必须是已经编译过的 Bundle 程序包，而且 Java 语言有非常好的跨平台特性，开发工具和编译环境非常多，非常方便大家进行编译和调试，因此并不需要语言编译模块的支持。添加 WEB 访问管理模块是为网络管理人员对运行环境部件进行管理维护提供了一种更加灵活的途径，但该模块不是运行环境部件必须的。

原型系统实现中执行控制器、网络通信模块、网络建立模块的功能和 3.3 小节中所规定

的功能相同；OSGI 容器正是构件运行容器的实现，负责功能构件集合的安装、卸载、启动、停止等维护工作；区别主要在系统构建部件解析模块，在原型系统实现时它负责对系统构建部件描述进行解析并通知执行控制器按照解析的安装维护指令指导 OSGI 容器安装维护功能构件集。接下来讲解具体实现部分，主要解析 OSGI 容器和系统构建部件解析模块的实现。

#### 4.3.2 基于 OSGI 框架的运行环境部件原型实现

目前 OSGI 框架是动态模块系统的事实上的工业标准，该架构一开始只是作为嵌入式设备和家庭网关的框架使用，不过它实际上可以适用于任何需要模块化、面向构件和动态部署的应用程序。Equinox 是 OSGI 框架的一个非常优秀的实现，同时也是 Eclipse 开发工具所使用的 OSGI 框架，本文在运行环境部件中 OSGI 容器的实现采用了 Equinox。Bundle 是 Equinox 管理的主要对象之一，也是 OSGI 框架动态部署安装功能实现的基础，因此首先介绍一下 Bundle 组成、Bundle 状态和 Equinox 提供的 Bundle 操作功能命令集合。

表 4.3 扩展的 MANIFEST.MF 文件属性

Table 4.3 The extended attributes of MANIFEST.MF

Bundle-Activator	Bundle 的启动器，包含 start 和 stop 方法。
Export-Package	导出的 package 声明，其它 Bundle 可以直接调用。
Import-Package	导入的 package 声明，调用其它 Bundle 开放的功能接口。
Bundle-SymbolicName	名称，一般使用类似于 Java 包路径的名字命名。
Require-Bundle	全依赖的 Bundle，一般不推荐使用。
Bundle-ClassPath	本 Bundle 的 class path，可以包含其它一些资源路径

Bundle 的本质是一个扩展了 MANIFEST.MF 文件属性的 Jar 包，扩展的 MANIFEST.MF 可以指导 OSGI 框架正确安装启动 Bundle 程序包，表 4.3 列出了主要扩展的属性及其含义。

当 Bundle 进入 Equinox 环境后，OSGI 框架会先读入 Bundle 的 MANIFEST.MF 中文件属性信息，装载相关的类和资源，解析 Bundle 之间的依赖关系，然后调用 Bundle 中 Activator 类的 start 方法启动该 Bundle 提供服务功能，最后通过调用 Bundle 中的 Activator 类的 stop 方法停止该 Bundle。Bundle 在 OSGI 框架中完整的生命周期状态转化如图 4.4 所示。

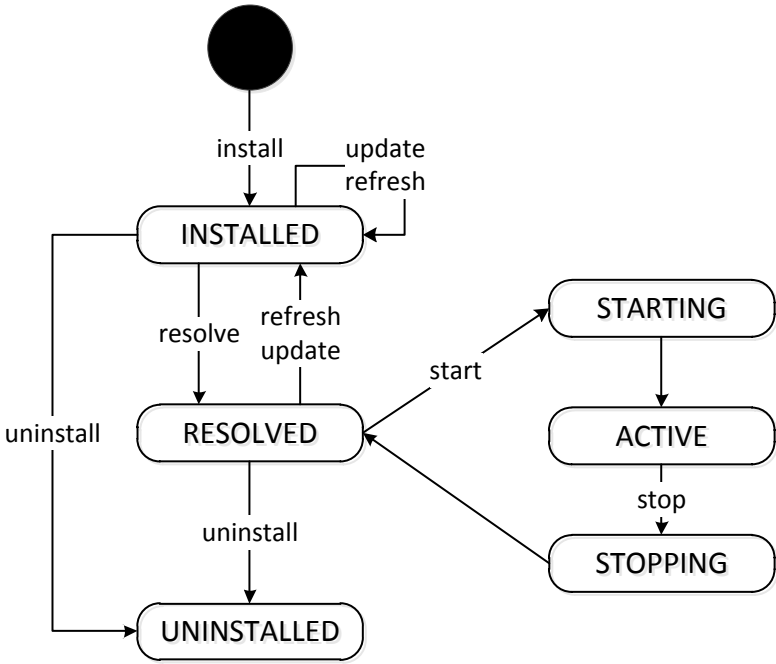


图 4.4 Bundle 状态转化图

Figure 4.4 The transition of bundle state

Equinox 不但为开发人员提供了框架级的控制命令，比如启动 OSGI 框架，关闭退出框架等，也为开发人员提供了操作 Bundle 的控制命令，开发人员可以通过 OSGI 控制台管理 OSGI 框架和 Bundle，也可以通过 Equinox 提供的开发接口编程实现对它们的管理。Equinox 中关于 Bundle 操作的控制命令主要包括控制命令如表 4.4 所示。

表 4.4 Bundle 操作命令集

Table 4.4 Bundle operation command set

Install	Bundle 的安装操作。
Uninstall	Bundle 的卸载操作。
Start	启动 Bundle 操作。
Stop	停止 Bundle 操作。
Refresh	刷新 Bundle 操作。
Update	更新 Bundle 操作。

本文利用 Equinox 系统提供的 OSGI 框架和 Bundle 开发接口实现了 OSGI 容器，该容器可以与系统构建部件解析模块相互协作功能完成容器中 Bundle 管理与维护。系统构建部件解析模块的实现使用了 DOM 技术，首先接受执行控制器通过网络通信模块获取的系统构建部件描述文档，使用 DOM 技术提取出系统构建部件描述文档中的执行方式和依赖关系图，然后根据执行方式和关系依赖图生成有效可行的安装序列进行 Bundle 的安装、卸载等工作。

下面以安装 Bundle 过程为例，展示执行控制器、系统构建部件解析模块和 OSGI 容器如何协作完成 Bundle 的管理：执行控制器接收到构件库管理部件发送过来的系统构建部件描述

文档后将该文档传递给系统构建部件解析模块进行解析，系统构建部件解析模块解析该文档后生成有效可行的 Bundle 安装序列，取出安装序列中的第一个 Bundle 信息控制执行控制器与构件库管理部件进行通信获取 Bundle 程序包，然后调用 OSGI 容器安装控制接口将该 Bundle 安装到 OSGI 容器中，最后调用 OSGI 容器的启动控制接口启动该 Bundle 提供服务，依次执行 Bundle 安装序列中其他 Bundle 直到安装部署完毕结束。

OSGi交换机管理

框架状态: ACTIVE

停止刷新

已装载的Bundles:

ID	名称	版本	状态	位置	操作
0	org.eclipse.osgi	3.7.1.R37x_v20110808-1106	ACTIVE	System Bundle	

未装载的Bundles:

位置	操作
file:/C:/Program%20Files/Apache%20Software%20Foundation/Tomcat%206.0/webapps/OsgiWebLauncher/WEB-INF/bundles/forwardingplane.jar	安装
file:/C:/Program%20Files/Apache%20Software%20Foundation/Tomcat%206.0/webapps/OsgiWebLauncher/WEB-INF/bundles/controlplane.jar	安装

操作成功

图 4.5 运行环境部件原型系统效果图

Figure 4.5 Prototype system of runtime environment element

网络通信模块和网络建立模块实现与构件库管理部件实现相同，使用 Netty 技术和 Pcap4J 技术。如图 4.5 展示了本文原型系统的 WEB 管理界面效果图。需要指出的是本文运行环境部件实现时将语言部件实现的转发设备功能开发类库和基于扩展协议解析树模型的解析模块作为基础构件安装在了 OSGI 容器中。

4.4 网络环境部件设计与实现

网络环境部件是构件库管理部件与运行环境部件进行通信的网络基础，在当前 SDN 中控制平面与数据平面的通信网络已经为本文架构的网络环境部件提供了具体实现。目前 SDN 中控制平面与数据平面的通信网络主要是基于以太网进行实现的，可靠稳定的高层通信协议可以采用 TCP/IP 协议，用于提供构件库管理部件与运行环境部件之间的通信。本文通过在控制器中部署已经成熟的 DHCP 服务动态为运行环境部件提供网络配置信息，同时网络管理员也可以手动配置这些信息。在解决了构件库管理部件与运行环境部件之间的基本通信问题之后，

网络环境部件要解决的另一个问题是解决系统构建部件描述和功能构件在运行环境部件与运行环境部件之间如何传输的问题。本文通过扩展 OpenFlow 协议实现系统构建部件描述文件和功能构件传输控制命令的传递，然后 Netty 技术在构件库管理部件与运行环境部件建立文件传输通道负责系统构建部件描述文件与功能构件的传输。

表 4.5 扩展 OpenFlow 协议消息表

Table 4.5 The messages of extended openflow protocol

SEND_CON 消息	向指定设备发送系统构建部件描述命令。
REPLY_CON 消息	确认系统构建部件描述正常接受或失败。
SEND_COM 消息	向构件库管理部件请求发送功能构件。
REPLY_COM 消息	向构件库管理部件确认发送功能构件是否成功。

表 4.5 描述了本文扩展的 OpenFlow 协议消息，图 4.6 展示了扩展的 OpenFlow 协议消息格式。

```
enum ofp_type {
    OFPT_CON_SEND=38; /*发送系统构建部件描述控制命令消息*/
    OFPT_CON_REPLY=39; /*确认发送系统构建控制命令消息*/
    OFPT_COM_SEND=40; /*发送构件请求控制命令消息*/
    OFPT_COM_REPLY=41; /*确认发构件请求控制命令消息*/
}

struct ofp_con_send{
    uint32_t    mid; /*发送系统构建部件描述控制命令 Id*/
    uint16_t    length; /*系统构建部件名称长度*/
    uint8_t     data[length]; /*系统构建部件名称，数据长度由 length 值确定*/
};

struct ofp_con_reply{
    uint32_t    mid; /*响应发送系统构建部件描述控制命令 Id*/
    uint32_t    type; /*确认消息类型，当前 0 代表请求处理失败，1 代表请求处理成功*/
};

struct ofp_com_send{
    uint32_t    mid; /*请求功能构件控制命令 Id*/
    uint16_t    length; /*功能构件名称长度*/
    uint8_t     data[length]; /*功能构件名称，数据长度由 length 值确定*/
};

struct ofp_com_reply{
    uint32_t    mid; /*响应请求功能构件控制命令 Id*/
    uint32_t    type; /*确认消息类型，当前 0 代表请求处理失败，1 代表请求处理成功*/
};
```

图 4.6 扩展 OpenFlow 协议消息格式

Figure 4.6 Message format of extended openflow protocol

参照图 4.6 对网络环境部件提供系统构建部件描述与功能构件的传输过程进行详细描述。首先，构件库管理部件使用 OpenFlow 协议通道向需要重构的运行环境部件发送 SEND\_CON 消息，消息包括消息 Id 和系统构建部件描述的名称；运行环境部件接收到 SEND\_CON 消息后解析出其中包含的消息 Id 和系统构建部件描述的名称，然后使用 Netty 主动与构件库管理部件建立起文件传输通道并把待传输的系统构建部件描述的名称发送给构件库管理部件，构件库管理部件使用文件传输通道把系统构建部件描述发送到运行环境部件；运行环境部件在接收完描述文件之后发送 REPLY\_CON 消息给构件库管理部件确认处理成功或失败，其中消息 Id 和 SEND\_CON 消息 Id 必须一致。系统构建部件描述被成功解析之后会生成构件安装序列指导运行环境部件进行功能构件的安装，在这个安装过程中运行环境部件会先使用 Netty 主动与构件库管理部件建立起文件传输通道，然后使用 OpenFlow 协议通道发送 SEND\_COM 消息，消息包括消息 Id 和功能构件的名称，构件库管理部件在接收到 SEND\_COM 消息后解析出功能构件和消息 Id，使用文件传输通道将功能构件发送到运行环境部件；运行环境部件在接收完构件之后发送 REPLY\_COM 消息给构件库管理部件确认处理成功或失败，其中消息 Id 和 SEND\_CON 消息 Id 必须一致。

## 4.5 构件库管理部件设计与实现

### 4.5.1 构件库管理部件的详细设计

构件库管理部件是功能构件和网络设备等资源的管理平台，如图 4.7 所示，本文实现的构件库管理部件实例由三个层次构成，其中第一层是 WEB 访问接口层，为网络管理用户提供用户管理、设备管理、构件管理、构造件管理、部署管理和网络管理的访问接口。第二层是构件库管理部件的核心模块，其中协调控制器负责接受 WEB 访问接口层提交的服务请求并调用相应处理模块进行具体处理，在相应功能模块将服务请求处理完毕之后返回服务处理响应消息到 WEB 访问接口层。协调控制模块还负责为各功能模块提供网络通信功能，用于实现对运行环境部件的通信与管理功能。第二层中的功能模块才是构件库管理部件的管理核心，其中用户管理模块、设备管理模块、构件管理模块、构造件管理模块、部署管理模块和网络管理模块六个模块。用户管理模块主要是维护管理员的基本信息，对管理人员管理网络设备的合法性进行验证；设备管理模块主要负责维护该构件库管理部件当前可以访问的转发

设备的基本信息；构件管理模块主要是维护已经存在的功能构件集合，实现对功能构件的添加、删除、修改和查找功能；构造件管理模块主要是负责维护系统构建部件描述文件集合，实现对系统构建部件描述文件的添加、删除、修改和查找功能；部署管理模块主要负责网络管理员利用系统构建部件描述和功能构件集对指定的转发设备进行服务功能的初始化或者维护重构，该模块会调用网络通信模块与运行环境部件通信协同完成此功能；网络管理模块主要负责设置接入构件库管理部件的转发设备初始网络配置信息的基本信息和获取方式，比如 IP 地址获取是自动获取还是手动配置。构件库管理部件涉及到对功能构件、设备信息等基本资源的持久化管理因此引入持久化模块来实现对上述各种资源的持久化功能的支持，六个功能模块可以利用持久化模块实现对功能构件、设备基本信息等资源的持久化存储。第三层是通信模块层主要为构件库管理部件和运行环境部件之间的通信提供网络服务。

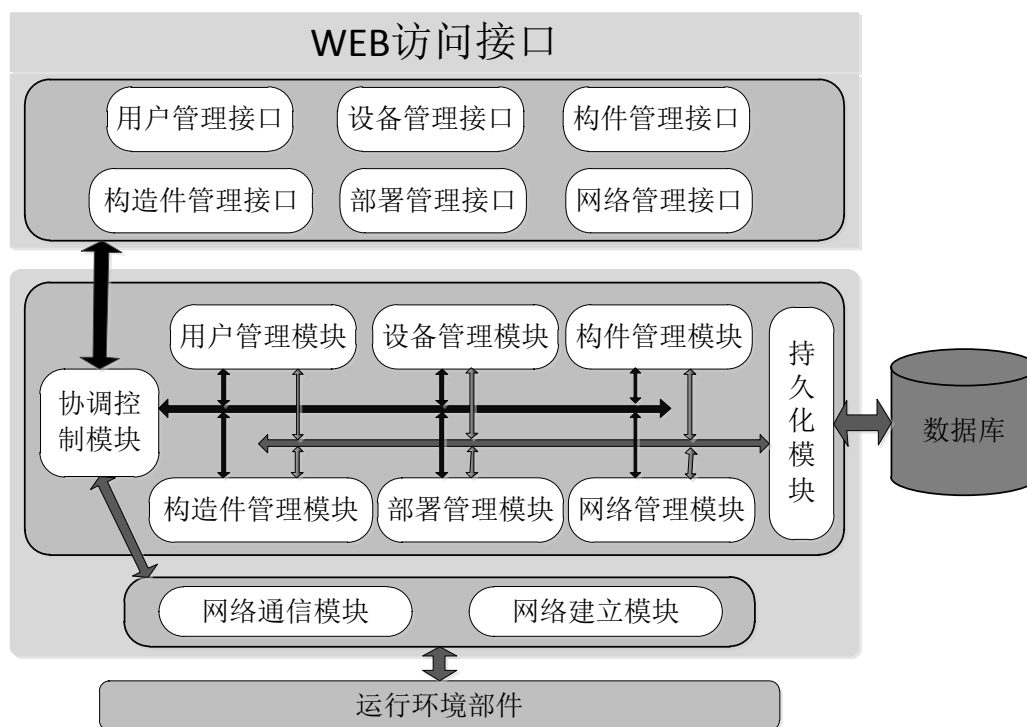


图 4.7 构件库管理部件实例结构图

Figure 4.7 The structure of component library management element instance

#### 4.5.2 构件库管理部件的原型实现

如 4.8 所示为构件库管理部件设计中的核心数据结构图，其中，构件类主要用于记录功能构件的基本信息，设备类主要用于记录转设备的基本信息，构建组件类主要用于记录系统构造部件描述文件的基本信息，设备与构建组件绑定类主要用于记录系统构造部件描述文件

与转发设备之间的发布关系。在这些主要数据结构基础上本文对构件库管理系统进行原型系统实现。

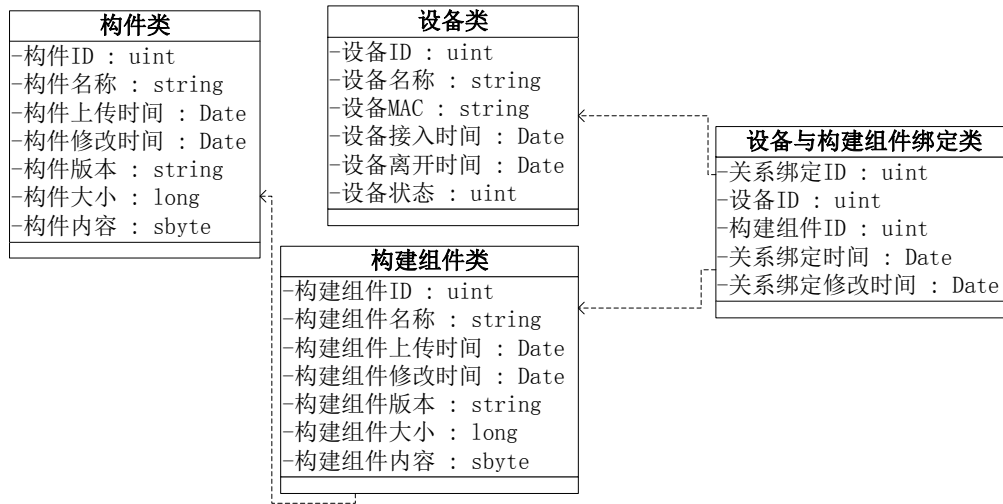


图 4.8 构件库管理部件关键数据结构类

Figure 4.8 The key data structure of component library management element

使用 JSP 技术和 Ajax 技术实现了构件库管理部件的 WEB 访问接口层，然后基于 Struct 技术和 Spring 技术实现了中间第二层中协调控制模块的服务请求路由和响应功能，用户管理模块、设备管理模块、构件管理模块、构造件管理模块、部署管理模块和网络管理模块六个功能模块用业务 Java Bean 方式实现，由此可以利用 Spring 提供的依赖注入功能将六个功能模块注入到 Spring 容器中提供相应服务功能，以方便各个功能模块的维护与扩展。持久化模块的实现使用了 Hibernate 技术进行实现，数据存储的数据库使用了 MySQL 数据库。通信模块层中网络通信模块使用 Netty 技术实现，网络建立模块使用 Pcap4J 技术实现，其中，Netty 是由 JBOSS 提供的一个开源框架，主要提供异步的、事件驱动的网络应用程序开发框架和工具，可以用来快速开发高性能、高可靠性的网络服务器程序和客户端程序，它已经在各个领域有着非常广泛的应用，比如 OpenDayLight 控制器的实现就使用了该技术。

需要说明的是功能构件集合与系统构建部件描述文件的存储方式，运行环境部件实现时构件运行容器是使用 OSGI 框架实例 Equinox 进行实现的，因此每个转发设备功能构件应该是已经封装好的 Bundle 程序包，其实就是在扩展了的 Jar 包。系统构建部件在实现时是使用 XML 标签进行实现的，因此每个系统构建部件描述本质就是一个 XML 文档。在原型系统实现时是以二进制 Blob 类型进行存储的，在填写功能构件和系统构建部件描述文档基本信息同时要上传 Jar 包和 XML 文档。



下面通过构件库管理部件原型系统中的构件管理、转发设备管理和统构建部件管理这三个模块管理界面效果图展示本文实现的原型系统。

如图 4.9 展示的是构件库管理部件原型系统的功能构件管理界面效果图。

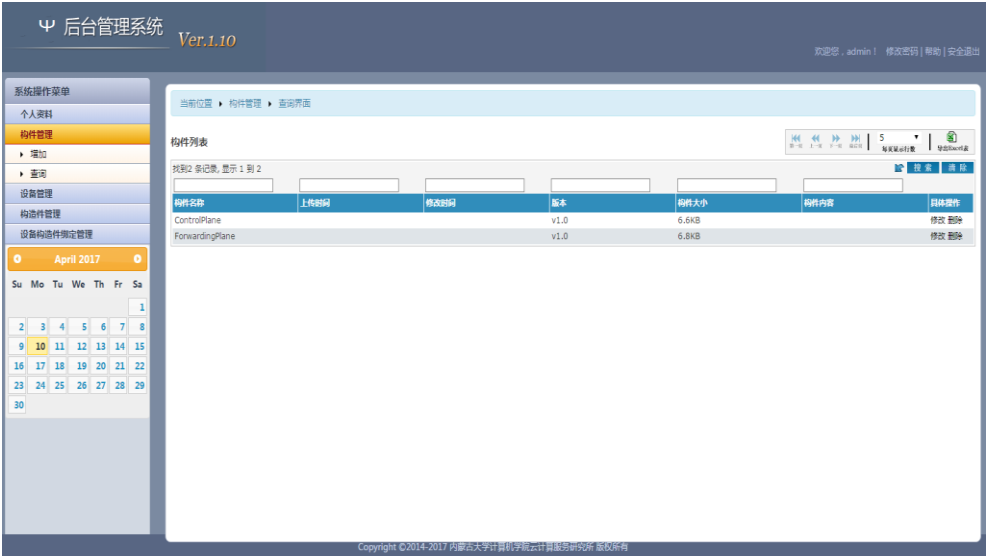


图 4.9 构件管理界面效果图

Figure 4.9 Component management screenshot

如图 4.10 展示的是构件库管理部件原型系统的转发设备管理界面效果图。

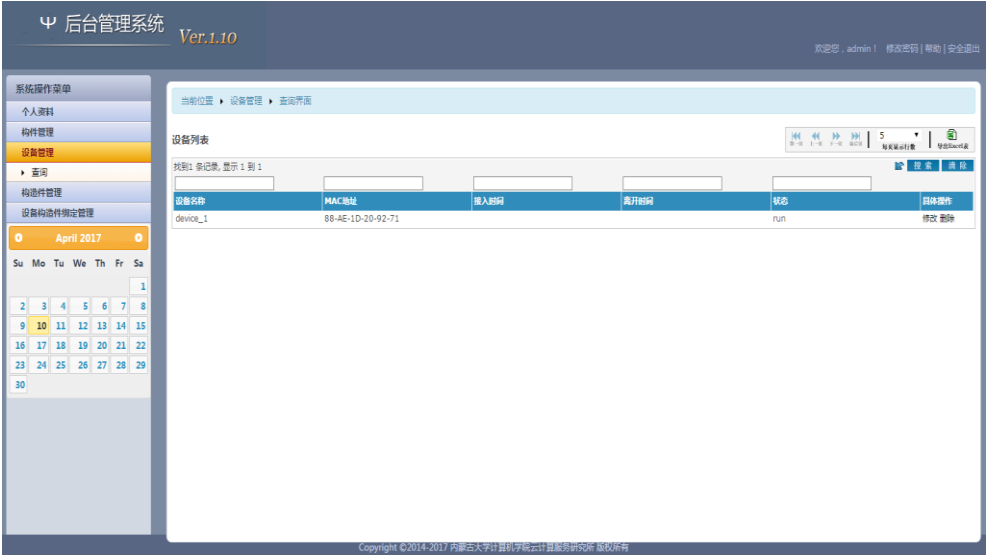


图 4.10 转发设备管理效果图

Figure 4.10 FE management screenshot

如图 4.11 展示的是构件库管理部件原型系统的系统构建部件管理界面效果图。



图 4.11 系统构建部件描述管理界面效果图

Figure 4.11 System construction element instance management screenshot

4.6 基于扩展协议解析树模型解析模块设计与实现

4.6.1 扩展协议解析树模型

**定义 1.** 在网络协议解析过程中，一些协议字段或者协议的解析依赖于其它协议字段的内容值，这种依赖关系称为协议解析依赖关系。

**定义 2.** 根据定义 1 将网络协议字段分为两类：一类是不影响其它字段和协议解析的协议字段，一般负责描述协议信息内容，该类字段定义为协议内容节点；另一类是协议字段描述的内容取值影响其它字段或协议的解析，该类字段定义为协议判定节点。

**定义 3.** 为了在协议解析过程中保留协议基本信息，本文引入了协议根节点。

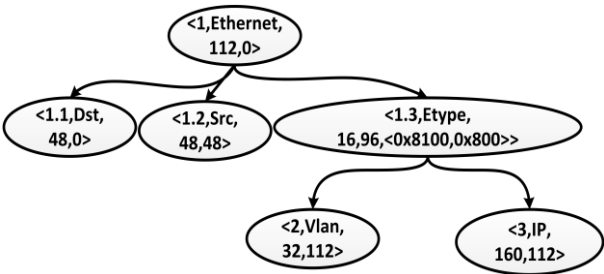


图 4.12 Ethernet 协议的扩展协议解析树

Figure 4.12 EPPTM of Ethernet protocol

如图 4.12 中，以 P4 中的 Ethernet 协议为例，Etype 字段的内容取值决定了下一步要解析的协议，是一个协议判定节点。Src 字段不影响其它字段或协议的解析，是一个协议内容节点。

协议根节点是某种协议的起始节点，标记一种协议，协议根节点是一种虚拟节点，在协议中没有对应字段，如图 4.12 中 Ethernet 节点。

**定义 4.** 扩展协议解析树节点是由协议根节点、协议内容节点和协议判定节点三部分组成，其中：

(1) 协议根节点是某种协议的起始节点，同时也是一种虚拟节点，在协议中没有对应字段。定义表示为<sequence,name,length,pre-length,childlist>五元组。其中：

1) sequence: 表示该协议在扩展协议树中的标记，可以唯一标识扩展协议树中的一个协议节点；

2) name: 表示该协议的名称；

3) length: 表示该协议头的一般长度，不包括可选项字段；

4) pre-length: 表示从起始协议的协议根节点到达该协议根节点时的总长度；

5) childlist: 表示子协议节点列表。

协议根节点只是用来描述该协议的基本信息，在协议中没有对应具体协议字段。如图 4.12 中的 Ethernet 节点即为协议根节点。

(2) 协议判定节点定义表示为

<sequence,name,length,pre-length,<<v1,child1>,<v2,child2>,<v3,child3>,...,<vn,childn>>>五元组。其中：

1) sequence: 表示该字段在扩展协议树中的标记，可以唯一标识扩展协议树中的一个协议判定节点；

2) name: 表示该字段的名称；

3) length: 表示该字段所占长度；

4) pre-length: 表示从起始协议的协议根节点到达该协议判定节点时的总长度；

5) 内容值与子节点对列表

<<v1,child1>,<v2,child2>,<v3,child3>,...,<vn,childn>>: 表示该节点的子节点应该如何解析。

如图 4.12 中 Etype 节点即为协议判定节点。

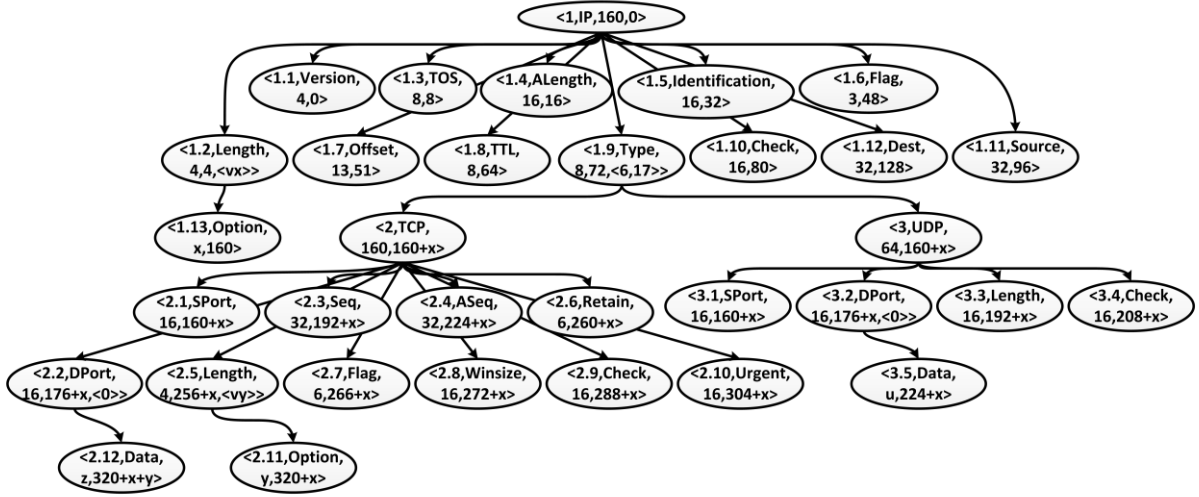


图 4.13 IP、TCP 和 UDP 组成的扩展协议解析树

Figure 4.13 EPPTM consisting of IP, TCP and UDP

(3) 协议内容节点定义表示为 $\langle \text{sequence}, \text{name}, \text{length}, \text{pre-length} \rangle$ 四元组。其中:

- 1)  $\text{sequence}$  表示该字段在扩展协议树中的标记, 可以标识扩展协议树中的一个协议内容节点;
- 2)  $\text{name}$  表示该字段的名称;
- 3)  $\text{length}$  表示该字段所占长度;
- 4)  $\text{pre-length}$  表示从起始协议的协议根节点到达该内容协议节点时的总长度。

如图 4.11 中 Src 节点即为协议内容节点。

**定义 5.** 扩展协议解析树是由协议根节点、协议判定节点和协议内容节点按照协议数据包格式的协议解析依赖关系组成一棵父子关系明确的树形协议组织结构。

本文之所以称为扩展协议树, 是相对于与网络协议解析工作中存在的协议树而言的, 协议树<sup>[30]</sup>的概念是多个协议构成的树形结构。在扩展协议解析树形协议模型中, 各种节点中存储的内容是字段信息和协议信息。如图 4.13 所示的扩展协议解析树是由 IP、TCP 和 UDP 三种协议构成的, 而当 IP、TCP 和 UDP 每个协议独立存在时, 每个协议也可以表示为一棵扩展协议解析树。

当前仍然存在两个问题需要解决, 第一个问题是如何生成扩展协议解析树。第二个问题是有些协议字段长度可能只有在解析时才能确定, 如 IP 协议的数据字段。第一个问题的解决是通过扩展协议解析树模型描述语言对相关协议进行描述, 然后由解析器自动生成扩展协议解析树。在 4.6.2 小节中会详细介绍本文提出的语言。针对第二个问题, 扩展协议解析树中引入  $\text{length}$  函数、 $\text{prelength}$  函数和基于两个函数的长度表达式, 可在解析时计算长度动态变化的

协议字段的长度值。`length` 及 `prelength` 函数均以 `sequence` 为参数，可以获取节点 `length` 属性的值和 `prelength` 属性的值。如图 4.13 中 UDP 协议的 Data 字段长度可以通过 UDP 的 Length 字段的值减去数据首部长度，由此可以得到 UDP 协议的数据字段长度表达式为 `length(3.3)-(prelength(3.5)-prelength(3.1))`。

#### 4.6.2 扩展协议解析树描述语言

扩展协议解析树模型描述语言主要由 `<protocolfiles>`、`<protocolfile>`、`<logicdev>`、`<ruledev>`、`<protocolinspire>` 五个 XML 格式的描述标签组成。当前规定协议字段长度描述采用的单位是 bit。下面将对这五个标签属性和功能进行详细介绍。其中 `<protocolfiles>` 协议列表标签，是一个辅助标签，在其中可以列出多个协议的描述信息。

`<protocolfile>` 标签主要用于描述协议根节点，它的属性包括 `sequence`、`name`、`headerlength` 和 `prelength`。`sequence` 用来描述协议在扩展协议树中的唯一标识符，`name` 用来描述协议的名称，`headerlength` 用来描述协议头长度或该协议数据包长度，`prelength` 用来描述从起始协议达到该协议根节点时的总长度。

`<logicdev>` 标签主要用于描述协议判定节点，它的属性包括 `sequence`、`name`、`prelength` 和 `value`。其中 `sequence` 属性用于描述协议字段的唯一标识符，`name` 属性用于描述协议字段名称，`prelength` 属性用于描述从起始协议的协议根节点到达该协议判定节点时的总长度。`value` 属性用于描述判定条件内容值。本标签在一个协议描述中允许出现多条，解析引擎在执行时会根据描述信息自动生成内容值与子节点的对应列表。

`<ruledev>` 标签主要用于描述协议内容节点，它的属性包括 `sequence`、`name`、`prelength`。其中 `sequence` 属性用于描述协议字段的唯一标识符，`name` 属性用于描述协议字段名称，`prelength` 属性用于描述从起始协议的协议根节点到达该内容协议节点时的总长度。

`<protocolinspire>` 标签是一个辅助标签，用于协议解析时找到下一个协议。通过该标签可以使各个协议的描述可以独立进行。它的属性只有 `sequence`，用于描述下一个协议的唯一标识符。

图 4.14 中介绍了扩展协议解析树模型描述语言对 IP 协议扩展协议解析树的描述。从中可以发现 `sequence` 值为 1.9 的 `<logicdev>` 标签使用中出现了三次，第一次出现的标签的作用是通知扩展协议解析树引擎按多少 bit 截取该判定字段，第二次和第三次出现的 `<logicdev>` 标签主

要用于生成内容值与子节点对应列表。

```
<protocolfile sequence="1" name="ip" headerlength="160" prelength="0">
  <ruledev sequence="1.1" name="Version" prelength="0">4</ruledev>
  <ruledev sequence="1.2" name="HeaderLength" prelength="4">4</ruledev>
  <ruledev sequence="1.3" name="Tos" prelength="8">8</ruledev>
  <ruledev sequence="1.4" name="ALength" prelength="16">16</ruledev>
  <ruledev sequence="1.5" name="FragIdentifier" prelength="32">16</ruledev>
  <ruledev sequence="1.6" name="FragmentFlags" prelength="48">3</ruledev>
  <ruledev sequence="1.7" name="FragmentOffset" prelength="51">13</ruledev>
  <ruledev sequence="1.8" name="Ttl" prelength="64">8</ruledev>
  <logicdev sequence="1.9" name="Type" prelength="72">8</logicdev>
  <ruledev sequence="1.10" name="Checksum" prelength="80">16</ruledev>
  <ruledev sequence="1.11" name="SourceIPAddress" prelength="96">32</ruledev>
  <ruledev sequence="1.12" name="DestIPAddress" prelength="128">32</ruledev>
  <logicdev sequence="1.9" name="Type" value="6">
    <protocolinspire sequence="2">tcp_header.xml</protocolinspire>
  </logicdev>
  <logicdev sequence="1.9" name="Type" value="17">
    <protocolinspire sequence="3">udp</protocolinspire>
  </logicdev>
</protocolfile>
```

图 4.14 IP 协议的扩展解析树描述语言描述

Figure 4.14 EPPTDL description of IP

#### 4.6.3 基于扩展协议解析树模型解析模块设计与实现

解析器的主要工作是根据扩展协议解析树模型中包含的协议头格式信息进行协议数据包的解析，数据包解析完毕后一般要做进一步的处理，如 SDN 交换机会根据协议头信息匹配转发表决定该数据包是转发还是丢弃。因此解析器要提供对解析后处理能力的支持功能，在解析器设计中引入了事件触发机制，可以在某个数据包头或字段解析完毕后触发该协议或字段上注册的处理事件执行对数据包的进一步处理。处理事件根据注册节点的不同可以分为协议处理事件和字段处理事件，其中，注册在协议根节点的处理事件称为协议处理事件，负责提供对整个协议数据包解析完毕后的进一步处理功能的支持；注册在协议判定节点和协议内容节点上的处理事件称为字段处理事件，负责提供对单个字段解析完毕后的进一步处理功能的支持。一个节点上可以注册多个事件，以此增加对数据包处理功能支持的灵活性。如图 4.15 中节点 1 上注册了一个处理事件队列。

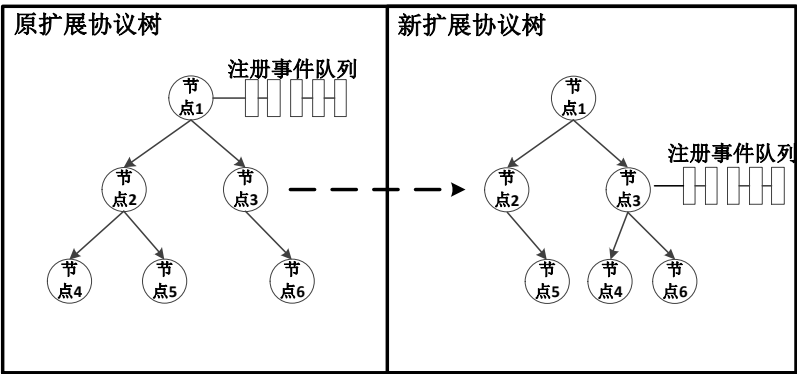


图 4.15 扩展协议解析树更新过程图

Figure 4.15 The update process of EPPTM

本文将解析器设计为解析处理模块、扩展协议解析树模型维护模块和事件处理程序维护模块三个模块。协议解析模型库将在部署机制设计部分中进行详细介绍，它在本部分为解析器提供模型和事件处理程序部署服务。下面介绍各模块的主要功能和作用，如图 4.16 所示为解析器结构模块图。

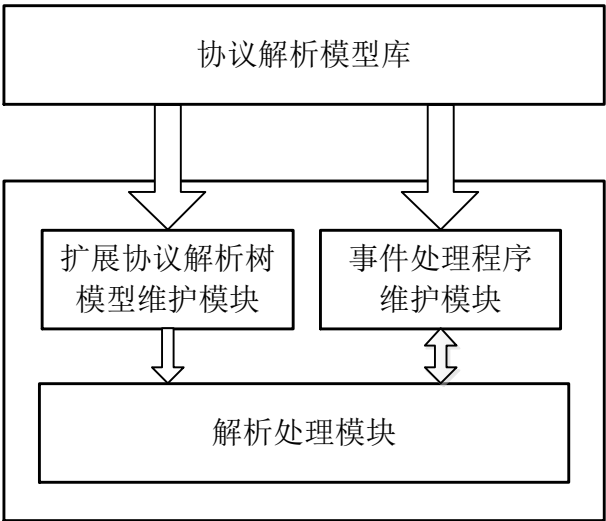


图 4.16 解析器模块结构图

Figure 4.16 The modular structure diagram of parser

解析处理模块是解析器的核心模块，它负责监听网络数据包并按扩展协议解析树模型中的协议头信息对数据包进行解析，同时检测节点上的注册处理事件是否应该触发。在扩展协议解析树模型和处理功能发生更新时解析处理模块负责获取扩展解析树模型维护模块的扩展协议解析树模型和事件处理程序维护模块中的事件处理程序组成新的工作扩展协议解析树模型提供解析服务。

为了实现在运行时快速部署协议解析更新的功能，在解析处理模块的实现上需要增加特殊的设计，使它可以同时维护两棵扩展协议解析树，一棵是当前运行使用的，另一棵是在读

取协议解析模型库中的扩展协议解析树描述时生成的临时扩展协议解析树。当协议解析模型库中的协议描述或事件处理程序发生变化时，解析处理模块一方面利用原来的扩展协议解析树继续提供解析服务，另一方面从协议解析模型库中读取变化后的协议解析描述和事件处理程序生成新的临时扩展协议解析树，在新的扩展协议解析树已生成完毕且解析处理模块按原有的扩展协议解析树解析完正在解析的数据包后，将新的临时扩展协议解析树更新为运行使用的扩展协议解析树，原有的扩展协议解析树作废，以后的数据包解析将按新的扩展协议解析树进行解析。通过这种方式实现动态更新协议解析功能，图 4.15 展示了解析处理模块中两棵扩展协议解析树模型正在进行更新的过程。

扩展协议解析树模型维护模块负责与协议解析模型库通信获取扩展协议解析树模型描述生成临时扩展协议解析树模型，同时为解析处理模块提供不带处理事件的临时扩展协议解析树模型。事件处理程序维护模块同样负责与协议解析模型库通信，获取与解析器中扩展协议解析树模型配套的事件处理命令和事件处理程序，用于操作解析处理模块中扩展协议解析树模型上注册的事件处理程序。事件处理命令包括事件处理程序查询、事件处理程序添加、事件处理程序更新和事件处理程序删除四种负责维护解析器中的事件处理程序的维护。

#### 4.6.4 配套部署机制设计与原型系统实现

利用 SDN 对转发设备的集中管理优势，我们提出了协议解析模型库来集中管理转发设备的扩展协议解析树模型和事件处理程序。协议解析模型库可以作为 SDN 控制器的子功能为转发设备提供解析处理功能部署服务，也可以作为独立的服务程序与 SDN 控制器配合提供解析处理功能部署服务，它是与解析器配套部署机制的核心。

本文的关注点是解析处理功能部署，因此可以将一个转发设备看作为一个解析设备，我们提出了如图 4.17 中的部署架构。协议解析模型库维护着在当前 SDN 网络管理域中所有解析设备的配置镜像信息，包括扩展协议解析树模型描述、事件处理命令和事件处理程序等。管理员通过协解析议模型库管理服务实现对扩展协议解析树模型描述的修改与更新，对事件处理命令和事件处理程序的查询、添加、更新和删除功能。当协议解析模型库中某个解析设备的配置镜像信息发生修改时通过 SDN 控制器通知该解析设备获取最新的配置镜像信息进行解析处理功能的更新部署。由上述分析本文将协议解析模型库设计为解析设备配置镜像信息库、配置镜像信息库管理模块和 SDN 控制器协调模块三部分组成。其中，解析设备配置镜



像信息库维护着每个解析设备的基本解析处理配置信息；配置镜像信息库管理模块则负责扩展协议解析树模型描述的更新，同时负责事件处理命令和事件处理程序的查询、添加、修改和删除等基本管理功能，完成对配置信息库信息的维护与升级；SDN 控制器协调模块负责协调 SDN 控制器通知解析设备实现对解析处理功能的更新部署。

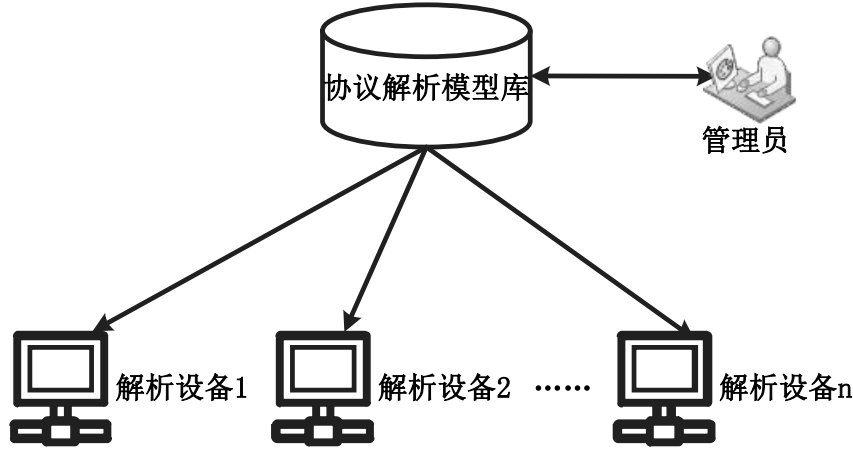


图 4.17 部署机制整体架构图

Figure 4.17 The framework of deployment mechanism

#### 4.6.5 解析模块和部署机制性能评估

为了验证本文设计的解析器及其配套部署机制的可行性，我们对解析器和部署机制进行了原型实现。考虑到本文模型的协议无关性，为了降低环境搭建的复杂度，当前解析器和部署机制的开发没有在 SDN 环境下进行，而是在传统网络环境下进行了开发与验证。原型系统的实现证明了我们设计的可行性，基于该系统本文也对解析器及其配套部署机制的性能进行了评估。

本文设计两组实验对解析器的性能进行了评估，以单位时间内处理数据包的个数为评估标准。设  $T_p$ : 解析完所有数据包花费的时间，以秒为单位， $C_p$ : 数据包的数量， $CT_p$ : 单位时间内处理数据包的个数， $C$ : 统计次数，由此可得：

$$CT_p = \frac{\sum_l^c C_p}{\sum_l^c T_p} \quad (1)$$

第一组实验主要是评估解析器的解析性能，以我们实现的传统固定解析流程解析器为比较对象。在实验中首先在真实网络环境中获取 1 万条、5 万条、10 万条、15 万条和 20 万条待解析的 IP 协议数据包，然后依次使用我们设计的解析器和传统解析器对它们进行解析并记

录解析所花费的时间。通过图 4.18 中的实验结果可以看出本文设计的解析器解析性能比传统固定解析流程解析器的解析性能稍差。通过公式（1）计算可得本文中提出的解析器的  $CT_p$  为 2756 条/秒，传统固定解析流程解析器的  $CT_p$  为 3591 条/秒。

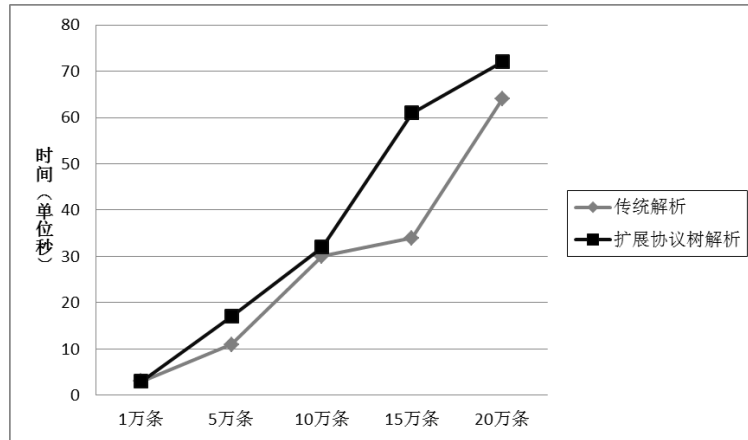


图 4.18 传统解析与扩展协议解析树解析对比

Figure 4.18 Traditional parsing and EPPTM parsing comparison

第二组实验主要评估本文提出的解析器在支持特定字段解析时可以带来的性能提升情况。在实验中，同样先在真实网络环境中获取 1 万条、5 万条、10 万条、15 万条和 20 万条待解析的 IP 协议数据包，然后依次使用本文解析器实现的选定协议字段解析功能对 IP 协议中的目的地址字段进行解析，传统固定解析流程解析器则只能先解析整个 IP 数据包然后获取目的地址字段信息，分别记录解析它们解析所花费的时间。通过图 4.19 中的实验结果可以看出在支持特定字段解析时本文解析器的解析性能相对于传统固定流程解析器的解析性能提高了很多倍。利用公式（1）计算可得本文解析器的  $CT_p$  为 25500 条/秒，传统固定解析流程解析器的  $CT_p$  为 2881 条/秒。这主要得益于扩展解析树解析方法可以很好地支持对特定字段直接进行解析，而不用等待所有字段解析完后才能获取特定字段的内容。

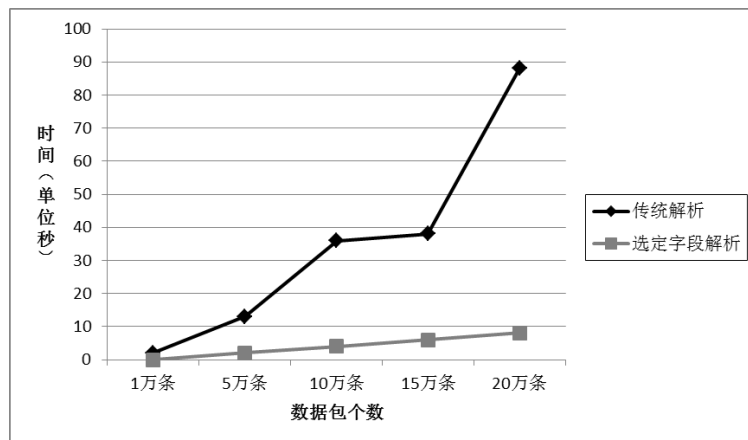


图 4.19 传统解析与选定字段解析对比

Figure 4.19 Traditional parsing and select field parsing comparison

本文设计了如图 4.20 所示实验场景下对部署机制的性能进行评估。用两台运行在实验室云平台中的虚拟机和两台实体主机模拟了四个数据包解析设备，用一台实体主机模拟了模型描述服务器。两台实体机通过交换机直接连接到模型描述服务器，两台虚拟机通过路由器和交换机连接到模型描述服务器。在四个模拟网络数据包解析设备上安装本文设计的解析器。

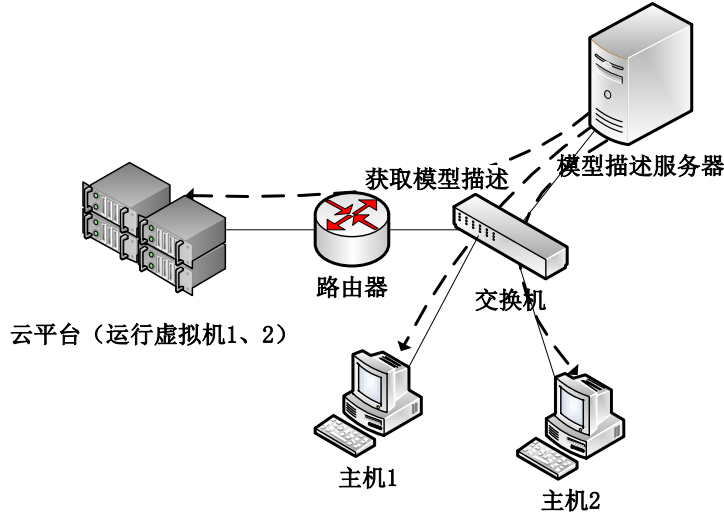


图 4.20 部署机制实验模拟场景图

Figure 4.20 Deployment mechanism experimental simulation scenario

通过分析发现，该部署机制在部署过程中主要消耗时间的地方在于通过网络获取模型描述信息部分和解析器读取模型描述信息自动生成临时扩展协议解析树部分，而从临时扩展协议解析树转换为解析时使用的扩展协议解析树的时间只涉及到执行一条命令，因此时间非常短，在与上面两部分时间对比的情况下可以忽略。设  $T_d$ : 协议部署时间， $T_n$ : 模型描述信息在网络中传输时间， $T_g$ : 协议树生成时间， $C$ : 统计次数， $T_{avg}$ : 平均协议部署时间。由此可得：

$$T_d = T_n + T_g \quad (2)$$

$$T_{avg} = \sum_{i=1}^c T_d / C \quad (3)$$

在如图 4.20 所示的实验场景情况下，为解析器添加通过网络获取模型描述信息所花费时间的记录功能和自动生成临时扩展协议解析树所花费时间的记录功能，用于统计  $T_n$  和  $T_g$ 。四个模拟解析设备同时访问模型描述服务器，记录描述信息在网络中传输时间和扩展协议解析树模型生成时间。实验执行了两次，实验结果记录如表 4.6 和表 4.7 所示。

通过实验结果可以发现该部署机制在 IP、TCP 和 UDP 三个协议构成的扩展协议解析树模型情况下可以在不到 650 毫秒的时间内完成协议的部署工作。 $T_{avg}$  作为该部署机制的平均性能评估参数，通过公式 (3) 可以计算得到  $T_{avg}$  为 498 毫秒。由此可以看出与传统手工部署

解析功能相比，该部署机制是高效实用的。

表 4.6 第一组实验结果汇总

Table 4.6 The first experimental result

机器名称	网络传输耗时	生成模型耗时
虚拟机 1	438 毫秒	63 毫秒
虚拟机 2	509 毫秒	93 毫秒
主机 1	360 毫秒	47 毫秒
主机 2	470 毫秒	71 毫秒

表 4.7 第二组实验结果汇总

Table 4.7 The second experimental result

机器名称	网络传输耗时	生成模型耗时
虚拟机 1	422 毫秒	63 毫秒
虚拟机 2	444 毫秒	61 毫秒
主机 1	375 毫秒	63 毫秒
主机 2	461 毫秒	44 毫秒

协议解析处理无关性问题是数据转发平面可扩展性问题的核心之一。本文针对该问题提出了扩展协议解析树模型及其描述语言，基于该模型及其描述语言进行了可扩展解析模块和它的配套部署机制的设计与原型系统实现。通过分析实验评估可以看出本文设计的解析模块具有良好的可扩展性，在支持特定字段解析时性能会大幅提升，同时该解析模块配套的部署机制的性能也有非常不错的表现。由此可以看出本文提出的数据包解析方案是解决 SDN 数据转发平面的协议解析处理无关性问题的一个不错的选择。实验结果也显示当前解析模块在通常情况下的解析性能仍有待提高，在未来工作中，我们将尝试通过并行处理机制提高解析模块的解析性能，完善解析模块和部署机制的实现。

## 4.7 本章小结

在本小节中对本文提出的语言部件、构件库管理部件、系统构建部件、运行环境部件和网络环境部件中涉及的部分关键技术给出了初步的解决方案并进行了原型实现。同时介绍了基于扩展协议解析树模型解析模块的设计与原型实现，其中，本文提出的基于扩展协议解析树模型的解析方法是一种用于解决网络数据包解析协议无关性的比较简单易行的解决方案。在介绍完原型系统实现的基础之上，我们在第五章将通过应用场景分析说明本文架构的优势，同时通过原型系统构建一个简单的 UUID 网络证明本文架构的可行性与灵活性。

## 第五章 应用场景分析与原型系统验证

### 5.1 应用场景分析

本文架构的本质是开放转发设备的编程能力，本小节将通过应用场景分析该架构的优势。

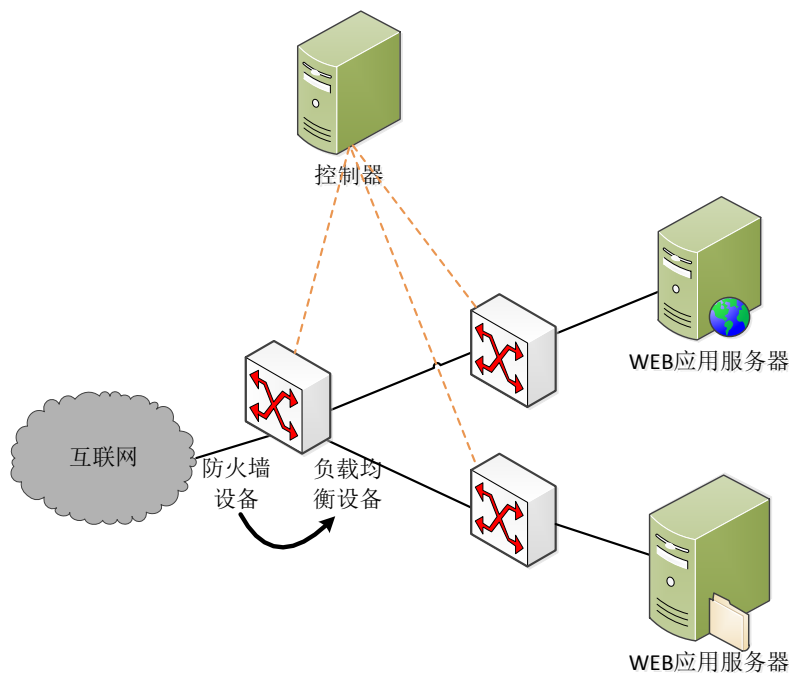


图 5.1 应用场景 1

Figure 5.1 Application scenarios 1

如图 5.1 所示场景中，与互联网直接相连的转发设备，在最开始的业务需求是一台防火墙设备，用来屏蔽恶意攻击，但是随着业务的发展现在希望与互联网直接相连的转发设备是一台负载均衡设备。在传统网络中这种设备的替换过程是首先通知网络用户 WEB 应用服务将要暂停进行维护，然后将防火墙设备从网络中去除，再将负载均衡设备接入网络，进行测试，最后通知网络用户 WEB 应用服务可以正常使用，这个过程是非常耗时的，同时成本也是非常高昂的且有可能意味着转发设备资源的浪费，如防火墙设备也许从此之后再也不会使用。而在本文架构中这种设备功能的替换将会非常快捷方便，对业务影响更小，也不会产生转发设备资源的浪费，本文架构的替换过程是首先用语言部件开发负载均衡设备功能，然后通过构件库管理部件对与互联网直连的运行环境部件进行负载均衡设备功能的动态部署并进行测试，会影响正常应用业务的时间只可能发生在负载均衡设备功能动态部署的时候，但动

态部署时间相对于传统网络中的部署过程将是忽略不计的，而且硬件转发设备可以最大程度重用而不是浪费。由此可以看出在本文架构中单个转发设备节点转发功能是可重构的。

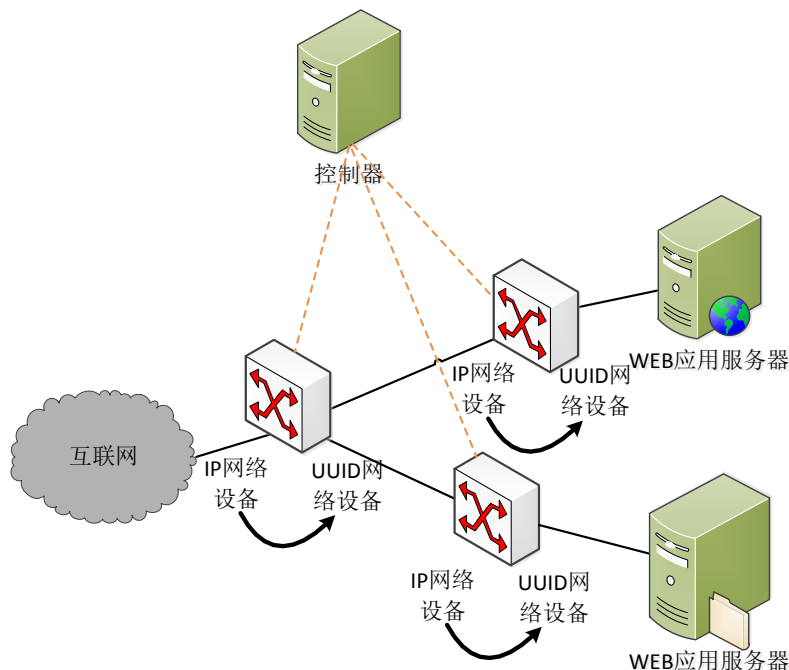


图 5.2 应用场景 2

Figure 5.2 Application scenarios 2

如图 5.2 所示场景中，某公司最初因为业务需要上了一批 IP 网络设备用于满足公司 WEB 应用服务器的网络需求，但是随着业务需求的变化现在该公司需要把网络设备升级成 UUID 网络设备。这在传统网络中进行升级将是非常困难和昂贵的，因为网络中所有设备只能丢弃而重新采购并替换成 UUID 网络设备。而在本文架构中实现这一升级将是非常方便高效的，且成本较低，通过语言构件编写 UUID 网络功能构件集合并通过构件库管理部件部署到运行环境部件中就可以完成网络环境的升级。由此可以看出本文架构中构件库管理部件管理的所有转发设备节点是可重构的，从而整个网络是可重构的。

## 5.2 原型系统部署 UUID 网络

为了验证本文架构的可行性，我们设计了一个非常简单的 UUID 网络用于计算机之间进行数据传输，UUID 协议提供了 IP 协议类似的网络标识功能。利用第四章实现的原型系统对 UUID 网络进行了快速实现与动态部署从而验证本文架构的可行性和灵活性。如图 5.3 展示了 UUID 数据包格式及每个字段的含义，UUID 协议相对于 IP 协议最大的区别是主机地址的表

示不再是 IP，而是 UUID 生成器生成的 UUID 标识符，控制器可以集中管理这些 UUID 标识符为每台连接到控制器的主机分配一个唯一的 UUID 标识符。UUID 协议和 IP 协议一样底层网络实现时是基于以太网的。

```

struct uuid_packet{
    uint8_t    version;/*协议版本信息*/
    uint8_t    flag;/*标记是否有多个数据包，0 为单个数据包，1 为多个数据包*/
    uint16_t   fragidentifier;/*当 flag 为 1 时有效，同一个 pid 数据包分片编号*/
    uint32_t   pid;/*数据包标识*/
    uint32_t   sourceuuid[9];/*源主机 UUID 地址*/
    uint32_t   destuuid[9];/*目的主机 UUID 地址*/
    uint16_t   type;/*数据包数据类型，取值见 uuid_packet_type 中取值 */
    uint16_t   length;/*数据包有效数据字节个数，包括 UUID 头信息*/
};
enum uuid_packet_type {
    UUID_CHARACTER=1;/*字符类型*/
    UUID_CHARACTER=2;/*字节流类型*/
}

```

图 5.3 UUID 数据包格式

Figure 5.3 UUID packet format

下面重点介绍如何利用第四章中的原型系统在四台 PC 机上实现了 UUID 网络，其中一台做为实现了构件库管理部件的控制器，一台作为运行环境部件实例，另外两台作为通信主机。整个拓扑图如图 5.4 所示。

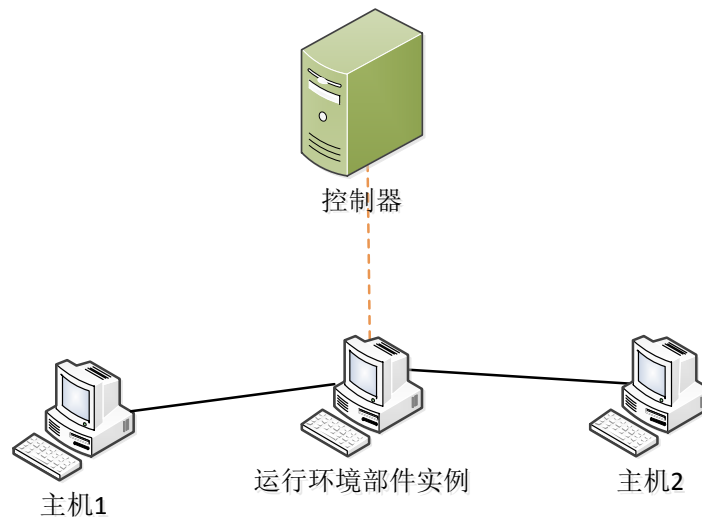


图 5.4 UUID 部署拓扑图

Figure 5.4 The topology of UUID deployment

在运行环境部件中部署 UUID 网络首先要解决的问题是数据包解析问题，本文通过扩展协议解析树描述语言对 UUID 协议进行描述，生成描述文档并通过控制器下发到运行环境部件实现对 UUID 网络协议的解析。如图 5.5 展示了 UUID 协议的扩展协议解析树语言描述。

```
<protocolfiles>
<protocolfile sequence="1" name="uuid" prelength="0">
  <ruledev sequence="1.1" name="version" prelength="0">1</ruledev>
  <ruledev sequence="1.2" name="flag" prelength="1">1</ruledev>
  <ruledev sequence="1.3" name="flagidentifier" prelength="2">2</ruledev>
  <ruledev sequence="1.4" name="pid" prelength="4">4</ruledev>
  <ruledev sequence="1.5" name="sourceuuid" prelength="8">9</ruledev>
  <ruledev sequence="1.6" name="destuuid" prelength="17">9</ruledev>
  <ruledev sequence="1.7" name="type" prelength="26">2</ruledev>
  <ruledev sequence="1.8" name="length" prelength="28">2</ruledev>
  <ruledev sequence="1.11" name="data" prelength="30">length(1.8)-30</ruledev>
</protocolfiles>
```

图 5.5 UUID 协议的扩展解析树描述语言描述

Figure 5.5 EPPTDL description of UUID

在解决了运行环境部件 UUID 数据包解析问题之后，本文使用基于 Java 的转发功能开发库设计与实现了转发功能 Bundle 和生成转发表 Bundle。其中，转发功能 Bundle 主要包括的内容有网络端口、数据包缓存和转发表的初始化、拼装数据转发工作流程，如图 5.6 展示了部分实现代码。生成转发表 Bundle 用于生成与该运行环境部件相连的 PC 主机 UUID 标识和转发端口 MAC 地址映射表，当运行环境部件接到数据包时转发功能 Bundle 会解析数据表并查找转发表进行数据包的转发，在本文中全局控制平面为空。

```
public static void switchWork() throws InterruptedException{
    while(true){
        byte[] data = ports.get(0).getInBuffer().popPacket();
        ////开始数据包解析
        SimpleParser parser = new SimpleParser();
        BasePacket packet = parser.parse(data);
        packet = parser.parse(packet);
        ////匹配流表项确定处理动作
        TableMatcher matcher = new TableMatcher();
        packet = matcher.match(tables.get(0), packet);
        ////动作命令执行器
        ActionExecutor executor = new ActionExecutor();
        executor.executeActions(packet, ports); //转发数据包
        Thread.sleep(100);
    }
}

public static void main(String[] args) throws PcapNativeException, InterruptedException {
    initPorts(100,100); //初始化交换机端口
    initTables(); //初始化流表
    switchWork();
}
```

图 5.6 数据平面 Bundle 功能部分代码

Figure 5.6 Partial code of data plane bundle

将转发功能 Bundle 和生成转发表 Bundle 存入构件库管理部件，进行系统构建部件描述



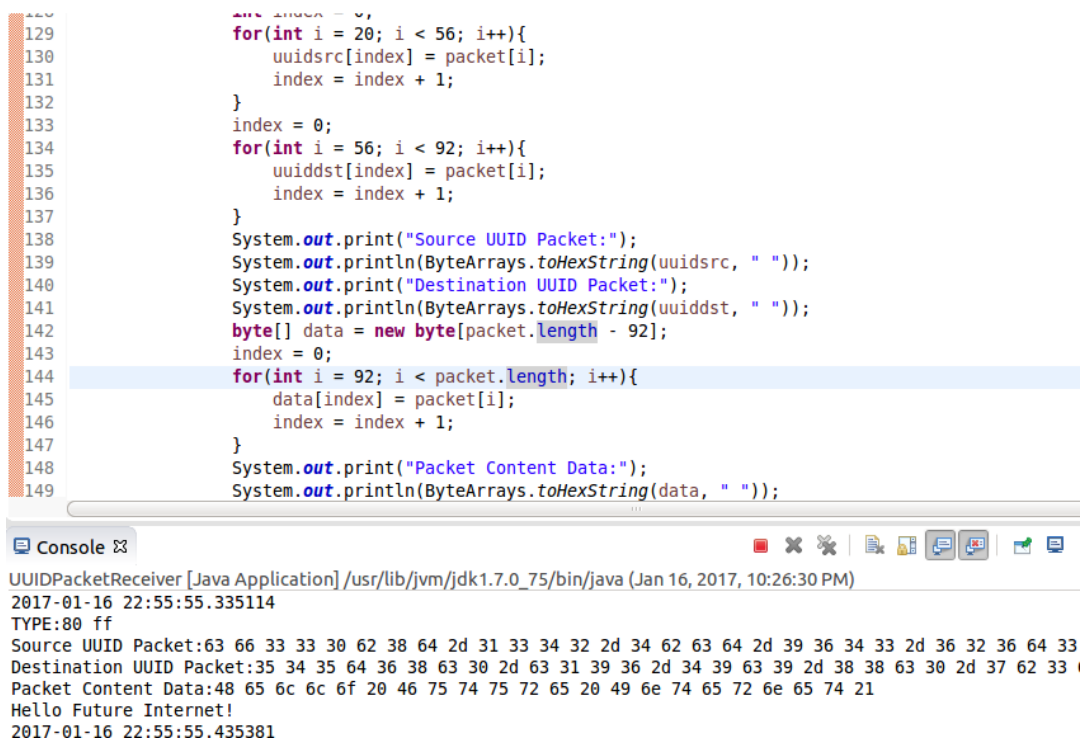
的编写，系统构建部件描述如图 5.7 所示。然后使用构件库管理部件中的操作接口对运行环境部件的转发设备功能进行部署，完成 UUID 网络转发设备的初始化。

```
<depgraph>
<components>
  <component ID="dataplane_v_1_0" Level="1" ></component>
  <component ID="controlplane_v_1_0" Level="1" ></component>
</components>
<relations>
  <relation sID="controlplane_v_1_0" dID="dataplane_v_1_0"></relation>
</relations>
</depgraph>
```

图 5.7 系统构建部件描述

Figure 5.7 The description of system construction element

在 UUID 网络转发设备初始化完成后，在两台要通信的主机上开发部署了 UUID 数据包发送和接收模块，完成通信主机端对 UUID 网络的支持，从而完成整个 UUID 原型网络的搭建，图 5.8 展示了两主机之间通信效果图。



```
129     for(int i = 20; i < 56; i++){
130         uuidsrc[index] = packet[i];
131         index = index + 1;
132     }
133     index = 0;
134     for(int i = 56; i < 92; i++){
135         uuiddst[index] = packet[i];
136         index = index + 1;
137     }
138     System.out.print("Source UUID Packet:");
139     System.out.println(ByteArrays.toHexString(uuidsrc, " "));
140     System.out.print("Destination UUID Packet:");
141     System.out.println(ByteArrays.toHexString(uuiddst, " "));
142     byte[] data = new byte[packet.length - 92];
143     index = 0;
144     for(int i = 92; i < packet.length; i++){
145         data[index] = packet[i];
146         index = index + 1;
147     }
148     System.out.print("Packet Content Data:");
149     System.out.println(ByteArrays.toHexString(data, " "));
```

```
UUIDPacketReceiver [Java Application] /usr/lib/jvm/jdk1.7.0_75/bin/java (Jan 16, 2017, 10:26:30 PM)
2017-01-16 22:55:55.335114
TYPE:80 ff
Source UUID Packet:63 66 33 33 30 62 38 64 2d 31 33 34 32 2d 34 62 63 64 2d 39 36 34 33 2d 36 32 36 64 33
Destination UUID Packet:35 34 35 64 36 38 63 30 2d 63 31 39 36 2d 34 39 63 39 2d 38 38 63 30 2d 37 62 33
Packet Content Data:48 65 6c 6c 6f 20 46 75 74 75 72 65 20 49 6e 74 65 72 6e 65 74 21
Hello Future Internet!
2017-01-16 22:55:55.435381
```

图 5.8 主机接收数据包效果图

Figure 5.8 The host receive UUID packet

### 5.3 本章小结

本小节通过应用场景分析了本文架构和 SDN 技术结合可以很好地解决数据平面可扩展

性和动态部署问题，可以支持转发设备服务功能的灵活扩展和动态部署，使之前一些难以解决的问题变得非常容易解决。为了验证本文架构的可行性，设计并原型实现了一种基于以太网的 UUID 网络，其中使用 UUID 协议取代 IP 协议作为设备唯一标识，通过实验中两个台机器利用 UUID 网络进行通信证明了本方案的可行性。

## 第六章 结论及未来工作

### 6.1 论文工作总结

本文站在网络转发设备的服务功能重构和维护的角度提出了基于 SDN 的可扩展网络转发设备架构。可扩展转发设备架构主要包括语言部件、构件库管理部件、系统构建部件、运行环境部件和网络环境部件，本文在给出了各个部件基本规范的基础上，对各个部件的部分关键技术进行了详细设计并进行了原型系统实现。其中，比较重要的内容有提出了基于图论的系统构建部件的实现方案，基于 OSGI 框架对运行环境部件进行了原型系统实现，提出了基于扩展协议解析树模型的网络数据包解析方案，同时扩展了 OpenFlow 协议使其可以支持对运行环境部件的网络服务功能进行重构。最后通过应用场景分析展示了可扩展转发设备架构在灵活扩展和动态部署方面的优势，并通过 UUID 网络功能的部署证明了该架构的可行性。

### 6.2 下一步工作

当前本文中主要关注点在于可扩展转发设备架构的设计和其中部分关键技术的原型实现上，对于性能方面的问题没有过多考虑，而且由于时间问题当前的原型实现仍然有很多问题等待改进和完善。特别是运行环境部件的实现采用了 OSGI 框架中比较成熟的基于 Java 的实现方案，该方案的好处是扩展性强，适应不同的操作系统，但是缺点也非常明显，Java 语言要运行在 Java 虚拟机上，降低了转发设备原型系统的转发性能。因此在下一步工作中，我们会不断改进和完善可扩展转发设备架构的设计，同时对运行环境部件使用 C++ 语言进行实现，目标是提供一款类似于 OpenvSwitch 的软转发设备。

## 参考文献

- [1] McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM CCR, 2008,38(2): 69–74.
- [2] Jain S, Kumar A, Mandal S, Ong J, Poutievski L, Singh A, Venkata S, Wanderer J, Zhou J, Zhou M, Zolia J, Hölzle U, Stuart S, Vahdat A. B4: Experience with a globally-deployed software defined WAN. In: Proc. of the ACM SIGCOMM. 2013. 3–14.
- [3] Doria A, Salim JH, Haas R, Khosravi H, Wang W, Dong L, Gopal R, Halpern J. Forwarding and control element separation (ForCES) protocol specification. IETF RFC 5810, 2010.
- [4] Tennenhouse DL, Wetherall DJ. Towards an active network architecture. In: Proc. of the IEEE DARPA Active Networks Conf. and Exposition. 2002. 2–15.
- [5] Tennenhouse DL, Smith JM, Sincoskie WD, Wetherall D, Minden GJ. A survey of active network research. IEEE Communications Magazine, 1997,35(1):80–86.
- [6] Greenberg A, Hjalmtysson G, Maltz DA, Myers A, Rexford J, Xie G, Yan H, Zhan JB, Zhang H. A clean slate 4D approach to network control and management. ACM SIGCOMM CCR, 2005,35(5):41–54.
- [7] Yan H, Maltz DA, Ng TSE, Gogineni H, Zhang H, Cai Z. Tesseract: A 4D network control plane. In: Proc. of the USENIX NSDI. 2007. 369–382.
- [8] 左青云,陈鸣,赵广松,邢长友,张国敏,蒋培成.基于 OpenFlow 的 SDN 技术研究[J]. 软件学报,2013, 24(5):1078-1097.
- [9] 张朝昆, 崔勇, 唐嵩祎, 吴建平.软件定义网络 (SDN) 研究进展[J].软件学报, 2015,26(1):62-81.
- [10] Open Networking Foundation. ONF SDN Evolution. 2016.  
[https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-535\\_ONF\\_SDN\\_Evolution.pdf](https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-535_ONF_SDN_Evolution.pdf)

- [11] Open Networking Foundation. OpenFlow switch specification, version 1.5.1. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf> . 2015.
- [12] Haoyu Song. Protocol-oblivious forwarding: unleash the power of SDN through a future-proof forwarding plane, Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, 2013, page:127-132.
- [13] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, David Warlker. P4: programming protocol-independent packet processors, ACM SIGCOMM Computer Communication Review, July, 2014. Vol 44, page:87-95
- [14] R Narayanan, G Lin, AA Syed, S Shafiq, F Gilani. A Framework to Rapidly Test SDN Use-Cases and Accelerate Middlebox Applications. Local Computer Networks (LCN), 21-24 Oct. 2013 IEEE 38th Conference on. pp763-770.
- [15] XinLei Hu, XianRong Wang, Lu Wang, Hua Li. Design of Extensible Forwarding Element Architecture and Its Key Technology Verification, Proceedings of 2016 IEEE International Conference on Integrated Circuits and Microsystems, November, 2016, page:214-218.
- [16] 黄韬, 刘江等, 软件定义网络核心原理与应用实践, 北京: 人民邮电出版社, 2014.
- [17] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. "NetITGA: reusable router architecture for experimental research." Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow, ACM, 2008.
- [18] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. L. F. Kaashoek. The Click modular router. ACM Transactions on Computer Systems, 18(3):263-297, Aug. 2000.
- [19] M. Handley, O. Hodson, E. Kohler, "XORP: an Open Platform for Network Research" . ACM SIGCOMM Comp. Commun., vol. 33, issue 1, Jan. 2003.
- [20] Xie, Gaogang, et al. "PEARL: a programmable virtual router platform". Communications Magazine, IEEE 49 J (2011): 71-77.
- [21] ChinaByte. ENP—重定义以太转发技术. <http://it.chinabyte.com/444/12789944.shtml>. 2013.
- [22] Linux foundation collaborative projects. Open vSwitch. <http://openvswitch.org/>. 2016.

- [23] 周通,支持多协议的 SDN 交换机的设计与实现[D],北京:北京邮电大学,2013.
- [24] Open Networking Foundation. OpenFlow switch specification, version 1.3.0. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>. 2012.
- [25] 李华,叶新铭,吴承勇,王龙.下一代网络协议测试数据半自动生成方法研究[J].计算机科学,2008,35(12):102-105.
- [26] Fulvio Rizzo,Mario Baldi. NetPDL: An extensible XML-based language for packet header description[J]. Computer Networks. 2006,50(5):688-706.
- [27] Petr Kobiersky,Jan Korenek,Libor Polcak.Packet Header Analysis and Field Extraction for Multigigabit Networks[C]. Design and Diagnostics of Electronic Circuits & Systems, DDECS '09. 12th International Symposium on,2009,96 - 101.
- [28] 卢良进,徐向华,童超.无线传感网络协议分析技术研究是实现[J]. 传感技术学报,2009,(22): 1828-1833.
- [29] Michael Attig,Gordon Brebner.400 Gb/s Programmable Packet Parsing on a Single FPGA.Architectures for Networking and Communications Systems (ANCS)[C], 2011 Seventh ACM/IEEE Symposium on,2011:12 - 23.
- [30] Christos Kozanitis,John Huber,Sushil Singh, George Varghese.Leaping multiple headers in a single bound:wire-speed parsing using the Kangaroo system[J]. INFOCOM, 2010 Proceedings IEEE, 2010,(3): 1 - 9.
- [31] Ralph Duncan,Peder Jungc. packetC Language for High Performance Packet Processing. High Performance Computing and Communications[C], HPCC '09. 11th IEEE International Conference on, 2009.(7): 450 - 457.
- [32] 董永吉,郭云飞,黄万伟,夏军波.一种新的高速报文解析结构研究[J]. 电子与信息学报,2013, 35(5):1083-1089.
- [33] Viktor Puš, Lukáš Kekely, Jan Kořenek. Low-Latency Modular Packet Header Parser for FPGA,ANCS '12 Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems, 2012.page: 77 - 78.
- [34] 熊安萍.基于 Winsock 技术的数据包解析研究[J].计算机科学, 2006, 33(12): 81-82.

- [35] 谢小特,专有网络协议数据包分析软件的设计与实现[D],湖南:国防科学技术大学,2008.
- [36] 张友生等编著,软件体系结构(第2版),北京:清华大学出版社,2011.
- [37] 刘强,汪斌强,徐恪.基于构件的层次化可重构网络构建及重构方法.计算机学报, 2010,33(9):1557-1568.
- [38] 陈文龙,徐恪,徐明伟,杨扬.基于构件的可重构路由开发环境.信息工程大学学报, 2009,10(1):28-33.
- [39] 李鹏,兰巨龙.一种新型的可重构路由交换通用平台.计算机应用研究, 2009,26(6):2016-2019.
- [40] 陈文龙,可扩展路由器大规模路由管理研究[D],北京:北京科技大学,2011.
- [41] 袁博,基于可重构技术的网络节点节能问题关键技术研究[D],郑州:解放军信息工程大学,2012.
- [42] OSGi Alliance.OSGi Release 6. <https://www.osgi.org/developer/downloads/release-6/>.2014.
- [43] 王映辉,王立福.软件体系结构演化模型.电子学报, 2005, 33(8):1381-1386.
- [44] Steffen Kächele, Jörg Domaschka, Holger Schmidt, Franz J. Hauck.nOSGi:a posix-compliant native OSGi framework. Proceeding COMSWARE '11 Proceedings of the 5th International Conference on Communication System Software and Middleware, July 2011.
- [45] OpenDaylight. Online: <https://www.opendaylight.org/>.
- [46] ONOS. Online: <http://onosproject.org/>.
- [47] Floodlight. Online:<http://www.projectfloodlight.org/>.
- [48] Gude N, Koponen T, Pettit J, etc. NOX: towards an operating system for networks[J]. ACM SIGCOMM Computer Communication Review, 2008,38(3):105-110.
- [49] NetFPGA.Online: <http://netfpga.org/site/#/>.

## 致谢

时间过的很快，研究生三年时光转眼而过。回首三年往事，有过欢喜有过忧愁，有过兴奋有过失落，唯一不变的就是自己那颗不忘的初心，内大三年求学路充满了回忆，即使离开也会永记于心。在此，我要对我的导师李华教授致以最真挚的感谢，正是她一丝不苟的治学态度和勤奋刻苦的工作作风深深影响了我的日常生活和学习态度，我每一次成长和进步的背后都离不开她的谆谆教诲。李老师不仅悉心指导我的课题研究和学习，在生活上也给予了我无微不至的关怀和帮助。本文的成稿同样离不开李老师不厌其烦的修改和耐心指导。在这里我再一次向李华教授表达我最真挚的敬意，因为您不仅是我的研究生导师，更是我的人生路上的良师益友，真得很感谢您的培养与照顾！

再次，感谢给予我知识的所有老师和朋友们，尤其是阮宏伟老师和王显荣老师。阮老师在本文后期的发展思路给予了非常宝贵的建议，王显荣老师在前期本文课题提出过程中给予了非常重要的指导。在这里，我要向这些幕后的英雄们表达我真挚的谢意。

然后，我要感谢所有和我一起生活和学习过的同学们。王长忠师兄，张玉荣师姐，虽然他们已经离开了学校，但是他们在生活和学习上给了我很多的帮助。同时也感谢我的同学：杨晓、郝世杰和王春井在生活和学习上带给我很多的快乐和帮助；还有我的师兄师姐师弟师妹们：李元平、郑冰、王璐、高宇、尹小敏、陈云清、刘建琪、刘麒、杨珍、刘亚、云晖等，感谢他们对我的帮助。

最后，感谢我的家人，如果没有他们默默的付出，我的生活将是另一番情景，正是他们的鼓励和支持让我一直坚持到今天。

同时，感谢评审和答辩中的每位老师，感谢你们的宝贵意见。



## 攻读硕士期间发表的学术论文

- [1] XinLei Hu, XianRong Wang, Lu Wang, Hua Li.Design of Extensible Forwarding Element Architecture and Its Key Technology Verification, Proceedings of 2016 IEEE International Conference on Integrated Circuits and Microsystems, November,2016.page:214-218.
- [2] 刘麒, 徐阳, 吕婷, 胡新磊, 李华.基于 HTML5 WebWorker 组件的 DDoS 攻击方式和检测.计算机应用与软件, 2016,33(12):295-300.

## 主持与参加项目

- [1] 2016.05-2017.05 主持内蒙古自治区研究生科研创新资助项目《基于 SDN 的可扩展交换机架构设计及关键技术验证》
- [2] 2015.01-2017.12 内蒙古自然基金项目《基于 SDN 的数据中心多控制器测试研究》
- [3] 2012.01-2015.12 国家自然科学基金项目《面向属性的 CPN 建模及 On the Fly 辅助的测试生成方法研究》
- [4] 2015.01-2017.12 内蒙古高校科学技术研究项目(NJZY010)