

本科生毕业设计

|  |
| --- |
| 基于QT的网络数据包分析软件 |

|  |  |
| --- | --- |
| 院 系 | **计算机科学与技术** |
| 专业班级 | **计科201408班** |
| 姓 名 | **陈延良** |
| 学 号 | **U201414782** |
| 指导教师 | **周正勇** |
| 2018年06月10日 | |

**学位论文原创性声明**

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名： 年 月 日

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密囗，在 年解密后适用本授权书

2、不保密囗 。

（请在以上相应方框内打“√”）

作者签名： 年 月 日

导师签名： 年 月 日

摘 要

本次毕业设计为基于Qt的网络数据包分析软件，设计一款能够读取pcap文件，将其中的网络数据包内容显示出来，并进行统计分析，以图表的形式展示给用户。

本次设计采用libpcap+Qt的结构进行，通过wireshark抓包软件来抓去本地的一些网络数据包，然后保存为pcap文件格式。

×××××××××××××××××××××××××××××××××（目的，方法，结果，结论 四要素）。目的，基础研究设计，使用的方法和技巧，主要研究发现，总结你的解释，结论和含义。在什么背景下，我用什么方法研究什么问题，最终得到什么结果，这个研究有什么意义。

**关键词**：pcap文件；网络数据包分析；Qt界面开发；libpcap函数库

Abstract

Failure resilience is of pivotal importance in practical network function virtualization (NFV) systems, but has been mostly absent in the existing ones. The absence is mainly  
 due to the challenge of patching source code of the existing NF software for extracting important NF states, a necessary step toward flow migration and replication to provide failure tolerance. This paper proposes NFActor, a novel NFV system that uses the actor programming model to provide transparent resilience, high scalability and low overhead in network flow processing. In NFActor, a set of efficient APIs are provided for constructing NFs, with inherent support for scalability and resilience. A per-flow management principle is advocated, different from the existing practice, which provides dedicated micro service chain services for individual flows, enabling decentralized flow migration and scalable flow replication. We implement NFActor and show that it achieves good scalability, prompt flow migration and failure recovery with large numbers of concurrent flows. We also show that NFActor can enable applications such as live NF update and correct MPTCP subflow processing, which cannot be efficiently achieved in previous systems.

**Keywords:** Network Function Virtualization (NFV), Failure resilience, Actor model, Flow migration, Service Chain

目 录

[摘 要 II](#_Toc485274437)

[Abstract II](#_Toc485274438)

[1 绪 论 2](#_Toc485274439)

[1.1 课题研究的背景和意义 2](#_Toc485274440)

[1.2 国内外研究现状 2](#_Toc485274441)

[1.3 课题要求 2](#_Toc485274442)

[1.4 研究方案 2](#_Toc485274443)

[1.5 课题来源 2](#_Toc485274444)

[1.6 论文结构 2](#_Toc485274445)

[2 NFActor框架介绍 2](#_Toc485274446)

[2.1 总体框架 2](#_Toc485274447)

[2.2 运行时系统 2](#_Toc485274448)

[2.3 网络功能接口 2](#_Toc485274449)

[2.4 虚拟交换机 2](#_Toc485274450)

[2.5 协调器 2](#_Toc485274451)

[2.6 本章小结 2](#_Toc485274452)

[3 系统设计与实现 2](#_Toc485274453)

[3.1 错误恢复 2](#_Toc485274454)

[3.2 流量迁移 2](#_Toc485274455)

[3.3 动态扩展 2](#_Toc485274456)

[3.4 实现细节与特点 2](#_Toc485274457)

[3.5 本章小结 2](#_Toc485274458)

[4 性能测试与分析 2](#_Toc485274459)

[4.1 系统常规运行演示 2](#_Toc485274460)

[4.2 分组处理的吞吐量测试 2](#_Toc485274461)

[4.3 流量迁移的耗时测试 2](#_Toc485274462)

[4.4 动态扩展吞吐量的测试 2](#_Toc485274463)

[4.5 流备份的性能测试 2](#_Toc485274464)

[4.6 其他应用的测试 2](#_Toc485274465)

[4.7 本章小结 2](#_Toc485274466)

[5 总结与展望 2](#_Toc485274467)

[致 谢 2](#_Toc485274468)

[参考文献 2](#_Toc485274469)

[附录 大学期间发表或提交的论文 2](#_Toc485274470)

# 绪 论

## 课题研究的背景和意义

随着计算机网络的不断发展，全球信息化已经成为当今社会发展的趋势。但由于计算机网络自身所特具有的特点，比如联结形式多样性和网络开放性、互连性等特征，所以导致网络易受黑客还有一些病毒的攻击。所以网络信息的安全和保密是一个至关重要的问题。对于军用的自动化指挥网络和银行等传输敏感数据的计算机网络而言，其网上信息的安全和保密尤为重要。因此，网络必须有足够强的安全措施，否则该网络将是个无用、甚至危及国家安全的网络。在计算机网络的世界里，存在着很多潜在的威胁，因此网络的安全措施应能全方位地应对各种不同的威胁，

计算机网络所面临的威胁大体可分为两种：一是对网络中信息的威胁；二是对网络中设备的威胁。影响计算机网络的因素很多，有些因素可能是有意的，又有可能是无意的；可能是人为的，业可能是非人为的；可能是外来黑客对网络系统的资源非法占有，归结起来，针对网络安全的威胁主要有三种：

1. 人为的无意失误：如操作员安全配置不当造成的安全漏洞，用户安全意识不强，用户口令选择不慎，用户将自己的账号随意转借他人或与别人共享等都会对网络安全带来威胁。
2. 人为的恶意攻击：这是计算机网络所面临的最大威胁，敌手的攻击和计算机犯罪就属于这一类。此类攻击又可分为以下两种：一种是主动攻击，它以各种方式有选择地破坏信息的有效性和完整性；另一类是被动攻击，它是在不影响网络正常工作的情况下，进行截获、窃取、破译以获得重要机密信息。这两种攻击均可对计算机网络造成极大的危害，并导致机密数据的泄漏。
3. 网络软件的漏洞和“后门”：网络软件不可能是百分之百的无缺陷和无漏洞的，然而这些漏洞和缺陷恰恰是黑客进行攻击的首选目标，曾经出现过黑客攻入网络内部的事件，这些事件的大部分就是因为安全措施不完善所招致的苦果。另外软件的“后门”都是软件公司的设计编程人员为了自便而设置的，一般不为外人所知，但一旦“后门”洞开，其造成的后果不堪设想。

为了及早发现并制止网络上的各种攻击，我们需要通过对网路上的数据进行分析来发现并找出问题，提前预防。这也是本此课题的一个重要目的。网络安全管理员运用网络封包截获技术，抓取网络中有用的数据包，然后通过对数据包内容进行分析，确定哪些是有害的或者含有攻击企图的包，以此来达到对网络攻击的预防。同时许多防火墙也是基于包过滤技术的，本文将介绍网络数据包协议分析程序的工作原理以及它的实现。

网络数据包协议分析程序是一种用于收集网络中有用数据程序，这些数据可以是用户的账号密码，也可以是一些商用机密数据等。它是利用计算机的网络接口截获目的地为其他计算机的数据报文的一种工具。

网络数据包协议分析程序的正当用处主要是分析网络的流量，以便找出所关心的网络中潜在的问题。例如，假设网络的某一段运行得不是很好，报文的发送比较慢，而我们又不知道问题出在什么地方，此时就可以用网络数据包协议分析程序来作出精确的问题判断。在合理的网络中，网络数据包协议分析程序的存在对系统管理员是至关重要的，系统管理员通过网络数据包协议分析程序可以诊断出大量不可见模糊问题，这些问题涉及两台乃至多台计算机之间的异常通讯、有些甚至牵涉到各种协议，借助网络数据包协议分析程序系统管理员可以方便确定出多少的通讯量属于哪个网络协议、占主要通讯协议的主机是哪一台、大多数通讯的目的地是哪台主机、报文发送占用多少时间、或者相互主机的报文传送间隔时间等待，这些信息为管理员判断网络问题、管理网络区域提供了非常宝贵的信息。

## 国内外研究现状

### 网络功能虚拟化（Network Function Virtualization）

现在国内外已经有相当多的成熟且功能强大的网络数据包协议分析软件。比较著名的网络数据包分析软件有：wireshark、TcpDump、EtherPeek等。

Wireshark：Wireshark是一个开放源码的网络分析系统，也是目前最好的使用较多的开放源码的网络协议分析器，支持linux和windows平台。Wireshark起初由Gerald Combs开发，随后由一个松散的Wireshark团队组织进行维护开发。它目前所提供的强大的协议分析功能完全可以媲美商业的网络分析系统。自从1998年发布最早的0.2版本至今，大量的志愿者为Wireshark添加新的协议解析器，如今Wireshark已经支持五百多种协议解析。很难想象如此多的人开发的代码可以很好的融入系统中；并且在系统中加入一个新的协议解析器很简单，一个不了解系统的结构的新手也可以根据留出的接口进行自己的协议开发。这些都归功于Wireshark良好的设计结构。事实上由于网络上各种协议种类繁多，各种新的协议层出不穷，一个好的协议分析器必需有很好的可扩展性和结构。这样才能适应网络发展的需要不断加入新的协议解析器。

TcpDump：顾名思义，TcpDump可以将网络中传送的数据包的“头”完全截获下来提供分析，它支持针对网络层、协议、主机、网络或端口的过滤，并提供and、or、not等逻辑语句来帮助你去掉无用的信息。用尽量简单的话来定义TcpDump就是：dump the trafiic on a network，根据使用者的定义对网络上数据包进行截获的包分析工具，TcpDump以其强大的功能，灵活的截取策略，成为每个高级的系统管理员分析网络，排查问题等所必备的工具之一。

EtherPeek：这个工具软件开始只是一个网络分析器型的数据包监测软件，经过这些年的发展已经成为一个真正的网络管理工具并具有网络监视和分析等新的功能，被美国联邦调查局用来追踪逃犯、贩卖毒品的人、电脑黑客和一些被怀疑为外国间谍的人。是一个直观，功能强大的以太网网络和协议分析器。支持Macintosh和Windows平台。EtherPeek把查找和修复多平台上的复杂网络任务变得简单化。EtherPeek采用工业标准，非常容易使用，提供解码、过滤和诊断网络的功能。以友好图形界面出名，EtherPeek提供非常详细且多样化的网络使用信息，网络结点的会话和数据包内容。在有问题的局域网络中使用EtherPeek执行一个自定的诊断测试，监控网络的通信和事件，跟踪非法的网络活动，测试和调试网络软硬件。

### Actor模型简介

Actor编程模型已被用于构建许多大规模分布式系统[1][18]。每个actor都是一个独立的执行单元，可以看作是一个逻辑线程。在最简单的形式中，actor包含内部actor状态（例如，统计计数器，传出请求数），用于接收传入消息的邮箱和几个消息处理函数。Actor使用消息处理函数传入的消息，通过内置消息传递通道与其他Actor交换消息，并创建新的actor。多个actor异步运行，就像在自己的线程中运行一样，这简化了对分布式协议的编程，并消除了可能导致系统崩溃的潜在竞争条件。Actor通常运行在强大的运行时系统上，该系统调度actor执行并实现网络透明，即actor与在不同运行时系统上运行的远actor进行通信的时候，就好像它们都运行在相同的运行时系统上一样。

Actor模型可以非常自然的适用于构建分布式NFV系统。我们可以创建一个actor作为一个流处理单元，并且构建流分组处理和控制消息（例如，用于流迁移和复制）作为actor上的消息处理函数。当有了能够启动其他actor的能力之后， actor本身几乎可以以分布式方式实现流量迁移和备份。据我们所知，NFActor是第一个使用actor模型来构建有故障恢复功能并且能高效处理分组的NFV系统。

目前已经存在几个流行的actor框架已被用于构建广泛的分布式应用，例如Scala Akka，Erlang，Orleans和C ++ Actor Framework。这些框架都还没有被用来优化过NFV系统。在我们的初始原型实现中，我们在C ++ Actor框架之上构建了NFActor，但是消息传递的效率经过实验发现并不尽如人意。原因主要在于在在CAF框架中传递actor消息使用的是内核网络栈，对于NFV系统具有不可容忍的数据交换的开销[22]。于是我们为NFActor定制一个高性能的actor框架。

## 课题要求

使用actor编程模型设计一个拥有故障恢复、高扩展性、低消耗特性的NFV分布式系统来处理网络流量。具体要求如下：

1. 自定义实现Actor模型的框架并使用自定义actor模型来完成运行时系统中流actor的构建。
2. 利用DPDK设计并实现运行时系统之间的可靠通信。
3. 使用Protobuf自定义协调器与运行时系统通信的消息类型，并使用GRPC实现协调器与运行时系统之间的通信。
4. 设计NF的统一接口，使用户可以利用接口来设计新的网络功能。
5. 设计流量迁移和备份的策略，使得系统可以支持高效的错误恢复功能和动态扩展功能。

## 研究方案

在本次课题的研究中，网络数据包分析软件实现对pcap文件格式类型的网络数据包解析和统计功能，完成一些常用协议的识别和分析：要求至少实现TCP/IP协议簇几个基本协议的分析——ARP、TCP、UDP、ICMP以及一些应用层的常用协议如DHCP、DNS、HTTP等。

## 课题来源

课题来源于香港大学工程学院计算机系系统与网络实验组的网络功能虚拟化项目。

## 论文结构

本文共分为五个章节：

第一章，首先介绍了一下NFV系统的研究背景，研究现状，介绍了Actor模型，接着引出了我们需要设计的NFV系统：NFActor. 并且从actor的机制上说明了实现系统的可行性，与之前研究的区别以及可能带来的性能上的提升。

第二章，根据课题需要，我们首先介绍了一下系统的整体架构，然后分别详细介绍了系统的各个重要组成部分，包括其工作原理，设计思路，和实现细节。

第三章，在整个系统的构建介绍完成之后，开始介绍一些系统操作的设计与实现，比如流量迁移的设计，备份操作的设计，还有控制器与运行时系统之间的通信、不同运行时系统之间的通信等操作的具体设计与实现将在本章节进行介绍。在这一章节，我们还将我们的一些实现亮点分别进行展示，这些亮点主要是设计上的一些策略或者是与之前的研究不同的一些设计细节等。

第四章，我们将会进行实验测试结果的展示，主要包括系统总吞吐量、系统进行流量迁移、备份所消耗的时间和性能的测试，以及NFActor在一些之前研究无法适用的应用场景下进行高效分组处理的实验结果展示。

第五章，我们将对整个项目进行总结，总结其已经实现的所有功能和达到的性能以及展望项目未来的发展前景和研究方向。

# 网络数据包介绍

在这一章本文将介绍各种不同网络数据包的结构。

## pcap文件类型

运行时系统是服务链的执行环境。运行时系统可以在容器中运行，以便在动态扩展和故障恢复的情况下快速启动。在同一物理服务器上可以运行多个运行时系统。在NFActor中，虚拟交换机也是和运行时系统一样的架构，只是没有服务链，运行时系统和虚拟交换机通过L2网络连接。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | | 2 | 3 | 4 | 5 | 6 | | | 7 | | 8 | 9 | A | | B | C | D | | E | F |
| **Pcap文件头（24字节）** | | | | | | | | | | | | | | | | | | | | | |
| **Pcap文件头** | | | | | | | | | | | **pcap数据包头（16字节）** | | | | | | | | | | |
| **Pcap数据包头** | | | | | | | | | | | **MAC以太网帧头（14字节）** | | | | | | | | | | |
| **MAC头** | | | | | | | | | **IP数据报头（一般20字节）** | | | | | | | | | | | | |
| **IP头** | | | | | | | | | | | | | | | **TCP头** | | | | | | |
| **TCP报文段头（一般20字节）** | | | | | | | | | | | | | | | | | | | **数据域** | | |
| **数据域** | | | | | | | | | | | | | | | | | | | | | |
| **Pcap数据包头（16字节** | | | | | | | | | | | | | | | | | | | | | |
| **MAC以太网帧头（14字节）** | | | | | | | | | | | | | | | | | | | **IP头** | | |
| **IP数据报头（一般20字节）** | | | | | | | | | | | | | | | | | | | | | |
| **IP头** | | **TCP报文段头（一般20字节）** | | | | | | | | | | | | | | | | | | | |
| **TCP头** | | | | | | | | **数据域** | | | | | | | | | | | | | |
| **数据域** | | | | | | | | | | | | | | | | | | | | | |
| **……** | | | | | | | | | | | | | | | | | | | | | |

每个虚拟交换机都配置了一个入口IP地址。协调器设置流规则以将数据流引导到虚拟交换机，进一步将它们分派到管理着服务链的运行时系统中。运行时系统创建一个用于处理每个流的专用流actor。处理完成后，流出的数据包被发送到另一个虚拟交换机，该交换机将流转发到其最终目的地[[1]](#footnote-1)。

NFActor的设计主要是为了达到以下几个目标：

**透明的错误恢复**。需要用来实现错误恢复的流量迁移和备份等操作会在服务链进行流处理的过程中以高度分布式的方式透明地运行。

**高可扩展性。**对于运行时系统和虚拟交换机，NFActor实现了良好的水平可扩展性，使得NFActor容易地使用与不同的工作负载。

**TCP／IP报文封装**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MAC帧头** | **IP头** | **TCP头** | **上层数据** | **FCS** |

**2.2.1 MAC数据帧头和帧尾**

MAC帧头共14字节，如下：

1. 目的MAC地址（48 bit）；
2. 源MAC地址（48 bit）；
3. 上层协议类型（16 bit）；
   1. 如0x0800代表上一层是IP协议，0x0806为ARP

MAC帧尾为4字节，如下：

1. 数据帧尾校验和

**高分组处理吞吐量**。在NFActor系统中，我们将流迁移、备份和动态扩展等功能的消耗降至最低以保证了系统处理分组时的高吞吐量。

网络数据包TCP，UDP，ARP，ICMP，HTTP

\*ICMP

ICMP（Internet Control Message Protocol）即网络控制信息协议，协议是IP协议的补充，用于IP层的差错报告，拥塞控制，路径控制以及路由器或主机信息的获取。ICMP与IP协议位于同一层（IP层），但ICMP报文是封装在IP数据包的数据部分进行传输的。ICMP分很多类，总的来看分为三类1）差错报告 2）控制报文 3）请求应答报文。

结构：8bit类型码 8bit代码 16bit校验和 32bit首部其他部分 然后是数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 以太网目的地址 | | | | 以太网源地址 | | | | | 帧类型 | 版本字 |
| 节TOS | 标识 | | 偏移 | TTL | 协 | 校验和 | 源IP地址 | | | 目的 |
| IP地址 | 类 | 码 | 校验和 | 标识符和序号 | | | | 数据选项 | | |
|  | | | | | | | | | | |

\*ARP

ARP（Address Resolution Protocol）即地址解析协议，是根据IP地址获取物理地址的一个TCP／IP协议。在网络中ARP请求报文通常是以广播的形式发送到局域网中，请求一个IP地址对应的物理地址，其数据包格式如下

结构：以太网首部14字节，6字节目的MAC地址，6字节源MAC地址，2字节帧类型，请求应答部分：2字节硬件类型，2字节协议类型，1字节硬件地址长度，1字节协议地址长度，2字节OP（为1表示ARP请求，为2表示ARP应答），6字节发送端以太网地址，4字节发送端IP地址，6目的以太网地址，4字节目的IP地址。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 以太网目的地址 | | | | 以太网源地址 | | 帧类型 | 硬件类 |
| 协议类 | 硬 | 协 | OP | 发送端以太网地址 | | 发送端IP地址 | |
| 目的以太网地址 | | | | 目的IP地址 |  | | |

前14字节为以太网首部，对于以太网首部来说，如果是请求报文的话，以太网目的地址为FF:FF:FF:FF:FF:FF;即广播报，帧类型的话ARP对应的值为0x0806

硬件类型表明ARP协议实现在那种情况类型网络上，值为1即表示以太网地址，协议类型表示解析协议（上层协议），一般为0800，即IP，硬件地址长度，也就是MAC地址长度，即6字节，协议地址长度，也就是IP地址长度，即4字节，操作类型（OP），表示ARP协议数据报类型，1表示ARP请求包，2表示ARP应答包，3表示RARP请求包，4表示RAPR应答包，然后发送以太网地址就是源MAC地址，发送端IP地址也就是源IP地址，目的以太网地址就是目的MAC地址，如果是请求报文的话为00:00:00:00:00:00，目的端IP地址就是目的端IP地址。

以太网帧类型：（2字节）

取值Ox0800代表IPv4协议帧

取值0x0806代表ARP协议帧

取值0x8864代表PPPoE协议帧

取值0x8100代表802.1Q tag协议帧

取值0x86DD代表IPv6协议帧

取值0x8847代表MPLS Label协议帧

取值0x0835代表RARP协议帧

取值0x8137代表IPX和SPX传输协议帧

IP头协议类型：（1字节）

取值1代表ICMP

取值6代表TCP

取值17代表UDP

## 网络数据包

### IP数据报

IP协议是TCP／IP协议族中最为核心的协议。它提供不可靠，无连接的服务，也即依赖其他层的协议进行差错控制。在局域网环境，IP协议往往被封装在以太网帧中传送。而所有的TCP，UDP，ICMP，IGMP数据都被封装在IP数据报中传送。IP数据报头一般为20字节，如下：

1. 版本信息（4 bit）；

用来表明IP协议实现的版本号，当前一般为IPv4，即0100

1. 头长度（4 bit）；

是头部占32比特的数字，包括可选项。普通IP数据报，没有任何选项的，该字段的值为5，即160比特=20字节。此字段最大值为60字节。

1. 服务类型（8 bit）；

其中前3比特为优先权字段（Precedence，现已被忽略）。第8比特保留未用。第4至7比特分别代表延迟，吞吐量，可靠性和花费。他们取值为1时分别代表要求最小延时，最大吞吐量，最高可靠性和最小费用。这4比特的服务类型中只能置其中1比特为1。可以全为0。服务类型字段声明了数据报被网络系统传输时可以被怎样处理。例如：TELNET协议可能要求最小的延迟，FTP协议可能要求有最大吞吐量，SNMP协议可能要求有最高可靠性，NNTP可能要求最小费用，而ICMP协议可能无特殊要求。大部分主机会忽略这个字段，但一些动态路由如OSPF，IS-IS可以根据这些字段进行理由决策。

1. 数据包长度（16 bit）；

指明整个数据报的长度（以字节为单位）。最大长度为65535字节。

1. 数据包标识（16 bit）；

用来唯一标识主机发送的每一份数据报。通常每发一份报文，他的值会加1。

1. 分片使用，标志（3 bit）+片偏移（13 bit）；

前3比特标志一份数据是否要求分段。后13比特指明该段偏移距原始数据报开始的位置。

1. 存活时间（8 bit）；

用来设置数据报最多可以经过的路由器数。由发送数据的源主机设置，通常为32，64，128等，每经过一个路由器，其值减1，直到0时该数据报会被丢弃。

1. 协议类型（8 bit）；

指明IP层所封装的上层协议类型，如ICMP（1），IGMP（2），TCP（6），UDP（17）等。

1. 首部校验和（16 bit）；

内容是根据IP头计算得到的校验和码。计算方法是，对头部中两个16比特进行二进制反码求和。（和ICMP，IGMP，TCP，UDP不同，IP不对头部后的数据进行校验）。

1. 源IP地址（32 bit）；
2. 目的IP地址（32 bit）；
3. 选项（如果有的话，32 bit）；

用来定义一些任选项：如记录路径，时间戳等，这些选项很少被使用，同时并不是所有主机和路由器都支持这些选项。可选字段的长度必须是32比特的整数倍。如果不足必须填充0以达到此长度要求。

。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | |  | |
| IP报头（20字节） | | | | | | | | | | | | | | | | |
|  | | | IP数据报 | | | | | | | | | | | | | |
| …… | | | | | | | | | | | | | | | | |

### ARP报文

流actor可以直接方便地迁移这些状态。在无备份的情况下，这个系统在处理完数据包之后，就会将数据包发送出系统，如果是有备份的情况下，这些数据包会连同NF状态一起被送到备份的运行时系统中去。接下来系统会检测是否有停机指令，如果

ARP格式：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | | 4 | | 5 | 6 | 7 | 8 | 9 | a | | b | c | | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | | | | | | HType | |
| PType | | | Hl | | Pl | | Op | | 发送端MAC地址 | | | | | | | | 发送端IP地址 | | | |
| 目的MAC地址 | | | | | | | | | 目的IP地址 | | | | |

1.硬件类型

硬件地址不只以太网一种，如果是以太网类型，此时值为1

2.协议类型

表示要映射的协议地址的类型，要对IPv4地址进行映射，此时值为0x0800

3.硬件地址长度

以太网类型地址长度对应6字节

4.协议地址长度

IPv4地址类型对应长度4字节

5.操作类型

操作类型字段，值为1，表示进行ARP请求；值为2，表示进行ARP应答；值为3，表示进行RARP请求；值为4，表示进行RARP应答。

6.发送端以太网地址

发送ARP请求或应答的硬件地址，这里是以太网地址，和MAC头的源地址相同。

7.发送端IP地址

发送ARP请求或应答的IP地址

8.目的以太网地址

目的以太网地址，请求时通常为广播，即为00:00:00:00:00:00。

9.目的IP地址

目的IP地址，请求中为请求MAC的IP地址，应答时为请求的源IP地址

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | |  | |
| ARP报文头（20字节） | | | | | | | | | | | | | | | | |
|  | | | ARP报文 | | | | | | | | | | | | | |
| …… | | | | | | | | | | | | | | | | |

### TCP报文段

流actor可以直接方便地迁移这些状态。在无备份的情况下，这个系统在处理完数据包之后，就会将数据包发送出系统，如果是有备份的情况下，这些数据包会连同NF状态一起被送到备份的运行时系统中去。接下来系统会检测是否有停机指令，如果TCP报文段头一般为20字节，如下：

1. 源端口号（16 bit）；
2. 目的端口号（16 bit）；
3. 序列号（32 bit）；
4. 确认号（32 bit）；
5. TCP头长度（4 bit）；
6. 保留部分（6 bit）；
7. 标志位（6 bit）；
8. 窗口大小（16 bit）；
9. 校验和（16 bit）；
10. 紧急指针（16 bit）；
11. 选项（如果有）；

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | |  | |
| IP数据报头（20字节） | | | | | | | | | | | | | | | | |
|  | | | TCP报文段头 | | | | | | | | | | | | | |
| …… | | | | | | | | | | | | | | | | |

### UDP报文段

流actor可以直接方便地迁移这些状态。在无备份的情况下，这个系统在处理完数据包之后，就会将数据包发送出系统，如果是有备份的情况下，这些数据包会连同NF状态一起被送到备份的运行时系统中去。接下来系统会检测是否有停机指令，如果UDP报文段头为8字节，如下：

1. 源端口号（16 bit）；
2. 目的端口号（16 bit）；
3. 数据包长度（16 bit）；

4）校验和（16 bit）；

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | |  | |
| IP数据报头（20字节） | | | | | | | | | | | | | | | | |
|  | | | UDP报文段头 | | | | | | | | | | | | | |
| …… | | | | | | | | | | | | | | | | |

### ICMP报文

ICMP（Internet Control Message Protocol）即网络控制信息协议，协议是IP协议的补充，用于IP层的差错报告，拥塞控制，路径控制以及路由器或主机信息的获取。ICMP与IP协议位于同一层（IP层），而ICMP报文是封装在IP数据包的数据部分进行传输的，不同于TCP和UDP，他并不传输用户数据，但对于用户数据的传递起者至关重要的作用。如下位ICMP的报文结构

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | | |  | |
| IP数据报头（20字节） | | | | | | | | | | | | | | | | | |
|  | | | ICMP固定报文头（8字节） | | | | | | | | | ICMP数据部分 | | | | | |
| …… | | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | |  | |
| IP报头（20字节） | | | | | | | | | | | | | | | | |
|  | | | ICMP报文 | | | | | | | | | | | | | |
| …… | | | | | | | | | | | | | | | | |

ICMP分很多类，总的来看分为两类1）通知错误原因的差错报文 2）请求应答的查询报文

ICMP报文格式及类型

1.通用格式

|  |  |  |
| --- | --- | --- |
| 类型 | 代码 | 校验和 |
| （这4字节取决于ICMP报文类型） | | |
| ICMP数据部分 | | |

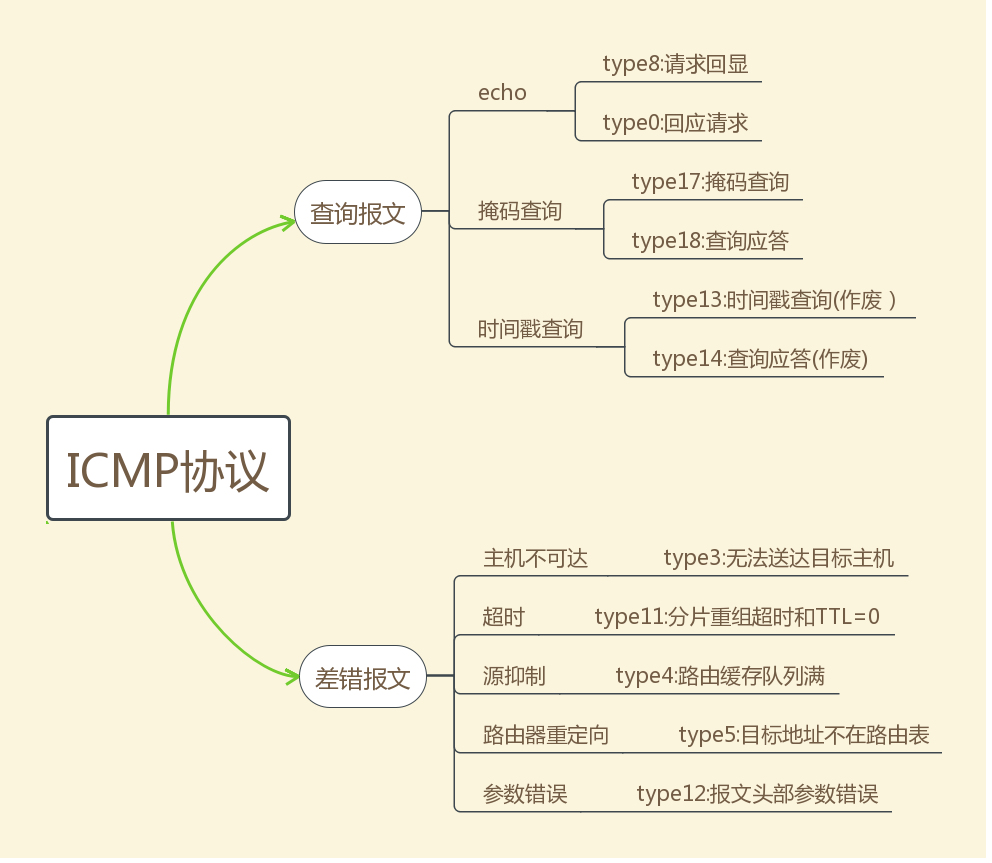
前8字节固定

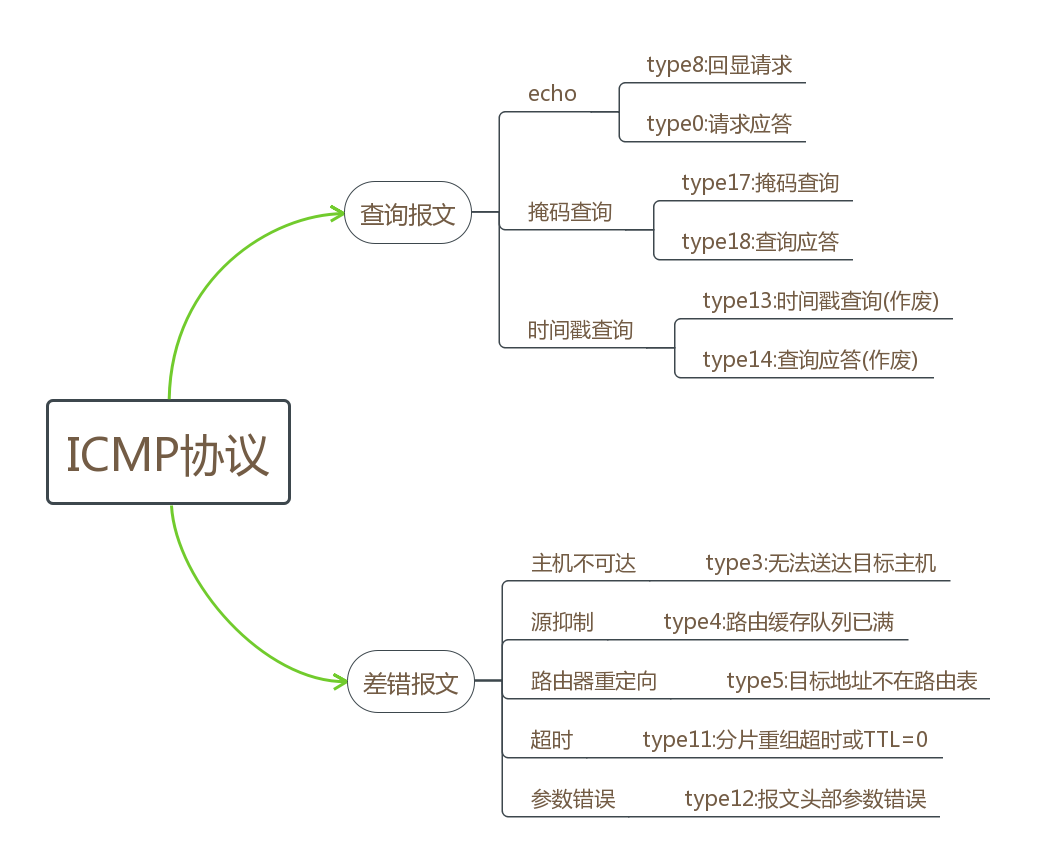
对于类型目的不可达（3），超时（11），参数错误报文（12），源抑制（4），重定向（5）：

|  |  |  |
| --- | --- | --- |
| 类型 | 代码 | 校验和 |
| 未用（全为0） | | |
| IP数据报头（包括选项）+IP数据报原始数据字段前8字节 | | |

对于类型Echo(8、10)、信息请求或信息响应，：

|  |  |  |
| --- | --- | --- |
| 类型 | 代码 | 校验和 |
| 标识符 | | 序列号 |
| ICMP数据部分 | | |





常见类型：

回显请求（0）：ping工具通过发送ICMP回显消息检查特定节点的IPv4连接已排查网络问题。

回显应答（8）：节点发送回显答复消息响应ICMP回显消息。

重定向（5）：路由器发送“重定向”消息，告诉发送主机到目标IPv4地址更好的路由。

源抑制（4）：路由器发送“源结束”消息，告诉发送主机以较低的频率发送数据报，以免因为路由器拥塞而导致数据报被丢弃。

超时（11）：当超过IP生存期时和分段IP数据报没有在期限内重新组合都会发送这种消息。

无法达到目标（3）：路由器和目标主机通知发送主机数据无法传送。

类型（8bit） 代码0～15（8bit） 校验和（16bit）

暂定不同类型有不同内容（32bit）

IP首部（包括选项）+原始IP数据报中数据的前8字节

类型大致分为差错报文，查询报文。

查询： 0：回送应答

8：回送请求

9：路由器公告

10：路由器请求

17：地址掩码

18：地址掩码应答

差错： 3：目标不可达

4：源抑制

5：重定向

11：超时

差错报文：

当发送一份差错报文时，报文始终包含IP的首部和产生ICMP差错报文的IP数据报的前8字节，这样接受iCMP差错报文的模块就会吧它与某个特定协议和用户进程联系起来（分别是IP头中协议字段和IP数据报前8字节中的端口号来判断）

### DNS报文

流actor可以直接方便地迁移这些状态。在无备份的情况下，这个系统在处理完数据包之后，就会将数据包发送出系统，如果是有备份的情况下，这些数据包会连同NF状态一起被送到备份的运行时系统中去。接下来系统会检测是否有停机指令，如果

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | | |  | |
| IP数据报头（20字节） | | | | | | | | | | | | | | | | | |
|  | | | UDP报文段头（8字节） | | | | | | | | | DNS报文 | | | | | |
| …… | | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | | |  | |
| IP数据报报头（20字节） | | | | | | | | | | | | | | | | | |
|  | | | UDP报文段头（8字节） | | | | | | | | | DNS报文 | | | | | |
| …… | | | | | | | | | | | | | | | | | |

××DNS报文格式及类型

|  |  |
| --- | --- |
| Transaction ID（会话标识） | Flags(标识) |
| Questions(问题数) | Answer RRs(回答 资源记录数) |
| Authority RRs(授权 资源记录数) | Additional RRs(附加资源记录数) |
| Name(查询名，长度不固定) | |
| Type(查询类型) | Class（查询类） |
| Name(查询名，长度不固定) | |
| Type(查询类型) | Class（查询类） |
| Time to live(生存时间) | |
| Data lenth(资源数据长度) | Data |
| Data（资源数据，长度不固定） | |
| Name(查询名，长度不固定) | |
| Type(查询类型) | Class（查询类） |
| Time to live(生存时间) | |
| Data lenth(资源数据长度) | Data |
| Data（资源数据，长度不固定） | |
| Name(查询名，长度不固定) | |
| Type(查询类型) | Class（查询类） |
| Time to live(生存时间) | |
| Data lenth(资源数据长度) | Data |
| Data（资源数据，长度不固定） | |

1．头部(蓝色区域)

1.1回话标识（2字节）：DNS报文的ID标识，对于请求报文和其对应的应答报文，这个字段是相同的，通过它可以区分DNS应答报文是哪个请求的响应。

1.2标志（2字节）：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| QR | Opcode | AA | TC | RD | RA | (Zero) | rcode |

QR：查询/响应标志，0为查询，1为响应

Opcode:0表示标准查询，1表示反向查询，2表示服务器状态请求。

AA:表示授权回答

TC：表示可截断的

RD：表示期望递归

RA：表示可用递归

Rcode:表示返回码，0表示没有差错，3表示名字差错，2表示服务器错误。

1.3数量字段

Questions、Answer RRs、Authority RRs、Additional RRs各自表示后面的四个区域的数目。Questions表示查询问题区域节的数量，Answer RRs表示回答区域的数量，Authority RRs表示授权区域的数量，Additional RRs表示附加区域的数量。

2．查询问题区域（绿色区域）

2.1查询名：长度不固定，且不使用填充字节，一般该字段表示的就是需要查询的域名，一般格式如下：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | h | u | b | 4 | h | u | s | t | 3 | e | d | u | 2 | c | n | 0 |

数字表明后面的字节数，hub前为数字3,hust前为4,edu前为3，cn前为2，最后结尾必须是0,所以域名为hub.hust.edu.cn。

2.2查询类型：

表3-1 DNS查询类型

|  |  |  |
| --- | --- | --- |
| 类型 | 助记符 | 说明 |
| 1 | A | 由域名获得IPv4地址 |
| 2 | NS | 查询域名服务器 |
| 5 | CNAME | 查询规范名称 |
| 6 | SOA | 开始授权 |
| 11 | WKS | 熟知服务 |
| 12 | PTR | 把IP地址转换为域名 |
| 13 | HINFO | 主机信息 |
| 15 | MX | 邮件交换 |
| 28 | AAAA | 由域名获得IPv6地址 |
| 252 | AXFR | 传送整个区的请求 |
| 255 | ANY | 对所有记录的请求 |

2.3查询类

通常为1，表明是internet数据

3．回答区域（紫色区域）

3.1域名

格式与Queries区域查询名字字段一样，有一点不同的是，当域名重复出现时，该字段使用2字节的便宜指针表示，具体格式时，前面两个最高位为11.用于识别指针，其余14位表示从DNS报文开始出计数，指出字段在报文中相应偏移字节数。典型的一个列子就是c00c(11000000 00001100)12正好时头部长度，正好指向Queries区域的查询名字字段。

3.2查询类型

同上，见表格

3.3.查询类

对于internet信息，总是IN

3.4生存时间

以秒为单位，表示的是资源记录的生命周期，一般用于当地地址解析程序取出资源记录后决定保存及使用缓存数据的时间，它同时也可表示该资源的稳定程度，极为稳定的信息会被分配一个很大的指，如84600，这是一天的秒数。

3.5资源数据

该字段也是一个变长字段，表示按照查询段的要求返回的相关资源记录，例如如果查询报文想要的回应是一个IP地址，这里就对应一个IP地址。

4．授权区域（黄色区域）

同上回答区域

5．附加区域（橙色区域）

同上回答区域

### DHCP报文

流actor可以直接方便地迁移这些状态。在无备份的情况下，这个系统在处理完数据包之后，就会将数据包发送出系统，如果是有备份的情况下，这些数据包会连同NF状态一起被送到备份的运行时系统中去。接下来系统会检测是否有停机指令，如果

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | | |  | |
| IP数据报头（20字节） | | | | | | | | | | | | | | | | | |
|  | | | UDP报文段头（8字节） | | | | | | | | | DHCP报文 | | | | | |
| …… | | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | |  | |
| IP报头（20字节） | | | | | | | | | | | | | | | | |
|  | | |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |

DHCP协议

|  |  |  |  |
| --- | --- | --- | --- |
| 报文类型（1B） | 硬件类型（1B） | 硬件地址长（1） | 跳数（1B） |
| 事务ID（4B） | | | |
| 秒数（2B） | | 标志位（2B） | |
| 客户端IP地址（4B） | | | |
| 你的IP地址（4B） | | | |
| 服务器IP地址（4B） | | | |
| 网关IP地址（4B） | | | |
| 客户机硬件地址（16B） | | | |
| 服务器主机名（64B） | | | |
| 启动文件名（128B） | | | |
| 选项（变长） | | | |

1.报文类型

报文类型码，只能取值1和2 ，请求包request为1，应答包reply为2

2.硬件类型

硬件类型代码，1代表千兆以太网

3.硬件地址长度

单位为字节，表明硬件地址长度，以太网对应值为6

4.跳数

DHCP跨网段使用时由reply agent使用，DHCP client发出的包此字段清零。

5.事务ID

值为DHCP client发送数据包时填充的一个随机数，用来在client和server之间表示一组request和reply。

6.秒数

由客户端填充，表示客户端开始请求分配IP或请求更新IP租期所经历的时间。

7.标志

标志字段，目前只使用左边最高位的bit，其余bit必须由client发包时清零，用来表示自己想从server或是reply agent那里得到的回复是unicase类型还是braodcast类型。

8.客户端IP地址

只有在client拥有IP，能够响应ARP请求时，并且client端处于BOUND，RENEW，或者RWBINDING状态时才会被client填充

9．你的IP地址

DHCP server分配给client的IP地址。

10.客户端IP地址

DHCP流程的下一个服务器地址，由DHCP server在DHCPOFFER和DHCPACK保证返回。

11.网关IP地址

DHCP的中继IP地址，当跨网段使用DHCP时，该字段由一个relay agent填充，并且每个agent都将hops加一，通常hops值有一个限制，超过此限制，该包将会被丢弃。

12.客户端硬件地址

占十六个字节

13.服务器名称

占64字节，可选的DHCP server的Hostname,是一个以null结尾的字符串。

14．文件名

启动文件名，128字节是一个以null结尾的字符串，当主机从网络启动时会使用。

15.选项

可选参数列表，长度不固定。

### HTTP报文

流actor可以直接方便地迁移这些状态。在无备份的情况下，这个系统在处理完数据包之后，就会将数据包发送出系统，如果是有备份的情况下，这些数据包会连同NF状态一起被送到备份的运行时系统中去。接下来系统会检测是否有停机指令，如果

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | 5 | | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| MAC帧头（14字节） | | | | | | | | | | | | | | | |  | |
| IP数据报头（20字节） | | | | | | | | | | | | | | | | | |
|  | | | TCP报文段头（一般20字节） | | | | | | | | | | | | | | |
|  | | | | | | | HTTP报文 | | | | | | | | | | |
| …… | | | | | | | | | | | | | | | | | |

## 本章小结

这一章从功能和结构的角度介绍了组成NFActor系统的各个部件。首先展示了NFActor的整体框架，主要由运行时系统、虚拟交换机和协调器三个部分组成。在运行时系统中会提供给用户网络功能的接口供用户自定义构建网络功能并且直接运用到服务链中，本章用四个小节详细地介绍了这四个组成部分的功能和结构，以及阐述了这样的框架与完成故障恢复功能之间的关系。

# 系统设计与实现

## 系统结构

接下来我们将从协调器、虚拟交换机、运行时系统三个方面来介绍NFActor的错误恢复机制。

### 协调器的错误恢复

由于协调器是单线程模块，因此我们可以将协调器维护的信息记录并复制到可靠的存储系统（如ZooKeeper）中。协调器由一个守护进程来监控，当协调器故障挂机后，守护进程会立刻重启协调器。在重新启动的时候，协调器会通过读取日志重建系统视图。

### 虚拟交换机的错误恢复

NFActor使用CRIU（一种用于检查/恢复Linux进程的流行工具），通过检查运行虚拟交换机的容器的内存映像来复制虚拟交换机。这个方法有一个主要的技术挑战是，CRIU在检查备份一个进程的时候必须停止这个进程。我们通过让虚拟交换机定期（默认情况下，一分钟）调用fork（）函数产生子进程，然后使用CRIU检查子进程来解决这个问题。以这种方式，虚拟交换机可以在不影响系统性能的情况下继续运行。

### 运行时系统的错误恢复

为了执行轻量级的运行时系统的备份，我们利用了actor抽象以及分离了NF状态和其核心处理逻辑。在运行时系统中，与流相关联的重要状态由流actor存储。运行时系统可以独立地备份每个流actor，而无需检查整个容器的映像。NFActor的备份策略与现有工作之间的最大区别是NFActor备份单独的流状态，而不是备份NF，备份对NF而言是透明的。每个流actor在同一个集群中的另一个运行是系统上备份自身，而不需要专用的备份服务器，实现非常好的可扩展性。同时，这种细粒度的备份使得NFActor实现了良好的吞吐量和快速的流恢复性能。

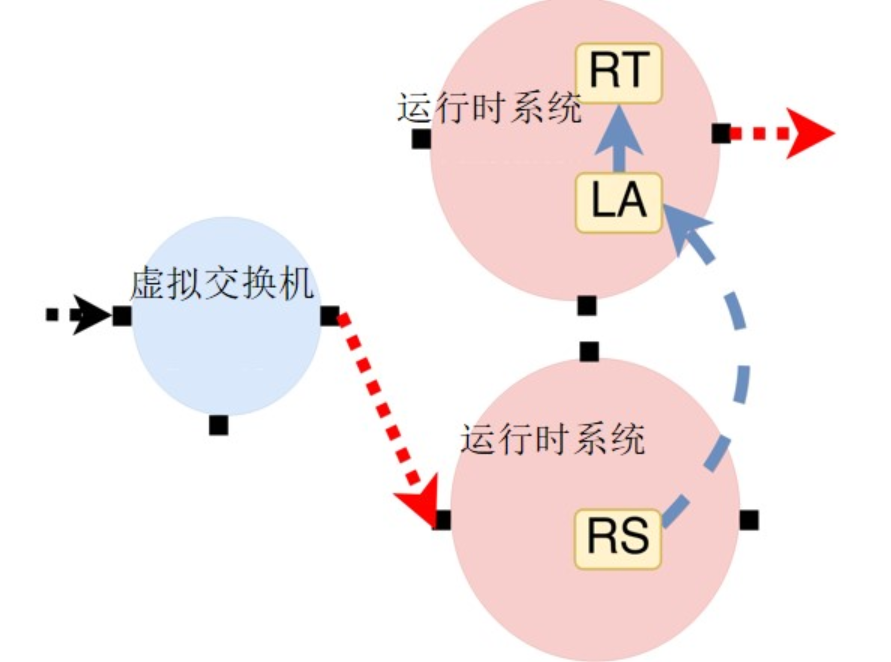


图 3‑1 流的备份

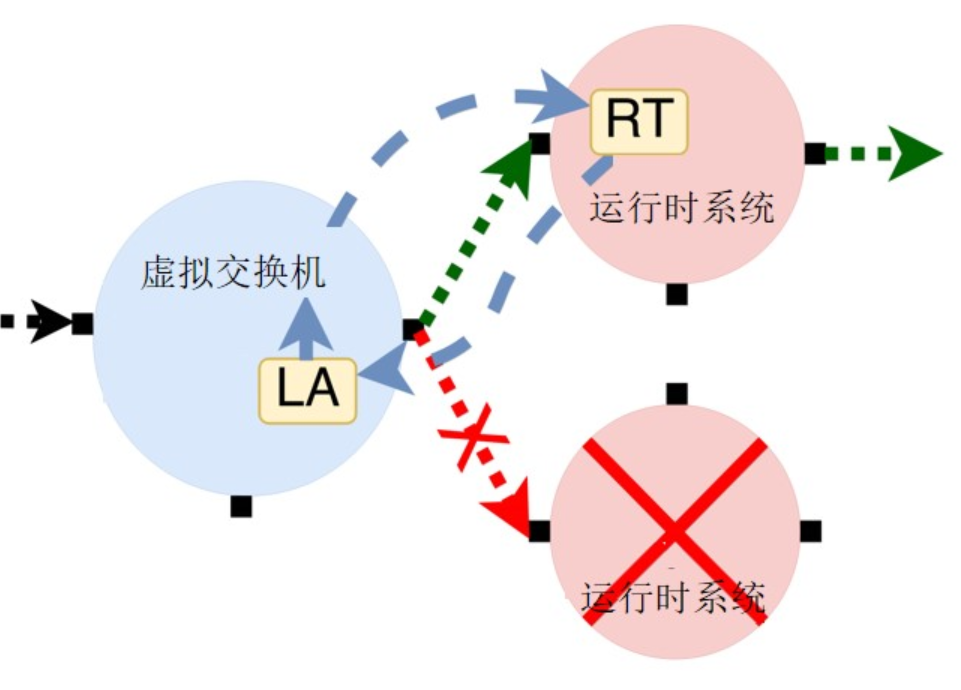


图 3‑2当源运行时系统挂机后，流的恢复

详细的流备份过程如图 3‑1 流的备份所示、流恢复过程如图 3‑2当源运行时系统挂机后，流的恢复 所示。当启动运行时系统的时候，协调器会发送RPC命令：set\_replicas（runtime\_id\_list）到其联络actor, 将处于相同集群中的运行时系统（这些运行时系统最好与主机运行时系统不在一个服务器上）设置成为这个运行时系统的备份。如果集群中没有可用的主机副本的运行时系统的话，协调器将启动新的运行时系统作为主机的备份。当在运行时系统创建一个流actor时，它通过向联络actor发送一个本地的actor消息来获取其复制目标运行时系统的id号。联络actor以轮询的方式，将协调器发送给它的运行时系统的id号转发给本地的流actor。

当流actor处理流数据包时，它会将包含当前流状态和数据包的备份消息直接发送到备份目标运行时系统的联络actor。联络actor使用数据包的5元组来检查此运行时系统是否存在这个副本流。如果没有，它将使用相同的5元组创建一个新的副本流actor，并将所有后续的这个流的备份消息转发给这个流actor。备份流actor是一种特殊的流actor，它处理接收到的备份消息，存储最新的流状态，然后直接从备份目标运行时系统的输出端口发送所包含的数据包。虽然它配置了流的服务链，但它没有实际执行这些服务链; 只有当源运行时系统故障挂机之后，它被恢复成为这个流的实际处理者之后，它才会激活服务链来处理这些收到的数据包。

如图 3‑1 流的备份所示，当运行时系统备份机制开启时，NFActor中的数据流的路径如下所示：虚拟交换机->运行时系统处理流的流actor ->有此流actor的备份的运行时系统->虚拟交换机 -> 最终目的地。此设计可以保证只有当数据包引起的所有状态更改都被备份之后，系统才会将数据包发送出去。

当运行时系统故障挂机时，协调器将向所有的备份依次发送recover的RPC命令。当运行时系统R接收到此RPC时，它会检查它是否确实存储了故障运行时系统的流副本。如果存储了，运行时系统R上的每个流副本流actor向负责将流转发到已经挂机了的运行时系统的虚拟交换机发送请求，并要求其将这个流的目标运行时系统从已挂机的运行时系统更改为运行时系统R. 当副本流actor收到了来自虚拟交换机的确认消息之后，这个流的数据包开始到达副本流actor，并在运行时系统R上成功恢复（如图 3‑2当源运行时系统挂机后，流的恢复 ）。在备份流actor成为处理流的真正操作者之后，它将立即按照上文提到过的备份方法去寻找它自己的备份。

我们的备份方法可以容忍一个运行时系统的故障。我们认为这个保证是足够的，因为两个运行时系统同时故障的概率非常低。另一种设计是重新启动挂机的运行时系统，拷贝备份的流状态，并在重启的运行时系统上恢复流actor。相比之下，我们的设计避免了重新启动流actor所造成的延时，并最大限度地减少了连续流处理的中断。缺点是将输出数据包转发到备份actor会带来额外的带宽消耗。

## 数据结构

主要包括一些头部数据结构，如MAC头，IP头，TCP头，UDP头，DHCP头，DNS头，ICMP头，ARP头等。

struct ether\_header{

uchar ether\_dhost[6]; //目的MAC地址（6字节）

uchar ether\_shost[6]; //源MAC地址（6字节）

ushort ehert\_type; //帧类型（2字节）

}

struct IPHeader{

uchar Ver\_Hlen;

uchar TOS;

ushort TotalLen;

ushort ID;

ushort Flag\_Segment;

uchar TTL;

uchar Protocol;

ushort Checksum;

uint SrcIP;

uint DstIP;

}

struct ICMPHeader{

uchar ICMPType; //类型（1字节）

uchar ICMPCode; //代码（1字节）

ushort Checksum; //校验和（2字节）

}

struct tcphdr{

ushort th\_sprot; //源端口（2字节）

ushort th\_dprot; //目的端口（2字节）

uint th\_seq; //序号（4字节）

uint th\_ack; //确认号（4字节）

uchar th\_;

uchar th\_flags; //标识TCP不同的控制信息（1字节）

ushort th\_win; //窗口大小（2字节）

ushort th\_sum; //校验和（2字节）

ushort th\_urp; //紧急指针（2字节）

uint th\_off;

uint th\_x2;

}

struct udphdr{

ushort uh\_sport; //源端口号（2字节）

ushort uh\_dport; //目的端口号（2字节）

ushort uh\_ulen; //头部长度（2字节）

ushort uh\_sum; //校验和（2字节）

}

struct arphdr{

ushort ar\_hrd; //硬件类型

ushort ar\_pro; //协议类型

uchar ar\_hln; //硬件地址长度

uchar ar\_pln; //协议地址长度

ushort ar\_op; //操作类型

}

## 动态扩展

协调器根据工作负载变化执行运行时系统和虚拟交换机的动态缩放。它充分利用快速可扩展的分布式流迁移机制，快速解决运行时系统过载问题，并能够立即关闭大部分空闲运行时系统。

协调器定期轮询所有运行时系统的负载统计信息，包括输入端口丢弃的数据包数目，当前数据包处理的吞吐量和活跃的流的数目。由于使用DPDK轮询输入端口，每个运行时系统中的工作线程始终将其CPU使用率保持在100％。因此，CPU使用率不能说明运行时系统是否已经超载。所以，协调器使用运行时系统输入端口上丢弃的数据包总数来确定是否过载，这是NFActor中非常有效的指标：过载的运行时系统无法及时轮询其输入端口的所有数据包，因此导致数据包丢包明显。每个运行时系统的最大数据包处理吞吐量由协调器记录，用于识别每个集群的空闲情况。

当运行时系统丢弃的数据包数量超过阈值（我们的实验设置为100）时，运行时系统被认为是过载了。如果集群存在一个过载的运行时系统的话，协调器将启动一个新的运行时系统，将其配置为过载运行时系统相同的服务链，并将可配置数量的流（如活跃的流的数目的一半）从过载的运行时系统迁移到新的运行时系统 ，直到所有解决所有的过载问题。如果新运行时系统过载，则会添加更多新的运行时系统。由于虚拟交换机的负载平衡流调度，使得现有运行时系统的负载大部分是均衡的，因此我们添加新的运行时系统，而不是把流迁移到现有的运行时系统中。

如果集群中所有运行时系统的当前吞吐量小于最大吞吐量的一半，协调器将执行收缩功能：它会选择吞吐量最小的运行时系统，将其所有流量迁移到其他运行时系统，并当在所有的流都已经成功迁移出去之后关闭这个运行时系统。

## 实现细节与特点

NFActor是使用C++进行编写的，除了实现的NF的代码之外，NFActor核心功能的代码一共约为8500行左右。我们自定义了一个actor库，并在Docker容器上运行我们的运行时系统和虚拟交换机。我们使用BESS作为连接不同运行时系统和虚拟交换机的数据交换连接工具，在每个服务器内构建虚拟L2网络，并将每个虚拟L2网络连接到连接所有服务器的物理L2网络。每个运行时系统的三个端口都是BESS ZeroCopy VPorts，它是采用了DPDK实现的高速虚拟端口，用于提取和传输原始数据包。使用DPDK之后，存储数据包的内存缓冲区直接可以映射到运行时系统的地址空间，避免了使用传统内核网络堆栈带来的开销。

**运行时系统的资源分配**。当在一个服务器上启动运行时系统时，协调器确保运行系统的工作线程被绑定到不同的CPU内核（不包括核心0和BESS使用的核心）。同一个服务器中的所有运行时系统的RPC线程都被集中固定在内核0上，因为它们在等待RPC请求的大部分时间都会处于休眠状态。

**自定义的actor库。**我们自己实现了一个actor的库，包括一个快速actor内分配器，一个快速actor计时器和一组用于实现新actor的C ++基类。在这个ac

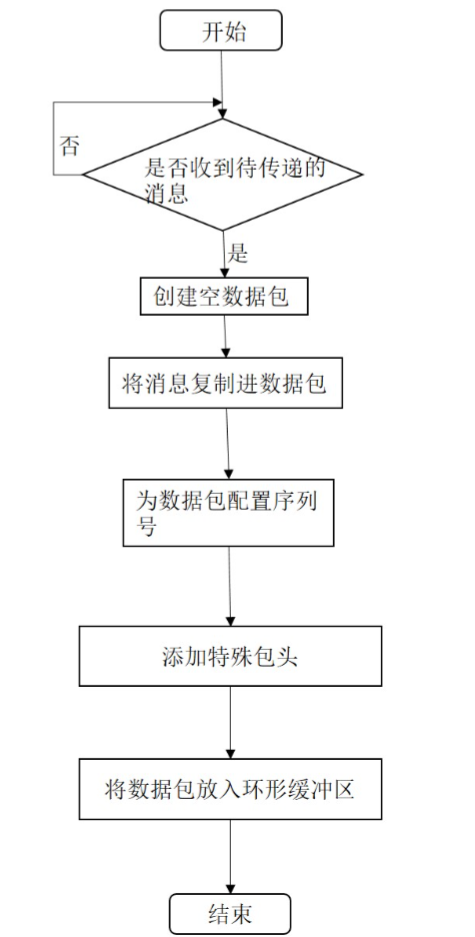


图 3‑6 actor间可靠传输模块工作流程

**可靠的远程actor消息传递。**在不同运行时系统的流actor之间传递的actor消息应该进行可靠的消息传递。我们构建了一个自定义的可靠消息传递模块，如图 3‑6，在这个模块中会将消息填入一个分组中，然后进行可靠传输。该模块为每个远程运行时系统和虚拟交换机创建一个环形缓冲区。当此运行时的流actor向远程actor发送消息的时候，模块将创建一个数据包，将消息的内容复制到数据包中，然后将数据包放入相应的环形缓冲区。这些数据包每个会被配置一个序列号，并附加一个特殊报头，以便将它们与数据包区分开来。

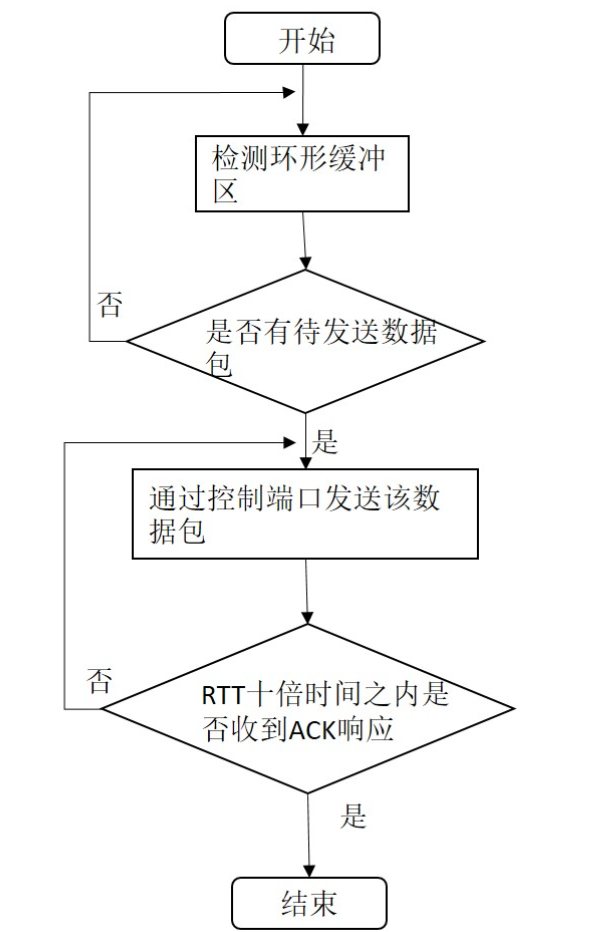


图 3‑7 工作线程发送消息模块流程

如图 3‑7工作线程从环形缓冲区中取出这些数据包，并将它们发送到相应的远程运行时系统。远程运行时系统收到这样的数据包后会发送确认消息说明自己收到了这个数据包。如果在可配置的超时（例如，RTT的10

## 本章小结

本章对NFActor各个组成部分的设计与实现进行了详细的论述。首先介绍了故障恢复、流量迁移、动态扩展三个系统管理功能的具体设计与实现。分析了在运行过程中可能遇到的种种情况并且综合各种情况选择最有利的设计方案。然后阐述了在设计过程中使用的一些设计方法和技巧比如自定义了actor的库以及使用DPDK进行actor之间的消息传递等。

# 性能测试与分析

## 测试环境

测试硬件环境如下：

系统：MacOS 10.13.3

处理器：2.3 GHz Intel Core i5

内存：8GB 2133 MHz LPDDR3

软件环境如下：

Qt：Qt 5.10.0（Clang 7.0（Apple）,64 bit）

QtCreator：Qt Creator 4.5.0

WireShark：Wireshark Version 2.4.3

## 运行测试

我们在通过测试一个包含有多种类型数据包的pcap文件来检验系统的功能。

首先在程序的初始界面选择该pcap文件，如图

（图 初始文件选择界面）

然后打开该文件的总揽界面如下。

（图 总揽初始界面）

对ICMP类型数据包进行测试：

（图 ICMP）

对TCP包

（图 TCP）

对HTTP包

（图 HHTP）

对 UDP包

（图 UDP）

对 DNS包

（图 DNS）

对 DHCP包

（图 DHCP）

对ARP包

（图 ARP）

## 本章小结

本章通过对不同类型的数据包进行解析测试。展示了系统对多个协议的支持情况。目前系统主要支持一下协议：ICMP、TCP、UDP、ARP等，同时在应用层协议有HTTP、DNS、DHCP等。系统通过表格的形式将数据包显示出来，然后通过点击表格的数据包条目，在文本框中进一步显示详细包信息，同时在下放将当前数据包的二进制数据显示给用户。

# 总结与展望

在这几个月的设计与实现中，NFActor基本实现了预期的所有要求：

1. 我们自定义了actor模型的库，并将其作为构建分布式NFV系统的基础，这个actor库中主要定义了actor之间传递的消息类型和actor之间的消息传递函数。消息传递函数又包括远程消息传递和本地消息传递两种类型，在这个基础上，我们实现了流量迁移错误恢复等功能。
2. 我们采取了一种新颖的单流执行的原理，让每个流由一个专门的流actor来进行管理，流actor内部为这个流提供专门的服务链服务。在进行流量迁移和错误恢复等功能的时候，流actor也会负责与其他系统的actor进行通信并且传递网络功能状态。这种分布式的设计使得协调器不需要过度地干预流量迁移与错误恢复的细节，降低了协调器的工作负载。
3. 我们设计了实现NF的API用来提供流的状态和NF处理逻辑的分离。用户可以通过这些API轻易地自定义一些符合系统要求的网络功能，并且能够直接将这些网络功能应用到运行时系统的服务链中去。
4. 我们使用DPDK实现了actor之间的消息传递机制，actor之间的消息会被封装成数据包通过控制端口互相传递，我们还专门设计了可靠传输模块来保证actor之间的可靠消息传递，我们还使用GRPC实现了协调器与运行时系统之间的命令传递机制。协调器通过一系列的RPC命令，就可以很容易地对虚拟交换机和运行时系统进行控制。
5. 我们还精心设计了NFActor的运行时系统和虚拟交换机的结构框架，主要分为工作线程和RPC线程，工作线程又分成了一系列的模块，各个模块互相配合协调工作，RPC线程用来接收来自协调器的RPC命令。让其实现了快速的流量迁移备份和缩放。

本文主要完成了如下工作：

1. 分析了×××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××
2. 从解决了…的基本问题入手，讨论了
3. 有哪些

接下来讨论一些本课题还有一些目前存在的问题和一些日后可进一步完善的地方。

1. 为了实现流状态和NF核心逻辑之间的清晰分离，NFActor需要使用一组API来重写NF。随着NFV的发展，越来越多的新NF将被创造出来。如果使用actor模型来构建NFV系统的设计被广泛接受的话，在这个API框架下重写NF是可行的。如果并没有被广泛接受，那么在这个框架下重写NF就会显得非常复杂和繁琐。
2. 虽然NFActor主要是针对有状态的NF而设计的NFV系统，但它也可以轻松处理无状态的NF。无状态的NF也可以受益于快速的分布式流迁移，因为它消除了由直接改变流的路径引起的分组乱序的可能。
3. NFActor目前只针对单个流的状态。当前的NFActor框架不能正确处理共享状态，即多个流共享的一个状态[3]。原因是我们目前的API设计无法正确分离这些共享状态。与其他流量共享状态的迁移和备份可能导致不可预测的错误。我们目前想到的可能的解决方案是在流actor中实现一个处理器来专门管理这些共享状态。我们打算在以后的研究中来实现这个功能。

NFActor可能会错误地处理具有数据包封装的流。NFActor使用5元组来区分流。但如果原始流包使用类似的包头封装，则不同的流可以共享相同的5元组。这对于通过相同的VxLAN隧道发送的流是一种常见的情况。在这种情况下，这些流由相同的流actor使用相同的服务链来处理，并更新同一个状态，这样一来肯定会导致错误的发生。如果我们知道输入流使用什么样的封装方式，我们就可以在虚拟交换机中添加解封装功能，以正确地提取出不同的流。这个功能我们也打算在以后的研究中实现。

致 谢

整个毕业设计从题目的选择到最后实现所有的功能用了大概四个月左右的时间，在这段时间里，我得到了很多的帮助，也从中学到了很多。

首先要感谢的是我的导师周正勇老师，在整个设计过程中，周正勇老师给予我无私的帮助，为我提供了种种专业知识上的指导，和一些富于创造性的建议，在我遇到问题，苦于无法解决时，老师总是能及时给予我细心而专业的回答，在后期答辩之前，周老师也仔细浏览了我的开题报告，文献翻译等内容并提出了许多珍贵的修改意见，周老师一丝不苟的作风，严谨求实的态度使我深受感动。正是在周老师的帮助下，我得以顺利的完成此次毕业设计。在此向周正勇老师表示深深的感谢和崇高的敬意！

其次，在临近毕业之际，我还要借此机会感谢这四年中给予我诸多教诲和帮助的各位老师们，是你们的教诲让我对计算机领域有了大致的了解并为以后进行更深入的研究打下了夯实的基础。感谢他们四年来的辛勤栽培，不积跬步何以致千里，各位任课老师认真负责，在他们悉心的帮助和支持下，我能够很好的掌握和运用专业知识，并在设计中得以体现，顺利完成毕业论文。

此外，我还要感谢各位同学和我的各位室友，在毕业时即这段时间里，你们给予了我很多启发，提出了许多宝贵的意见。当然，除了在学习上的互帮互助，还有在生活上的互相照顾。对于你们的帮助和支持，在此我表示深深的感谢！

最后，感谢母校，感谢学院。学院为我们提供了丰富的学习资源与优秀的学习环境，让我们在一个良好的氛围中度过了这美好的大学。

参考文献

1. Palkar S, Lan C, Han S, et al. E2: a framework for NFV applications[C]. Symposium on Operating Systems Principles. ACM, 2015:121-136.
2. Bremler-Barr A, Harchol Y, Hay D. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions[C]. Conference on ACM SIGCOMM 2016 Conference. ACM, 2016:511-524.
3. Sekar V, Egi N, Ratnasamy S, et al. Design and implementation of a consolidated middlebox architecture[C]. Usenix Conference on Networked Systems Design and Implementation. 2011:24-24.
4. Anderson J W, Braud R, Kapoor R, et al. xOMB: extensible open middleboxes with commodity servers[C]. Eighth ACM/IEEE Symposium on Architectures for NETWORKING and Communications Systems. ACM, 2012:49-60.
5. A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual Middleboxes as First-class Entities[R]. Technical report, UW-Madison 2012.
6. Zhang W, Liu G, Zhang W, et al. OpenNetVM: A Platform for High Performance Network Service Chains[C]. The Workshop on Hot Topics in Middleboxes and Network Function Virtualization. ACM, 2016:26-31.
7. Gember-Jacobson A, Viswanathan R, Prakash C, et al. OpenNF: enabling innovation in network function control[C]. ACM Conference on SIGCOMM. ACM, 2015:163-174.
8. Rajagopalan S, Dan W, Jamjoom H, et al. Split/merge: system support for elastic execution in virtual middleboxes[C]. Usenix Conference on Networked Systems Design and Implementation. 2013:227-240.
9. J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr[C]. In Proc. of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI’16), 2016.
10. Ballani H, Costa P, Gkantsidis C, et al. Enabling End-Host Network Functions[J]. ACM SIGCOMM Computer Communication Review, 2015, 45(4):493-507.
11. Paxson V. Bro: a system for detecting network intruders in real-time[C]. Conference on Usenix Security Symposium. USENIX Association, 1998:3-3.
12. Yoan N. Iptables[M]. Miss Press, 2013.
13. Sherry J, Gao P X, Basu S, et al. Rollback-Recovery for Middleboxes[C]. ACM Conference on Special Interest Group on Data Communication. ACM, 2015:227-240.
14. Chrome G. HTTP persistent connection[J]. 2015.
15. Surhone L M, Tennoe M T, Henssonow S F, et al. Ffmpeg[M]. 2010.
16. Housley R, Hoffman P. Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP[J]. Ietf Rfc Sri Network Information, 1999, 11(3):82--89.
17. Maheshwari A, Sharma A, Ramamritham K, et al. TranSquid :Transcoding and Caching Proxy for Heterogenous E-Commerce Environments[C]. International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/e-Business Systems. IEEE Computer Society, 2002:50.
18. J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr[C]. In Proc. of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI’16), 2016.
19. NFV Paper [EB/OL].https://portal.etsi.org/NFV/NFV White Paper2.pdf,2017-05-05.
20. Hwang J, Ramakrishnan K K, Wood T. NetVM: high performance and flexible networking using virtualization on commodity platforms[C]. Usenix Conference on Networked Systems Design and Implementation. USENIX Association, 2014:445-458.
21. S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy. SoftNIC: A Software NIC to Augment Hardware[R]. Technical report, EECS Department, University of California, Berkeley, 2015..
22. Martins J, Ahmed M, Raiciu C, et al. ClickOS and the art of network function virtualization[C]. Usenix Conference on Networked Systems Design and Implementation. USENIX Association, 2014:459-473.
23. Panda A, Han S, Jang K, et al. NetBricks: taking the V out of NFV[C]. Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2016:203-216.
24. A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual Middleboxes as First-class Entities[R]. Technical report, UW-Madison 2012.
25. Palkar S, Lan C, Han S, et al. E2: a framework for NFV applications[C]. Symposium on Operating Systems Principles. ACM, 2015:121-136.
26. Rajagopalan S, Dan W, Jamjoom H. Pico replication:a high availability framework for middleboxes[C]. Symposium on Cloud Computing. 2013:1-15.
27. Sherry J, Gao P X, Basu S, et al. Rollback-Recovery for Middleboxes[C]. ACM Conference on Special Interest Group on Data Communication. ACM, 2015:227-240.
28. Qazi Z A, Tu C C, Chiang L, et al. SIMPLE-fying middlebox policy enforcement using SDN[J]. Computer Communication Review, 2013, 43(4):27-38.
29. Zhang W, Hwang J, Rajagopalan S, et al. Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization[C]. International on Conference on Emerging NETWORKING Experiments and Technologies. ACM, 2016:3-17.
30. Bremler-Barr A, Harchol Y, Hay D. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions[C]. Conference on ACM SIGCOMM 2016 Conference. ACM, 2016:511-524.

附录：大学期间发表或提交的论文

1. A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual Middleboxes as First-class Entities[R]. Technical report, UW-Madison 2012.
2. Palkar S, Lan C, Han S, et al. E2: a framework for NFV applications[C]. Symposium on Operating Systems Principles. ACM, 2015:121-136.

**成 绩 评 定**

**指导教师评定意见**

一、对毕业设计（论文）的学术评语（应具体、确切、实事求是）

|  |
| --- |
| 陈延良同学学习态度评价…工作时间进度评价…论文内容与价值…论文结构评价…论文规范性评价…作者学术水平评价…，论文（基本）达到本科毕业设计论文的要求，同意参加本科毕业答辩。  导师评语建议学生先写个草稿，由导师审阅论文时一并修订成型，最终评语可以直接打印，导师签字即可，）  （答辩前导师需要给出具体成绩，下表评分中合计部分更新域可自动运算求和） |

二、对毕业设计评分

(1)理工医科评分表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 调研论证  (10分) | 外文翻译  (5分) | 设计(论文)撰写质量  (10分) | 学习态度  (10分) | 基本理论和基本技能  (50分) | 创 新  (15分) | 合 计  (100分) |
| 得分 | **8** | **4** | **8** | **8** | **40** | **12** | **80** |

(2)文科评分表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 文献阅读与文献综述  (10分) | 外文翻译  (10分) | 论文撰写质量  (10分) | 学习态度  (10分) | 学术水平、论证能力和创新  (60分) | 合 计  (100分) |
| 得分 |  |  |  |  |  |  |

指导教师签字：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 年 月 日

**答辩小组评定意见**

一、评语（根据学生答辩情况及其设计（论文）质量综合评价）

|  |
| --- |
|  |

二、评分

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 答 辩 情 况 | | 论 文 质 量 | | 合 计  (100分) |
| 答辩情况  (15分) | 回答问题情况  (25分) | 规范要求与文字表达  (20分) | 学术水平  (40分) |
| 得分 |  |  |  |  |  |

答辩小组长签字：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 年 月 日

**毕业答辩及成绩评定说明**

1. 毕业答辩
2. 答辩前，答辩小组应详细审阅每个学生的毕业设计（论文），为答辩做好准备。
3. 严肃认真组织答辩，开好答辩会。
4. 指导教师应参加所指导学生的答辩会，但评定其成绩时宜回避。
5. 答辩中要做好记录以供成绩评定时参考。
6. 成绩评定
7. 答辩前每个学生都要将自己的毕业设计（论文）在指定时间内交给指导教师，由指导教师审阅，写出评语并预评分。
8. 答辩工作结束后，答辩小组应举行专门会议进行讨论，在参考指导教师预评结果的基础上，结合学生毕业设计（论文）质量和学生答辩情况，综合评定每个学生的成绩。
9. 院（系）对专业答辩小组提出的优秀和不及格的毕业设计（论文），要组织院（系）级答辩，最终确定成绩，并向学生公布。
10. 各专业学生的最后成绩应符合正态分布规律。
11. 请用蓝、黑钢笔手写或五号宋体字编辑，签名须手写，A4纸双面打印。

**毕业设计（论文）成绩评定**

|  |
| --- |
| 班号：**计科201408班** 学生姓名：**陈延良**  综合成绩：\_\_\_\_\_\_**86**\_\_\_\_\_\_\_分（折合等级\_\_\_\_\_**良**\_\_\_\_\_\_\_）  答辩小组长（签名）：\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 年 月 日 |

1. [↑](#footnote-ref-1)