

题 目 一种基于微服务架构的PAAS平台

软件学院 院（系） 软件工程 专业

学 号 71113333

学生姓名 陈浩远

校内导师 刘奇其

企业导师 薄宏剑

起止日期 2017年1月 ～ 2017年6月

设计地点 华为技术有限公司南京研究所

一种基于微服务架构的PAAS平台

# 摘 要

伴随电信行业3G/4G等业务的增加，数据量出现了爆炸式的增长，数据的规模已经达到了PB级别。海量数据带来了大量的信息，同时也迫使电信行业为了处理这些数据而不得不使用hadoop/spark等分布式大数据技术。由于电信运营商自身结构复杂，产品和业务繁多，电信用户多，每天能产生大量的数据，因此产生了很多不同的hadoop/spark需求，对于如何管理和调度众多的大数据任务也成为了运行商头疼的一件事。

华为大数据部门决定开发一个PAAS平台用来统一管理和调度hadoop/spark任务（以后称之为统一运行平台/Runtime-Platform/RP）。本文结合我所参与的华为大数据统一管理平台的经验，详细介绍了华为这个PAAS平台的设计。该项目采用了分布式的开发理念，选择了较为流行的微服务架构，并且使用轻量级的docker容器单独部署服务，因此，该项目具有极高的可用性和鲁棒性。

关键词：微服务；分布式；PAAS；大数据；docker

A PAAS Platform Based on Micro Service Architecture

# ABSTRACT

With the telecommunications industry 3G / 4G and other business increases, the amount of data appeared explosive growth, the size of the data has reached the PB level. Massive data has brought a lot of information, but also forced the telecommunications industry in order to deal with these data had to use hadoop / spark and other large distributed data technology. Because telecom operators are complex in their own structure, many products and services, many telecom users can produce a lot of data every day, resulting in a lot of different hadoop / spark needs, how to manage and schedule a large number of big data tasks has become a runner A headache for one thing.

Huawei Big Data Department decided to develop a PAAS platform for unified management and scheduling hadoop / spark task (later called the unified operating platform / Runtime-Platform / RP). This paper introduces the experience of Huawei's PAAS platform in combination with the experience of Huawei's large data unified management platform. The project uses a distributed development concept, selected a more popular micro service architecture, and the use of lightweight docker containers to deploy services alone, so the project has a high availability and robustness.

Keywords: Micro service; distributed; PAAS; big data; docker

目 录

[摘 要 i](#_Toc484143549)

[ABSTRACT ii](#_Toc484143550)

[第一章 引 言 1](#_Toc484143551)

[1.1 选题背景和意义 1](#_Toc484143552)

[1.2 国内外研究现状 1](#_Toc484143553)

[1.3 论文主要内容 2](#_Toc484143554)

[1.4 论文组织结构 3](#_Toc484143555)

[第二章 系统需求分析 4](#_Toc484143556)

[2.1 系统功能需求 4](#_Toc484143557)

[2.1.1 网络流量采集 5](#_Toc484143558)

[2.1.2 网络流量分析 6](#_Toc484143559)

[2.1.3 流量监测配置 6](#_Toc484143560)

[2.2 系统数据需求 7](#_Toc484143561)

[第三章 系统方案设计 8](#_Toc484143562)

[3.1 系统架构设计 8](#_Toc484143563)

[3.2 网络流量采集功能设计 9](#_Toc484143564)

[3.3 网络流量分析功能设计 12](#_Toc484143565)

[3.4 流量监测配置功能设计 13](#_Toc484143566)

[3.4.1 基础配置 13](#_Toc484143567)

[3.4.2 高级配置 13](#_Toc484143568)

[3.4.3 组配置 14](#_Toc484143569)

[3.5数据库表结构设计 14](#_Toc484143570)

[3.5.1 汇聚数据表结构 14](#_Toc484143571)

[3.5.2 配置数据表结构 15](#_Toc484143572)

[第四章 系统实现与测试 20](#_Toc484143573)

[4.1 网络流量采集功能的实现 20](#_Toc484143574)

[4.1.1 网络流量采集流程 20](#_Toc484143575)

[4.1.2 网络流量接收服务 20](#_Toc484143576)

[4.1.3 网络流量聚合服务 22](#_Toc484143577)

[4.2 网络流量分析功能的实现 24](#_Toc484143578)

[4.3 流量监测配置功能的实现 31](#_Toc484143579)

[4.4 系统测试 35](#_Toc484143580)

[4.4.1 测试目的 35](#_Toc484143581)

[4.4.2 测试环境 36](#_Toc484143582)

[4.4.3 测试过程及结果 36](#_Toc484143583)

[第五章 工作总结与展望 38](#_Toc484143584)

[5.1 工作总结 38](#_Toc484143585)

[5.2 工作展望 38](#_Toc484143586)

[致 谢 39](#_Toc484143587)

[参考文献 40](#_Toc484143588)

# 第一章 引 言

## 1.1 选题背景和意义

当下，互联网技术飞速发展，网络的规模愈发壮大，中国互联网络信息中心的一份关于国内互联网发展状况的研究报告显示，截至2016年12月，中国网民的规模达到了7.31亿，全年共计新增网民4299万人。其中手机网民达到了6.95亿。与此同时，互联网的普及率达到了53.2%，相比2015年提高了2.9个百分比。因特网俨然已成为人们日常生活的必需品，无时不刻都在影响着人们的生活方式。

互联网规模的高速发展，为用户提供了多种多样的服务、更高的带宽以及可预测的QoS，网络用户可以更方便、准确的根据自己的需求获得想要的信息或资源，但是另一方面，网络规模的爆炸性增长，导致种类繁多的网络应用产生的数据也越来越庞大，其对带宽的要求也逐渐升高。与此同时，一些企业和大型机构的内部也愈发的依赖网络，企业之间的信息传递变得频繁，互联网数据中心（IDC）渐渐进入了人们的视野[1]。

面对如今庞大且复杂的网络结构，使得网络流量的模型不断更新换代，电商为了满足客户的需求将会面对巨大的挑战。网络流量的暴增，如何在保证网络基础设施能够正常运作的条件下做到更有效的使用网络带宽，更有序的管理互联网数据中心间的网络流量，已经引起了人们的高度重视[2]。通过对互联网中的流量进行解析，我们可以及时的发现网络中各种不正常行为，从而能够对网络流量进行管理以保证网络带宽的利用率，保障网络的高性能运作。

电商的管理者需要及时了解网络中的各种业务，及时掌握IDC之间网络流量的各项指标，发现并及时解决已经存在和可能存在的网络问题，这对于一个企业的核心利益有着很高的重要性。电商在管理网络时经常会遇到各种各样的问题，为了更加高效的管理网络，企业内部需要一套系统能够让网络的管理者知晓实时的网络使用状况。网络流量监测系统的意义就在于帮助电商管理者对IDC间网络流量的组成以及趋势进行实时的监控与分析[3]，从而更有效的利用网络资源，控制网络成本，并为网络的规划、调整、优化、发展等方面提供坚实的基础。

## 1.2 国内外研究现状

在21世纪初期，网络流量监测就已经出现，然而，由于当时的网络并没有非常普及，网络带宽很小，网络流量数据规模也不大，因网络流量导致的企业利益问题在当时也不明显，使得很多企业对网络流量监测并不重视。在网络不断发展的同时，网络流量逐渐膨胀，泄露公司机密、丢失客户资源的事件频频发生[4]，这会给一个企业带来沉重的打击，网络流量的监测逐渐引起人们的注视。

传统流量监测的方式有SNMP，端口镜像[5]等，SNMP方式只能计数到设备端口流量的大小，不能统计到流量发送和接收主机IP，端口，协议等信息，会话流量不能做到基于应用程序和主机之间的会话。只有统计到的流量大小而没有流量的详细构成和流向信息；端口镜像方式是完全复制流量然后去分析，需要支持逐个数据包分析的探针或分析系统。这种方式受到镜像端口转发能力的限制，并占用大量的网络带宽，一般用于分析少量端口和小流量的情况，无法做到大规模网络流量的流量分析。一种名为NetStream的流量监测技术解决了传统的流量统计方式的限制，它利用网络中现有的支持流技术的网络设备来完成流量统计，不需要额外的部署探针，只需要在现有设备上做简单配置，即可完成全网流量分析，并提供流量大小、流量详细组成和流量流向的分析报告。

由于IP网络的非连接性质，网络中不同业务的通信可能是由任何终端设备发送到另一终端设备的一组IP数据包,这组数据包实际上构成了数据流网络中的一个业务数据流。大部分数据流量很短，且是突发的双向数据流[6]。NetStream技术就是这样一种基于网络数据流信息的统计与分析的技术，它可以对网络中的流量和带宽使用情况进行分类和统计，基于业务和不同的QoS给出统计记录。从而提供如下应用：

（1）应用的识别：通过NetStream技术，可以分析出网络流中包含的应用信息，NDA可将NetStream统计数据分析后基于应用显示统计结果，以图表的方式展现给用户，应用流量图表中可以详细显示应用的协议、端口、流量等信息。系统提供了查看端口的功能，管理员可查看 TCP-APP 中所有的端口以及各个端口上流量的会话数据。并且可以在查看端口页面中快速定义新的应用。网络监视人员还可以获得应用所占全部数据量的比例等信息[7]。

（2）网络监控：企业局域网通过AR连接到Internet网络。通过在AR出口部署NetStream，对Internet出口进行实时的流量监控，可以分析出各种业务占用的出口带宽的情况，监视非工作需要的Internet访问，并且在攻击发生的时候及时在NDA上发出告警，以便网络管理人员分析问题原因，排除故障[8]。

（3）流量阈值告警：系统提供用户应用、会话、设备、借口等维度的流量告警功能。用户可自主选择监控对象设定阈值、定义告警级别，当监控对象的流量超过阈值时将产生告警，并发送邮件或短消息通知用户。网络流量告警的产生和清除都遵守一定的规则。

（4）网络的规划：系统为管理人员提供数据流的即时速率、平均速率、包大小等详细信息[9]，网络管理人员可以通过这些信息对流量的趋势和组成进行分析，从而更有效的利用网络资源，更加高效的管理网络，并为网络的调整、优化等方面提供坚实的保障。

## 1.3 论文主要内容

本文主要围绕基于NetStream技术的电商IDC间网络流量监测系统的设计与实现。该系统重点是帮助用户实现网络流量可视、故障可查、规划可依的网络透明化管理目标，提供多维度详细的流量分析报表，帮助管理员及时了解网络流量动态，确保网络带宽得到充分合理的使用，保障网络长期稳定高效运行。

## 1.4 论文组织结构

本论文主要介绍了基于NetStream的电商IDC间网络流量监测系统的设计与实现，主要内容组织结构如下：

第一章 引言，介绍了论文的选题背景以及实现此课题的意义，并说明了目前国内外网络流量监测技术的现状和本论文的主要内容。

第二章 系统需求分析，对系统所需要实现的功能以及采集数据的需求进行了分析。

第三章 系统方案设计，结合需求分析给出了系统的整体设计方案，并对各功能模块以及需要采集的指标的数据表结构进行了详细设计。

第四章 系统实现与测试，对系统的采集、分析模块的功能进行了实现，包括流量的接收、聚合、传输，以及NMC控制台对流量的处理过程和用户配置功能的具体实现，并对系统功能进行了测试。

第五章 工作总结与展望，对论文内容进行了总结，以及对系统当前不足提出的一些日后可以实行的优化改进建议。

# 第二章 系统需求分析

## 2.1 系统功能需求

在网络技术急速发展的今天，电商为了满足客户需求面对着巨大的挑战，流量的剧增致使老式的网络流量统计分析方法存在着种种缺陷，不再适合目前互联网的环境。电商在管理网络时经常会遇到各种各样的问题，为了更加高效的管理网络，企业内部需要一套能够满足更高要求的系统能够让网络的管理者知晓实时的网络使用状况。

传统的SNMP或者端口镜像等流量监测方式因为种种限制并不能满足当下网络规模的监控要求，基于网络流的NetStream流量监测技术则解决了传统的流量统计方式的限制，它利用网络中现有的支持流技术的网络设备来完成流量统计，不需要额外的部署探针，只需要在现有设备上做简单配置，即可完成全网流量分析，并提供流量大小、流量详细组成和流量流向的分析报告。

NetStream是H3C公司研发的一种基于网络数据流信息统计与分析的技术[10]。NetStream主要根据每个报文的7元组来区分不同的流，七元组包括报文的目的端口、源端口号、协议号、目的IP、源IP、输入或者输出接口、TOS。NetStream对每条独立的流进行分析与也可以对同种类的流进行聚合。从而为网络的管理者提供网络流量的使用状况。

NetStream网络流量监测系统包括三部分：流量输出器（NDE，Network Data Exporter）、

网流采集器（NTC，Network Traffic Collector）和网络管理控制台（NMC，Network

Management Console）[11]，它们之间的关系如图2.1所示。

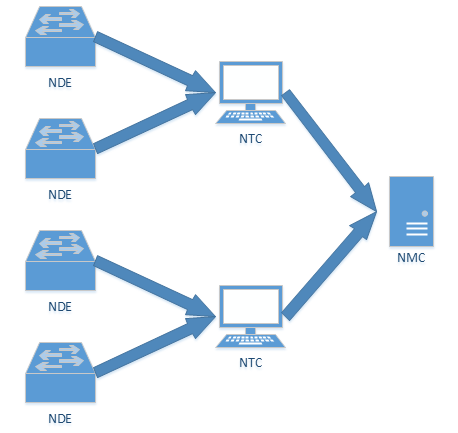


图2.1 NetStream系统组成关系

NDE（流量输出器）是一个支持“流”技术的网络设备，它主要作用是分析和处理网络流，并从中提取出匹配的数据流统计信息，接下来可以进行某些数据分类、聚合等操作，并将处理好的流量信息输出到NTC设备。NetStream网流分析组件捕获的网络设备流包括 Huawei的 NetStream、Cisco的NetFlow 以及 sFlow。

NTC（网流采集器）是一个负责分析NDE输出数据包的运行在系统中的应用组件，主要对输出的数据进行解析，过程中记录原始流量的日志信息，并对解析好的数据进行封装发送到网络管理控制台。网流采集器可以从多个NetStream的设备采集数据，并对数据进行聚合和过滤。

NMC（网络管理控制台）是一个流量分析工具[12]，提供智能的流量分析数据模型，将采集器上报的数据根据不同的模型进行分析和存储，提供多维度的流量分析数据展示给用户。NMC基于B/S架构，通过Web浏览器提供直观的、图形化的管理分析界面。

其中NTC和NMC两个部分可以集成安装在一台服务器上运行，也可以在不同的服务器上进行分机安装。

### 2.1.1 网络流量采集

作为一个网络流量监测系统，最基本的功能即是流量的采集功能。当网络设备配置并下发命令开启NetStream功能后即会对网络流量进行采样、统计、老化，最终输出到NTC进行数据的聚合、封装处理，其中NDE的统计信息提取与输出的一般过程如图2.2所示。

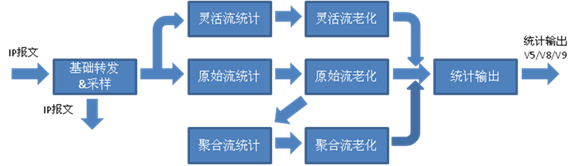


图2.2 NDE的统计信息提取与输出

NDE的基础转发单元收到IP报文后，除按照路由进行转发外，还可以根据具体的配置情况，分析转发的流量并提取统计信息，如果配置了采样功能，则对采样后的流量分析并提取统计信息。统计信息按照配置的统计方式建立统计记录 [13]，直接根据流量信息可以建立原始流和灵活流的统计记录，原始流老化后可根据配置建立聚合流统计记录。老化的统计记录，其数据在统计输出单元被封装成统计输出报文后，发送到网络流量采集器NTC。NetStream可以准确统计每一个流经设备的报文，但是在骨干网络接口上流量非常大，对NDE的处理压力也会增大。因此，在不用于准确计费的场景下，NetStream可以通过采样网络流量的方式来降低NDE的压力。NTC和NMC设备可以通过采样比来较准确的还原流量统计值。

为了满足客户对网络流量监测的需求，网络设备必须统计出相关信息来对流量的组成以及速率等进行分析，同时为了减轻NTC或者NMC设备负荷，其使用多种灵活的统计方式。不同的统计方式，以各自特征生成“流表”来记录“流”相关的数据。网络上的流基本上都是短时间阵发的，而NDE的流表容量是一定的，这就需要NDE能否有机制保证删除暂时不需要的流表，为后面到来的流提供流表空间。这个机制被称为统计流老化（简称流老化）。为了满足客户需求，统计流老化后，将组装成一定格式的统计记录报文，发送到NTC或NMC，这个过程被称为统计信息输出。NDE在进行流统计时，按照配置好的格式进行数据封装生成统计记录，统计记录报文基于UDP传输，报文简要格式如图2.3所示。

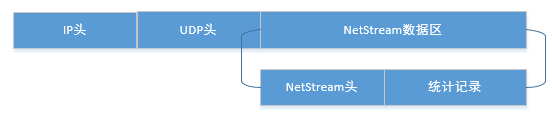


图2.3 传输报文格式

当统计记录通过NDE设备发送到NTC设备时，接口服务对不同格式的统计记录报文进行识别，并生成对应版本报文格式的流，为了实现对IP组、接口、应用等不同维度的流量进行分析，需要对所有的流分别进行各种维度的逐流1分钟聚合操作，并生成对应维度的数据文件，最后发送至NMC进行分析操作。

### 2.1.2 网络流量分析

为了达到对网络中流量进行监测的目的，帮助网络管理人员实时查看网络流量动态，网络流量监测系统的分析功能必不可少。NMC接收到NTC按照维度进行汇聚的数据文件应该作为基础数据及时入库，为了体现网络流量在不同时间内的趋势以及组成情况，对基于NTC的1分钟汇聚信息进行更高时间粒度的汇聚操作并同时入库保存，对于不同维度的数据，因为有不同的监控要求，应当提供不同的存储策略。界面使用图表的方式使得流量组成情况更加直观。

监测系统的全局概览提供各维度全网流量的流量动态，通过全局概览可以快速查看各维度的流量信息。由于网络流量的数量可大可小，持续时间可长可短，因此系统需要提供不同时间粒度的流量监控选项方便查看不同时间段的流量情况，对于特殊需求，也可以自定义流量监测的时间段。

全局流量概览只是对所有维度网络流量情况的大体分析，不能具体显示出某个维度的具体流量信息，因此需要提供单个维度的流量分析，界面展现该统计维度下的Top N流量组成排行，当选择一个具体的统计对象后，再单独展示该统计对象的流量趋势、速率、流量大小以及该对象相关的流量数据组成。不同维度的界面组成大致相同，以下不再赘述。

### 2.1.3 流量监测配置

网络流量的监控是对一台台网络设备进行监控，不同的场景下的设备数目不同，所以客户需要对设备进行管理，包括设备的增加、监控、删除等操作。对于不用的设备，也可能由不同的采集器进行流量的采集，采集器的配置也必不可少，包括对采集器的添加、删除以及采集器管理的设备的添加、删除，用户可以修改采集器的IP地址以及监听端口，命令通过JAVA的消息通信来通知采集器实施修改。同时对于系统中已经添加的设备，采集器自动发现流经该设备的流量的出和入接口，并提供接口的配置界面，以便客户的接口监控、去监控或者删除等操作。

更高级的配置包括协议配置、应用配置、流量告警功能，不同的场景下用户对于协议的监控有不同的要求。协议配置可以自定义对某些协议的监控以及去监控，同理应用配置可以自定义对某些应用的监控以及去监控，在客户通过对网络流量分析情况的观察并可以根据当前情况对某个维度的流量配置阈值告警，当该维度有流量超过定义的阈值后即会推送告警信息，用户可以查看不正常流量的具体信息，及时定位出现问题的位置以及原因。

当设备的数目比较多时，接口数据也可能随之变多，同理应用、IP的数目也可能变得很庞大，客户面对大量的接口或者应用信息时，对网络状况的分析将会变得很困难，这就需要提供组配置，客户可以根据自己的需求将一类接口创建为一个接口组，然后按照一个个组去分析流量情况，这样会使得分析变得更加方便，同理，应用组，IP组也是如此。

## 2.2 系统数据需求

整个系统的涉及到的数据大致分为两部分，一部分存储各维度的汇聚数据，另一部分存储系统的配置信息，这些数据都保存在NMC端。其中，汇聚数据主要是NTC发送来的应用、接口、IP组等维度的汇聚数据文件，配置数据主要是用户配置的设备、接口、采集器、应用等信息。

所有维度的汇聚数据表都应当有一条唯一且自增长的主键ID来区分每一条数据，同时为了区分每条流的数据，所有汇聚数据表都应该有设备ID、接口ID、流的方向，由于界面需要展示流量的大小，所以作为基础分析数据应该保存每条流的字节数以及包数，对于不同维度，应当拥有独有的ID用来识别该条数据应属于那个维度，例如应用维度应该有一个此维度汇聚表独有的APPID，而接口维度因为需要计算当前速率、平均速率、最大、最小速率等，则需要存储该时间粒度内的最大、最小、平均字节数。

配置数据主要存储系统的每项配置内容，基础的配置包括设备、接口、采集器信息，这些是作为流量监控的必备条件，其中设备表存储系统中发现的设备的ID、名称、IP地址、描述、监控状态等信息，接口表存储所有设备的接口ID、名称、索引、监控状态采样比等信息。采集器表存储采集器ID、采集器IP、名称、描述、端口以及绑定的NMC的IP地址等信息。作为高级配置的应用配置对应的应用表主要保存系统检测的全部应用类型，包括应用的ID、协议ID、描述、监控状态等。协议表保存监控的协议ID、名称、描述、监控状态、以及对应的标准化协议的RFC详细说明。组配置里面存储的是用户自定义信息，例如应用组的表保存应用组的ID、描述、以及绑定的多个应用的ID等信息，接口组、IP组均与应用组的表结构类似。

# 第三章 系统方案设计

## 3.1 系统架构设计

目前软件体系结构主要有两种：客户机/服务器结构(C/S)和浏览器/服务器结构(B/S)。C/S结构主要是由一台服务器和大量的客户端组成，其中服务器负责数据的存放和检索，需要配置大容量存储器，以及存放数据的数据库系统。B/S结构是经过网络技术的不断发展，在C/S结构的基础上进行优化而诞生的。这种结构下，浏览器实现了客户操作界面，代替了之前的客户端，后台服务器主要负责代码流程逻辑实现。同样的功能在之前可能需要复杂的专业软件才能实现，现在的浏览器技术不断成熟，不使用复杂的软件也可以实现多种多样的功能，简化了开发流程，降低了开发成本。对于客户来说变得更易于操作，对于企业来说，由于大部分的逻辑实现在后台服务器，工程师维护也变得更容易，企业也可以节约很多人力物力。

通过对C/S结构以及B/S结构的分析可以看出，B/S结构相比于C/S结构更方便安装与操作，对于企业开发与维护也更节约成本。因此，本系统的体系结构选用B/S结构更加合适。整个网络流量监测系统大致分为客户层、表示层、业务逻辑层和数据持久层。

* 客户端即用户浏览器。
* 表示层用AngularJS进行开发，整合Ajax，Html等，将配置文件以及jar包导入工程，以进行表示层的开发。AngularJS是一个基于JavaScript的开源MVC框架，用于支持单页面动态应用程序的开发，其目的是为了简化开发过程以及测试的复杂度使程序开发变得更加灵活、方便，同时让程序的可扩展性大大提升。
* 业务逻辑层整合Spring框架，运用依赖注入的方法，创建服务层Service，然后进行业务逻辑处理。Spring框架是一个轻量级的面向切面和控制反转的容器框架，系统服务层采用Spring框架进行开发，采用最基本的Javabean来简化开发，解决企业应用开发的复杂性。
* 数据持久层直接使用sql语句根据不同的数据库类型运用不同的数据库工具（例如sqlsever数据库的bcp和oracle数据库的sqlloader）对数据库进行批量且高效的操作。

网络流量监测系统采用基于NetStream的流量采集方法对电商数据仓库间传输的原始数据信息进行定时监听和采集，采集的报文指标有目的IP地址、目的端口号、源IP地址、源端口号、协议号和TOS[14]，根据这些指标来区分流信息，根据不同的流信息进行独立的数据统计，而后形成文件并定时传输至NMC控制台。流量采集器通过采集和分析流量输出器发出的IP流量信息来实现全网监控。在流量输出器上进行配置后，流量输出器上的流量日志会发送到流量采集器NTC，由采集器对原始流量日志进行分析后，从接口、应用、IP组等多个维度进行数据汇聚后发送给NMC控制台。

NMC对采集器传输过来的原始数据进行初步统计和解析[15]，并按照不同维度进行1分钟、10分钟、1小时、6小时、1天、1周的时间粒度进行流量数据汇聚，实际业务根据前台需求通过对基础信息的计算与查询提供客户多方面的数据展示（例如：某段时间流量平均速率、最大速率、最小速率、数据总大小），再根据用户设定的阈值判断是否发送告警信息。NMC控制台采用基于widget框架的eview组件提供统一的前台图表，后台对不同业务类型进行数据信息的查询和计算，生成各自业务的表格数据，通过监控页面展示出各维度流量的百分比、大小、速率等基本信息，客户可以根据自己的需求对相应维度的监控项在配置页面进行配置。网络流量监测系统的架构如图3.1所示。

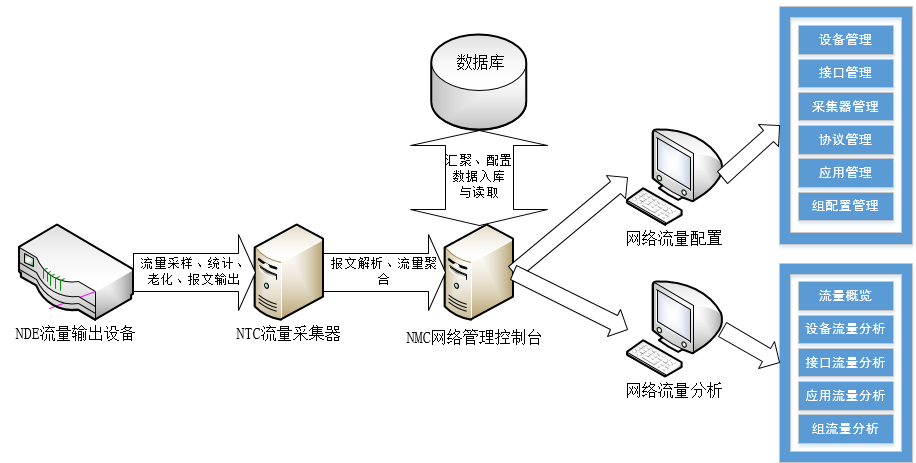


图3.1 网络流量监测系统架构图

## 3.2 网络流量采集功能设计

网络流量的采集始于网络流量设备对网络中流量的统计分析，包括流量的采样、统计、老化、输出四个步骤[16]，对于网络流量输出设备，为了满足客户需求，提供以下四种采样方式：

1. **固定报文间隔采样。**固定报文间隔采样是指每隔一定的报文数间隔对报文进行一次采样。在网络流量没有特殊特征的情况下一般使用此采样方式。
2. **随机报文间隔采样。**随机报文间隔采样是指在指定的报文间隔内随机对报文进行一次采样。此采样方式可以避免网络流量基于报文间隔数周期变化情况下，固定报文间隔采样导致统计失真的问题出现。
3. **固定时间间隔采样。**固定时间间隔采样是指每隔一定的时间间隔对报文进行一次采样。此采样方式用于在时间上均匀分布的流量。
4. **随机时间间隔采样。**随机时间间隔采样是指在指定的时间内随机对报文进行一次采样。此采样方式用于在时间上均匀分布的流量。可以避免网络流量基于时间周期变化情况下，固定时间间隔采样导致统计失真的问题出现。

为了减轻NTC或者NMC设备负荷，网络流量输出设备使用多种灵活的统计方式。不同的统计方式，以各自特征生成“流表”来记录“流”相关的数据。其中设备支持的统计方式如下：

**（1）原始流统计**

原始流统计是依据入/出设备接口索引、源IP地址、目的IP地址、源端口号、目的端口号、协议类型、TOS字段等七元组对报文分类的统计方法。原始流统计可以获取网络流量最详细的信息，除上述七元组外，还可以记录流的路由相关信息，例如源和目的自治域（as）、源和目的地址前缀长度掩码（prefix-mask）[17]、下一跳IP地址。

**（2）聚合流统计**

聚合流统计是在原始流统计的基础上，按照设定的规则将原始流统计信息进行合并的统计方法，支持的十种聚合规则如表3.1所示。

表3.1 聚合规则

|  |  |
| --- | --- |
| **聚合方式** | **聚合关键项** |
| 基于自治域聚合（as） | 输入设备接口索引、输出设备接口索引、目的AS号、源AS号 |
| 基于自治域-TOS字段聚合（as-tos） | TOS、字段输入设备接口索引、输出设备接口索引、目的AS号、源AS号 |
| 基于协议类型-端口号聚合（protocol-port） | 源端口号、目的端口号、协议号 |
| 基于协议类型-端口号-TOS字段聚合（protocol-port-tos） | 源端口号、目的端口号、协议号、TOS、输入设备接口索引、输出设备接口索引 |
| 基于源IP地址前缀聚合（source-prefix） | 源IP地址掩码长度、源AS号、源IP地址前缀、输入设备接口索引 |
| 基于源IP地址前缀-TOS字段聚合（source-prefix-tos） | TOS、源IP地址前缀、输入设备接口索引、源IP地址掩码长度、源AS号 |
| 基于目的IP地址前缀聚合（destination-prefix） | 目的IP地址掩码长度、目的AS号、目的IP地址前缀、输出设备接口索引 |
| 基于目的IP地址前缀-TOS字段聚合（destination-prefix-tos） | TOS、目的IP地址掩码长度、目的AS号、目的IP地址前缀、输出设备接口索引 |
| 基于源IP地址前缀和目的IP地址前缀聚合（prefix） | 目的AS号、源AS号、目的IP地址掩码长度、源IP地址掩码长度、目的IP地址前缀、源IP地址前缀、输入设备接口索引、输出设备接口索引 |
| 基于源IP地址前缀和目的IP地址前缀-TOS字段聚合（prefix-tos） | TOS、目的AS号、源AS号、目的IP地址掩码长度、源IP地址掩码长度、目的IP地址前缀、源IP地址前缀、输入设备接口索引、输出设备接口索引 |

例如，基于自治域聚合（as）的聚合关键项为“输入设备接口索引、输出设备接口索引、目的AS号、源AS号”当设备端接收到协议号或者源端口号、目的端口号不相同但是输入接口索引都为1，输出接口索引为2，目的AS号均为10，源AS号均为20的报文时，通过基于自治域聚合（as）的聚合，即使这些报文的协议号或者源端口号、目的端口号不相同，但是输入接口索引都为1，输出接口索引为2，目的AS号均为10，源AS号均为20的报文都认为是来自同一条流的报文，在聚合流表里合并为一条统计内容。

聚合流统计通过减少原始流统计报文的输出，使得统计的方式更加灵巧，并且达到降低NTC或者NMC的负担的目的。

**（3）灵活流统计**

灵活流统计是一种简化的原始流统计[18]方法。灵活流统计的报文分类元组可以根据用户自定义，用户可以从以下六个报文特征中自由选择报文分类统计的条件：

* 源IP地址（source-address）
* 目的IP地址（destination-address）
* 源端口号（source-port）
* 目的端口号（destination-port）
* 协议类型（protocol）
* TOS字段（tos）

通过灵活流统计可以在建立统计流表阶段聚合原始流，而不是在原始流统计输出后聚合，优点是可以减少NDE设备上流表的规格。同时也减少了NTC或者NMC设备的统计报文处理负担。

由于流量具有阵发性，NDE网络流量输出设备需提供老化机制[19]，具体老化机制如下：

* **定时老化。**定时老化是指一条流建立一段时候将其强制老化，如果老化后此流仍有数据传输，则建立新的流。定时老化又称为活跃老化，一般用于对统计结果实时性要求较强的场景下，例如计费或者网络应用实时性分析。
* **闲置老化。**闲置老化是指某条流最后一个报文到达后，在指定的时间内再没有报文到达后，将此流老化。闲置老化又称为非活跃老化，当定时老化被设置为较长时间时，通过设置较短的闲置老化时间可以快速老化已经“不存在”的流。
* **TCP FIN/RST触发老化。**FIN/RST触发老化主要用于快速老化TCP流表。对于TCP链接，当有标志为FIN或者RST的报文发送时，表示一次会话结束。因此当一条已经存在的TCP协议NetStream流中流过一条标志为FIN或者RST的报文时，可以立即把相应的NetStream流老化掉。
* **字节数计数溢出老化。**每个NetStream统计流可以记录最大4GB的报文字节计数，当报文字节计数超过4GB时，统计字段会因为溢出而从0开始记录，这将导致统计结果不准确。字节数计数溢出老化可以在统计数值达到定义最大值的情况下，将此条流表立即老化。
* **加速老化。**由于流具有突发性，短时间内大量的并发流会超出固定的流表规格，如果这部分流量的统计存在较多丢失，则严重影响统计结果的准确性。在此情况下，NDE通过加速老化，判断流表使用率达到设定规格时强制老化部分较早建立的流表，以保证突发的流量得以统计。

在网络流量设备最终阶段，统计报文从NDE输出到流量采集器NTC，不同格式的流量输出不同的报文，目前流量采集器支持以下两种统计记录报文格式：

* Cisco Systems NetFlow Services Export Version 5（简称V5）。该种报文格式相对固定，主要是根据七元组[20]来产生原始流的统计记录，优点是其输出的流统计记录字段非常全面，在NDE聚合之前就可以把全部的字段统计输出到采集器，但是由于其格式太过死板，不够灵活，这就给采集器设备带来了很大的压力，同样的，NMC控制台的压力也会同步上升，庞大的数据量限制了其适用场景为原始流的统计输出。
* Cisco Systems NetFlow Services Export Version 9（简称V9）。该报文格式相比V5更为灵活，报文的格式不再受限制，对于不同类型的流量用户可以自定义模板，有可扩展性，输出类型有两种，一种是统计数据，一种是选项数据。其可以用来输出聚合前的流记录，也可以输出聚合后的流记录。输出的数据量也较小，减轻了NTC采集器和NMC控制台的压力，适用于原始流、聚合流、灵活流的统计记录输出。

原始网流报文被网流采集器（NTC）接收之后，根据报文格式对报文进行解析，并生成对应模板的流，加入阻塞队列，流量监听服务对监控该队列，提取队列流从接口、应用、会话等维度进行原始报文1分钟粒度流量数据聚合处理，然后将聚合数据发送到网络管理控制台（NMC），进行多维度多级聚合处理。

## 3.3 网络流量分析功能设计

网络管理控制台（NMC）接收到流量采集器NTC的1分钟聚合数据后进行数据的入库，系统建立10分钟、1小时、6小时、1天、1周等粒度的聚合任务，聚合任务会定时将数据入库保存，不同维度的数据提供不同的存储策略，同时这些数据还可从数据库中转储成数据文件后永久保存下来。详细的汇聚规则如表3.2所示。

表3.2 详细汇聚规则

|  |  |  |  |
| --- | --- | --- | --- |
| **汇聚粒度** | **汇聚执行周期** | **汇聚基础数据** | **数据保存时间** |
| 1分钟数据 | 每分钟执行 | 原始数据 | 1天 |
| 10分钟数据 | 每10分钟执行 | 1分钟数据 | 10天 |
| 1小时数据 | 每小时执行 | 10分钟数据 | 1月 |
| 6小时数据 | 每6小时执行 | 1小时数据 | 3月 |
| 1天数据 | 每天执行 | 6小时数据 | 1年 |
| 1周数据 | 每周执行 | 1天数据 | 3年 |

网络管理控制台以图形化的方式提供多维度、多层次的流量分析报表。既可以直观显示全网接口、应用、主机等统计对象的Top N排行，又可以针对单个统计对象进行精细化分析，并且能准确查询到原始流量的具体信息，达到流量可视化的目的。

流量分析维度包含设备维度、接口维度、应用维度、主机维度、接口组维度、应用组维度、IP组维度等，每种维度都是先展现该统计维度下的Top N排行，选择一个具体统计对象后，再展现该统计对象的详细流量趋势及各相关流量数据的组成。

流量分析首页Dashboard流量概览界面从多维度显示Top N流量信息，包括Top N接口利用率、Top N设备流量、Top N应用流量、Top N接口流量等等，全网流量动态直观显示。并且用户可以定制操控版显示的维度信息来更方便的查看需要显示的流量信息。

如果用户需要查看某个维度更加具体的信息Dashboard流量概览菜单下分单个维度，进入单个维度的Top N流量排行，从全网角度展现该维度流量、数据包统计信息及流量趋势。单个维度的流量详细统计，包括流量、数据包双坐标趋势图及相关联流量信息排行，直观展现流量的组成。

对于Top N数据单条数据，进入单条数据信息界面，提供该条数据流量详细统计，在Top N数据的基础上增加了单条数据相对应的应用、接口、主机等等维度的流量组成，细化观察单条流量的组成。

系统还提供原始流量报表功能，展现最完整的流量数据信息，方便客户定位网络故障。

## 3.4 流量监测配置功能设计

控制台NMC通过图形化界面的方式给用户提供对设备及其监控信息的管理操作。网络流量监测配置分为三部分：基础配置、高级配置和组配置。其中，基础配置是实现对设备监控的第一步，基础配置提供采集器的管理、设备的添加、接口的监控以及采样比下发等操作；高级配置可以根据用户根据实际需要，进行更深层次的管理，比如协议配置以及应用管理以及流量阈值告警配置；组配置主要是为了方便用户将某些有关联的项进行分类，主要包括接口组、应用组、IP组。

### 3.4.1 基础配置

基础配置界面提供采集器的管理功能，对于采集器NTC与控制台NMC同机安装的情景下，安装之后默认添加本地采集器，添加的设备默认绑定到这个采集器，对于控制台NMC与采集器NTC分机安装的情况，则需要用户在安装采集器时绑定控制台的IP地址。

* 采集器配置界面用户可以查看采集器的状态和IP地址等信息，也可以设置采集接口的TOP N数目，系统默认为30个。同时打开采集器的流量取证开关，则采集器将会吧原始流文件上传到NMC控制台。
* 设备管理模块则会显示出采集器监控的全网上有流量上报的设备，用户可以选择性的对设备进行监控、删除、去监控等操作。
* 接口配置界面展示了向NMC控制台发送NetStream报文的设备接口，用户可对接口流入速率、流出速率和采样比进行配置，保证网流流量数据的正确性。

### 3.4.2 高级配置

* 协议配置：用户根据实际需要，可以选择性地对某些协议进行监控。
* 应用配置：列举了常用的543个网络应用，分为预定义应用和用户自定义应用。其中，预定义应用指的是预置的应用和设备识别并上报的应用；用户自定义应用指用户添加的应用，根据指定的协议(UDP/TCP)、端口范围和IP范围来定义应用。

### 3.4.3 组配置

* IP组配置：用户可将有关联的一组IP地址分为一组，如一个部门或一个楼层，方便查看该IP组的流量信息。
* 应用组配置：用户可将按照自己关注的点进行应用分类，如邮件类应用组，便于查看该应用组的流量信息。
* 接口组配置：用户可将相关联的接口定义成一个接口组，方便查看接口组流量信息。

## 3.5数据库表结构设计

本系统需要存入数据库的数据均在NMC端，其大致分为两种类型：一种存储接口、应用、IP组等维度的汇聚数据，另一种存储网络流量配置功能的设备、接口、采集器、应用、协议以及组配置等信息。

### 3.5.1 汇聚数据表结构

由于每个维度不同时间粒度的表结构相同，只是存储的数据不同，所以此处只列举出1分钟粒度的各个维度汇聚表结构。

**（1）应用维度**

应用维度的1分钟汇聚表名称为TBL\_FS\_APP1MIN，其表结构如表3.3所示。

表3.3 应用维度汇聚表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| ID | bigint (19) | 数据唯一编码，自增长，主键 |
| DEVICEID | integer (10) | 设备ID |
| INTERFACEID | integer (10) | 接口ID |
| FLOW\_DIRECTION | integer (1) | 流方向 |
| APPID | bigint (19) | 应用ID |
| BYTES | bigint (19) | 字节数 |
| PACKETS | bigint (19) | 包数 |
| FS\_TIME | bigint(19) | 流量准确发生时间 |

**（2）接口维度**

接口维度的1分钟汇聚表名称为TBL\_FS\_IF1MIN，其表结构如表3.4所示。

表3.4 接口维度汇聚表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| ID | bigint (19) | 数据唯一编码，自增长，主键 |
| DEVICEID | integer (10) | 设备ID |
| INTERFACEID | integer (10) | 接口ID |
| FLOW\_DIRECTION | integer (1) | 流方向 |
| AVG\_BYTES | bigint (19) | 平均字节数 |
| MIN\_BYTES | bigint (19) | 最小字节数 |
| MAX\_BYTES | bigint (19) | 最大字节数 |
| BYTES | bigint (19) | 字节数 |
| PACKETS | bigint (19) | 包数 |
| FS\_TIME | bigint(19) | 流量准确发生时间 |

**（3）主机维度**

主机维度的1分钟汇聚表名称为TBL\_FS\_HOST1MIN，其表结构如表3.5所示。

表3.5 主机维度汇聚表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| ID | bigint (19) | 数据唯一编码，自增长，主键 |
| DEVICEID | integer (10) | 设备ID |
| INTERFACEID | integer (10) | 接口ID |
| FLOW\_DIRECTION | integer (1) | 流方向 |
| AVG\_BYTES | bigint (19) | 平均字节数 |
| MIN\_BYTES | bigint (19) | 最小字节数 |
| MAX\_BYTES | bigint (19) | 最大字节数 |
| BYTES | bigint (19) | 字节数 |
| PACKETS | bigint (19) | 包数 |
| FS\_TIME | bigint(19) | 流量准确发生时间 |

对于接口组、应用组、IP组的汇聚数据，数据的存放还是分别在接口、应用、主机汇聚表内，只是在查询分析信息时取出每个组绑定的所有单个对象的数据进行汇总显示，这样可以节省汇聚表的数据量，减轻NMC的压力。

### 配置数据表结构

**（1）设备配置**

设备配置的数据库表名为TBL\_NAS\_DEVICE，其表结构如表3.6所示。

表3.6 设备配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| DEVICEID | integer (10) | 设备ID，自增长，主键 |
| DEVICE\_NAME | varchar(255) | 设备名称 |
| DEVICE\_IPADDR | varchar(255) | 设备IP地址 |
| DEVICE\_MACADDR | varchar(50) | 设备物理地址 |
| DEVICE\_DESCR | varchar(255) | 设备描述 |
| REFRESH\_TIME | bigint (19) | 刷新时间 |
| CREATE\_TIME | bigint (19) | 创建时间 |
| IS\_CONFIG | integer(1) | 配置标识符，默认0 |
| IS\_MANAGE | integer(1) | 是否监控，默认1 |
| SYSOID | varchar(255) | 系统对象标识 |
| DIS\_FLAG | integer(1) | 是否在界面显示，默认1 |

**（2）接口配置**

接口配置的数据库表名为TBL\_NAS\_INTERFACE，其表结构如表3.7所示。

表3.7 接口配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| INTERFACEID | integer (10) | 接口ID，自增长，主键 |
| DEVICEID | integer (10) | 设备ID |
| IF\_INDEX | integer(10) | 接口索引 |
| IF\_NAME | varchar(255) | 接口名称 |
| INTERFACE\_NAME | varchar(255) | 接口别名 |
| IF\_DESC | varchar(255) | 接口描述 |
| IN\_BAND\_WIDTH | bigint(19) | 流入带宽 |
| OUT\_BAND\_WIDTH | bigint(19) | 流出带宽 |
| UNMONITOR | integer(1) | 是否监控 |
| UNMONITOR\_STARTTIME | bigint (19) | 不监控的起始时间 |
| UNMONITOR\_ENDTIME | bigint(19) | 不监控的终止时间 |
| SAMPLE\_RATE | integer(10) | 接口采样比 |
| IS\_CONFIG | integer(1) | 配置标识符，默认0 |
| DIS\_FLAG | integer(1) | 是否显示标识，默认1 |

**（3）采集器配置**

采集器配置的数据库表名为TBL\_NAS\_COLLECTOR，其表结构如表3.8所示。

表3.8 采集器配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| COLLECTID | integer (10) | 采集器ID，自增长，主键 |
| NTC\_IP | varchar(64) | 采集器IP地址 |
| NTC\_NAME | varchar(50) | 采集器名称 |
| NTC\_DESC | varchar(255) | 采集器 |
| NTC\_PORT | varchar(50) | 采集器端口，默认9996 |
| BUFFER\_SIZE | integer(10) | 缓存大小，默认102400 |
| THREAD\_NUM | integer(10) | 线程数，默认8 |
| NTA\_SERVERIP | varchar(255) | NMC控制台IP地址 |
| NTA\_HTTPS\_PORT | bigint (19) | NMC控制台HTTPS端口，默认443 |
| FILE\_SEND\_PORT | integer(10) | 文件传输端口 |
| TIMEZONE | varchar(50) | 时区 |
| STARTUPTIME | bigint(19) | 启动时间 |
| COLLECT\_STATE | integer(1) | 采集器状态，默认1 |
| TRANSFER\_RAW\_FILE | integer(1) | 是否发送原始流文件，默认1 |

**（4）协议配置**

协议配置的数据库表名为TBL\_NAS\_PROTOCOL，其表结构如表3.9所示。

表3.9 协议配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| PROTOCOLID | integer (10) | 协议ID，主键 |
| PROTOCOL\_NAME | varchar(50) | 协议名称 |
| PROTOCOL\_DESC | varchar(255) | 协议描述 |
| MONITOR\_STATE | integer (1) | 监控状态 |
| REF\_RFC | varchar(50) | 标准化协议详细说明 |

**（5）应用配置**

应用配置的数据库表名为TBL\_NAS\_APPLICATION，其表结构如表3.10所示。

表3.10 应用配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| APPID | integer (10) | 应用ID，主键 |
| APP\_NAME | varchar(100) | 应用名称 |
| APP\_DESC | varchar(255) | 应用描述 |
| PROTOCOL\_ID | integer (10) | 协议ID |
| IS\_SYSDEF | integer (1) | 是否为系统预定义，默认1 |
| MONITOR\_STATE | integer (1) | 监控状态 |

**（6）应用组配置**

应用组配置的数据库由两张表组成，分别为TBL\_NAS\_APPGRP和TBL\_NAS\_APPGRPMAP。其中TBL\_NAS\_APPGRP表结构如表3.11所示，TBL\_NAS\_APPGRPMAP表结构如表3.12所示。

表3.11 TBL\_NAS\_APPGRP配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| APPGRPID | integer (10) | 应用组ID，自增长，主键 |
| APPGRPNAME | varchar(100) | 应用组名称 |
| APPGRPDESC | varchar(255) | 应用组描述 |

表3.12 TBL\_NAS\_APPGRPMAP配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| APPGRPMAPID | integer (10) | 应用组MAP ID，自增长，主键 |
| APPGRPID | integer (10) | 应用组ID |
| APPID | integer (10) | 应用ID |

**（7）接口组配置**

接口组配置的数据库由两张表组成，分别为TBL\_NAS\_IFGROUP和TBL\_NAS\_IFGRPMAP。其中TBL\_NAS\_IFGROUP表结构如表3.13所示，TBL\_NAS\_IFGRPMAP表结构如表3.14所示。

表3.13 TBL\_NAS\_IFGROUP配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| IFGROUPID | integer (10) | 接口组ID，自增长，主键 |
| IFGRPNAME | varchar(100) | 接口组名称 |
| IFGRPDESC | varchar(255) | 接口组描述 |

表3.14 TBL\_NAS\_IFGRPMAP配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| ID | integer (10) | 接口组MAP ID，自增长，主键 |
| IFGROUPID | integer (10) | 接口组ID |
| INTERFACEID | integer (10) | 接口ID |

**（8）IP组配置**

IP组配置的数据库由两张表组成，分别为TBL\_NAS\_IPGROUP和TBL\_NAS\_IPGROUPRANGE。其中TBL\_NAS\_IPGROUP表结构如表3.15所示，TBL\_NAS\_IPGROUPRANGE表结构如表3.16所示。

表3.15 TBL\_NAS\_IPGROUP配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| IPGROUPID | integer (10) | 接口组ID，自增长，主键 |
| GROUPNAME | varchar(100) | IP组名称 |
| GROUPDESC | varchar(255) | IP组描述 |
| IPGROUPSTATE | integer(1) | IP组监控状态 |
| IPBPS | bigint(19) | 每秒比特率 |

表3.16 TBL\_NAS\_IPGROUPRANGE配置表结构

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **含义** |
| IPRANGEID | integer (10) | IP组范围ID，自增长，主键 |
| IPGROUPID | integer (10) | IP组ID |
| RANGETYPE | integer (1) | 网络类型 |
| STARTIP\_ADDR | bigint(19) | 起始IP |
| ENDIP\_ADDR | bigint(19) | 结束IP地址 |
| SUBNET\_MASK | bigint(19) | 子网掩码 |
| NETWORK\_MASK | integer(10) | 掩码位 |

# 第四章 系统实现与测试

## 4.1 网络流量采集功能的实现

网络流量采集的主要功能是负责分析NDE输出数据包，并对对输出的数据进行适配解析，并对解析好的数据进行聚合、过滤和封装然后发送到网络管理控制台NMC进行分析处理。其中NDE设备对流量的采样、统计、老化、输出均在设备上实现，NTC实现对NDE设备输出的报文解析以及多维度聚合功能。

网络流量采集器作为一个应用部署在网络流量监测系统中，有集中式部署和分布式部署两种方式，可以全量安装网络管理控制台NMC和网络流量采集器NTC，也可以只安装NTC，分布式安装采集器时需要绑定可以正常使用的网络管理控制台的IP地址，否则系统无法安装。

### 4.1.1 网络流量采集流程

采集器启动后，对各个子服务进行初始化，首先是对接受服务的初始化：

this.receiverService = new ReceiverService();

接受服务的初始化主要是建立流量包空队列flowPacketQueue、创建流的适配器flowAdapterMgr以及初始化流量监听服务flowListeners，其中流量适配器主要区分NetStream流统计的版本格式。之后将初始化好的接受服务赋值给聚合服务进行初始化：

this.aggregationService = new AggregationService(this.receiverService);

然后判断NTC绑定的网络管理控制台NMC部分是否处于运行状态，如果控制台部分没有运行，则采集器启动终止。若NMC正在运行则进行各维度缓存模型的下载初始化以及接受聚合服务的启动。代码架构如下：

private static void initCache(AggregationService aggregationService, ReceiverService receiverService)

{

CollectorCache.getInstance().initListenPorts();

DeviceCache.getInstance().loadDevice();

InterfaceCache.getInstance().loadInterface();

ApplicationCache.getInstance().loadApplication();

ProtocolCache.getInstance().loadProtocol();

IpgroupCache.getInstance().loadIpGroup();

IpGroupToIpGroupCache.getInstance().loadIpgroupToIpgroup();

aggregationService.startService();

receiverService.startService();

}

### 4.1.2 网络流量接收服务

在聚合服务启动之初，先创建接收服务的流量监听器，其中每一种维度的流量类型作为一种HandlerType，每一种类型有自己独立的流量监听函数，具体实现如下：

private List<IFlowListener> createListeners()

{

List<IFlowListener> flowListeners = new LinkedList();

FlowListener listener = new FlowListener(HandlerType.RawFlow);//原始流类型

flowListeners.add(listener);

FlowListener intlistener = new FlowListener(HandlerType.Interface);//接口类型

flowListeners.add(intlistener);

if (Configuration.getInstance().isAppAggr())

{ FlowListener applistener = new FlowListener(HandlerType.Application);//应用类型

flowListeners.add(applistener); }

this.aggrFlowListeners.addAll(flowListeners);

return flowListeners;

}

**（1）报文版本格式解读**

接收服务启动时，首先检测监听端口是否有效，无效则终止服务，并抛出异常，记录日志信息。如果有效，则启动UDP报文解析线程，接收到的UDP报文通过buff的字节头获取该流的版本号，实现中的parseDatagramPacket函数将根据该版本号进行不同版本的解析，解析返回不同版本的数据包，再将该数据包加入到队列中等待下一步的处理，该线程从当前步骤直到采集器进程结束一直会运行，具体解析实现如下：

ByteBuffer buf = (ByteBuffer)key.attachment();

buf.clear();

SocketAddress inetsa = ((DatagramChannel)key.channel()).receive(buf);

if (inetsa != null) {

if (ReceiveSwitcher.isOpen())

{ buf.flip();

int bufsize = buf.limit();

byte[] bytarr = new byte[bufsize];

buf.get(bytarr);

DatagramPacket p = new DatagramPacket(bytarr, bufsize, inetsa);

this.counter += 1L;

//检测报文版本

byte[] buff = p.getData();

short version = (short)(int)Util.byteArray2Long(buff, 1, 1);

switch (version)

{ case 5:

packet = this.flowAdapterMgr.parseDatagramPacket(p, "netflow\_v5");

break;

case 9:

packet = this.flowAdapterMgr.parseDatagramPacket(p, "netflow\_v9");

break; }

if (packet != null)

{ this.flowPacketQueue.offer(packet);

synchronized (this.flowPacketQueue)

{ this.flowPacketQueue.notifyAll(); } }

}

}

**（2）流量包解码**

在流量包的报文格式确定之后，接下来就是根据报文格式将数据包转换成对应版本的流，此线程与报文解析线程一样，从接受服务运行开始到采集器结束都会运行。其中的process函数将流量包转换成对应报文格式版本的流。主要实现如下：

IFlowPacket packet = (IFlowPacket)this.flowPacketQueue.poll();

List<IFlow> flowList = packet.process();//将流量包转换成对应版本的流

boolean flowlistvalid = (flowList != null) && (!flowList.isEmpty());

if (flowlistvalid)

{ for (IFlowListener listener : this.listeners) {

listener.handleFlow(packet.getRouterIP(), flowList);//将转换好的流加入阻塞队列等待聚合}

}

在每个维度的监听器的handleFlow 函数内主要讲转换好的流列表加入到监听器的阻塞队列中等待各个监听器的聚合处理：

this.flowBlockingQueue.offer(new FlowListenerObject(ip, flowList));

### 4.1.3 网络流量聚合服务

聚合服务启动前，先创建各个维度的流量监听线程，并将它们加到接收服务与聚合服务的流量监听列表中，然后将流量监听线程列表的各个线程启动，聚合服务的流量监听线程从聚合服务开启到采集器结束运行之间一直运行。

每个维度分为三层：处理层（Handler），负责对每条流进行提取、分割；持久层（Holder），进行同种流的聚合处理，以及输出统计数据的字符串拼接；导出层（Exporter），将处理层的保存的流量缓存写入文件并将此文件加入压缩文件map中等待压缩线程对文件进行打包压缩。

**（1）处理层（Handler）**

其间监听线程接收服务处理完成的流阻塞队列，根据根据每个维度的handlerType进行不同的聚合处理，具体实现如下：

AbstractHandler handler = null;

FlowListenerObject listenerObject = this.flowBlockingQueue.take();

if ((listenerObject.flowList != null)

&& (!listenerObject.flowList.isEmpty()))

{ if (this.handlerType != null) {

handler = HandlerType.getHandler(

listenerObject.ip, this.handlerType);

} else if (this.extHandlerName != null) {

handler = HandlerType.getHandler(

listenerObject.ip, this.extHandlerName); }

if (handler != null) {

handler.process(listenerObject.flowList);//处理流进程 }

}

每个维度handler的process函数处理机制类似，以下以应用维度的process函数实现举例，process函数的主要功能是将流列表中每条流提取并拷贝自身并将这两条流按照出入方向进行分割，此举主要是方便控制台NMC处理在带接口特殊方向条件下的复制出来的流，复制出来的流出方向字段值为3，对于应用统计维度，如果分析条件没有接口条件，则需要去除方向为3的流量；如果有接口条件，则需要带上方向为3的流量，保证一个接口下统计的流量是准确的，否则会导致应用维度流量翻倍。其具体实现如下：

public void process(List<IFlow> flowList)

{ if (flowList == null) {

return; }

synchronized (this.lockObj)

{ for (IFlow flow : flowList)

{ List<IFlow> splitedFlows = flow.split(true);//对流的复制分割

if (splitedFlows != null) {

for (IFlow splitFlow : splitedFlows)

{ Map<Object, AbstractHolder> cache = getCache(splitFlow);

addToCache(splitFlow, cache);//进行流的聚合处理

}

}

}

}

}

//如果缓存中没有这条流，则直接将该流添加到缓存里，如果存在该流，则进行聚合处理

private void addToCache(IFlow flow, Map<Object, AbstractHolder> cache)

{

int key = flow.getAppID();

if (cache.containsKey(Integer.valueOf(key)))

{

AbstractHolder origin = (AbstractHolder)cache.get(Integer.valueOf(key));

origin.aggregate(flow);

}

else

{

cache.put(Integer.valueOf(key), new AppTrafficHolder(flow));

this.appFlowCount += 1;

}

}

**（2）持久层（Holder）**

持久层对应的处理层的aggregate具体操作是：

this.totalDOctets += flow.getdOctets();

this.totalDPkts += flow.getdPkts();

主要作用是对同种流的总字节数和总报数进行聚合。而输出文件字符串拼接函数getAggregateFlowString实现如下：

public String getAggregateFlowString(long flowSummaryTime)

{

StringBuffer appSB = new StringBuffer();

appSB.append(this.flow.getDeviceID());

appSB.append("|");

appSB.append(this.flow.getInterfaceID());

appSB.append("|");

appSB.append(flowSummaryTime);

appSB.append("|");

appSB.append(this.flow.getAppID());

appSB.append("|");

appSB.append(this.flow.getDirection());

appSB.append("|");

appSB.append(this.totalDOctets);

appSB.append("|");

appSB.append(this.totalDPkts);

return appSB.toString();

}

**（3）导出层（Exporter）**

导出层的主要功能是将处理层保存的流量缓存写入文件，并将此文件加入压缩线程的压缩文件map中等待压缩线程对文件进行打包压缩。具体实现如下：

List<AbstractHolder> appFlowList = getFlowList();

if (appFlowList == null) {

return; }

this.flowCount = appFlowList.size();

File saveFile = ExportUtil.createAppFile(this.routerIP, this.dumpTime, "APP");

ExportUtil.writeFile(appFlowList, saveFile, false, this.dumpTime);//写入文件

ZipThread.setZipFileListMap("APP", saveFile.getCanonicalPath());//把文件加入压缩map

采集器启动时会开启一个基于SFTP的文件传输Runnable去定时的将采集器NTC的聚合统计数据文件传输到控制台NMC端。

## 4.2 网络流量分析功能的实现

网络流量分析功能提供全网流量概览，以图形化的方式提供多维度多层次的流量分析报表，实时展现全网流量动态，用户可快速查看到各维度的流量信息。既可以直观显示全网接口、应用、主机等统计对象的Top N排行，又可以针对单个统计对象进行精细化分析，并且能准确查询到原始流量的具体信息。

控制台NMC启动后会启动一个DumpingThread线程，该线程每隔三十秒扫描一次汇聚数据文件夹，对新发送来的压缩文件进行解压入库，逻辑实现大体如下：

File pathFile = new File(this.path);//文件路径

File[] files = null;

synchronized (lockObjMap.get(this.name))

{ //取出文件夹内的文件列表

files = pathFile.listFiles(new DumpFileFilter(this.filter));

Arrays.sort(files, new FileModifyTimeComparator());

IFileCollector collector = new FileCollectorImpl();

IFileDumper dumper = new FileDumperImpl();

//解压并将数据入库

unzipAndDump(files, collector, dumper);

}

每个维度分别有1MIN、10MIN、1HOUR、6HOUR、1DAY、1WEEK六张汇聚表用来存储不同时间内的基础数据。其中，1MIN时间粒度的是由NTC流量采集器发送过来入库，剩下五种汇聚表均在上一种汇聚时间维度进行更长时间的汇聚，控制台会启动一个定时任务AggrTask定时进行聚合，由于不同维度的聚合配置类似，以下使用应用维度的聚合配置举例，其聚合属性配置在xml文件中，具体为下：

<!--具体聚合类型-->

<aggregation tableprefix="TBL\_FS\_APP" from="1MIN" to="10MIN" >

<attr name="tominunit" value="600000"/>

<attr name="from" value="1MIN"/>

<attr name="to" value="10MIN"/>

<attr name="base" value="0"/>

</aggregation>

<aggregation tableprefix="TBL\_FS\_APP" from="10MIN" to="1HOUR" >

<attr name="tominunit" value="3600000"/>

<attr name="from" value="10MIN"/>

<attr name="to" value="1HOUR"/>

<attr name="base" value="0"/>

</aggregation>

<aggregation tableprefix="TBL\_FS\_APP" from="1HOUR" to="6HOUR" >

<attr name="tominunit" value="21600000"/>

<attr name="from" value="1HOUR"/>

<attr name="to" value="6HOUR"/>

<attr name="base" value="3600000"/>

</aggregation>

<aggregation tableprefix="TBL\_FS\_APP" from="6HOUR" to="1DAY" >

<attr name="tominunit" value="86400000"/>

<attr name="from" value="6HOUR"/>

<attr name="to" value="1DAY"/>

<attr name="base" value="3600000"/>

</aggregation>

其对应的聚合SQL也配置在xml文件中，如图4.1所示。

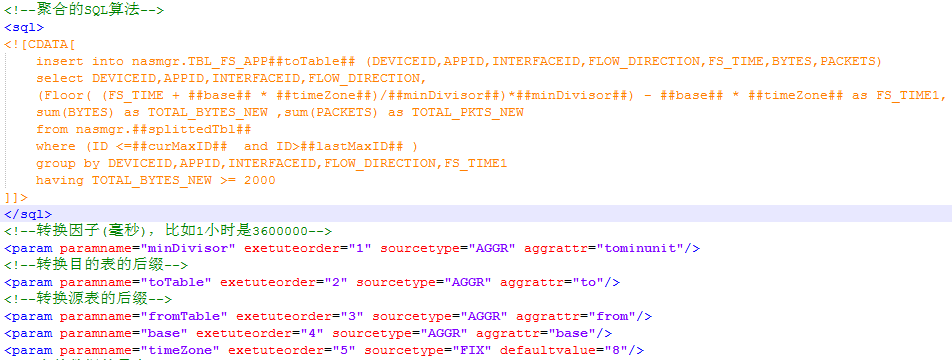


图4.1 聚合SQL算法及其SQL参数对应的聚合属性

网络流量分析界面图形统一使用一套widget框架模板，由于每个界面使用固定的画图控件固定，所以为了方便数据的查询与存储，在控制台NMC安装的时候则对每个控件初始化好固定的widgetid，根据前台参数生成查询条件filter，根据查询条件通过调用后台ds层的restful服务接口根据widgetid调用对应的namingsql获取图表数据并返回前台显示。以一种控件的查询函数举例，代码逻辑实现大致如下：

protected WidgetData getTotalData(List<WidgetData> topnData, Map<String, Object> conMap)

{

//用户设置的查询条件

List<WidgetCustomValue> customValues =

this.widgetObj.getChartDataType().getCumstomValue();

WidgetData resultData = null;

//根据widgetinfo生成的查询map调用sql查询数据

Map<String, Object> totalData =

WidgetDAO.getInstance().queryTotal(this.widgetSql.getTotalSql().getSql(),

conMap, customValues);

resultData = this.dataProcessor.constructTotalData(totalData, this.widgetObj);

return resultData;

}

其对应的后台sql配置在xml文件中，如图4.2所示。

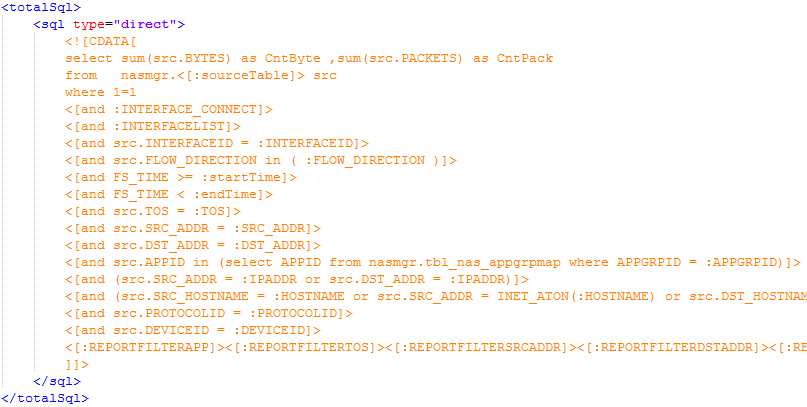


图4.2 查询全部数据sql

前台使用公司内部eview组件进行画图，图4.3是当chartType为35即折线图时的画图组件定义结构：

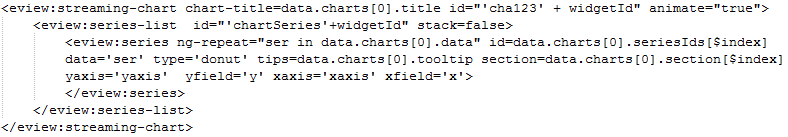


图4.3折线图控件定义结构

网络流量分析界面包括三种画图控件，分别是圆饼图、折线图、区域图。圆饼图展现的是全部流量的类型百分比，应用维度饼图如图4.4所示，展示了全部应用流量各种协议所占比例，图中显示了显示了TOP 5的流量排行，TOP之外的都会被分类到other里面，用户可以自定义需要显示的TOP N个数来查看更多比例分配。



图4.4 应用维度饼图

折线图展现出某个对象的流量趋势图，图4.5显示了某个接口近十五分钟的流量趋势，下方可以提供此段时间内的最大、最小、平均速率以及流量信息的查看。用户可以自定义时间段方便查看更多时间段内的流量趋势状况。

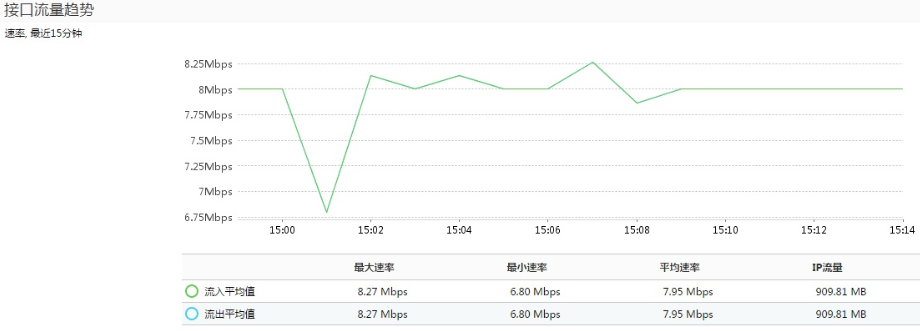


图4.5 接口流量趋势图

区域图显示出某个对象在某段时间内的TOP N流量组成，图4.6显示了某主机在某时间段内的流量组成，默认显示TOP 5，同样支持自定义TOP N显示数目。

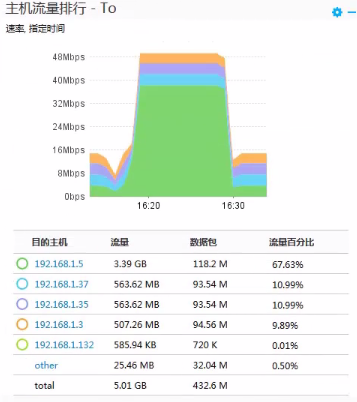


图4.6 主机流量区域图

这三种组件根据不同场景进行组合，来展现多维度多层次的网络流量分析。在网络流量配置模块添加了设备以及网络中可以监控到流的前提下，网络流量分析的流量概览Dashboard界面如图4.7所示。该界面显示了全网流量的大致分布，包括各种维度的流量比例组成，其中接口维度还提供速率和利用率的概览，但是不提供具体维度的详细流量组成信息，当需要查看某个维度的具体流量信息时则需要点击某个维度的分析界面，接口维度相比其他维度多一个接口分析列表界面，缺少对应维度的TOP N流量趋势界面，而对于其他界面框架基本一致，只是显示的文字和数据不同，所以以下以接口维度的具体流量分析界面以及其他维度的TOP N流量趋势界面相结合进行展示。

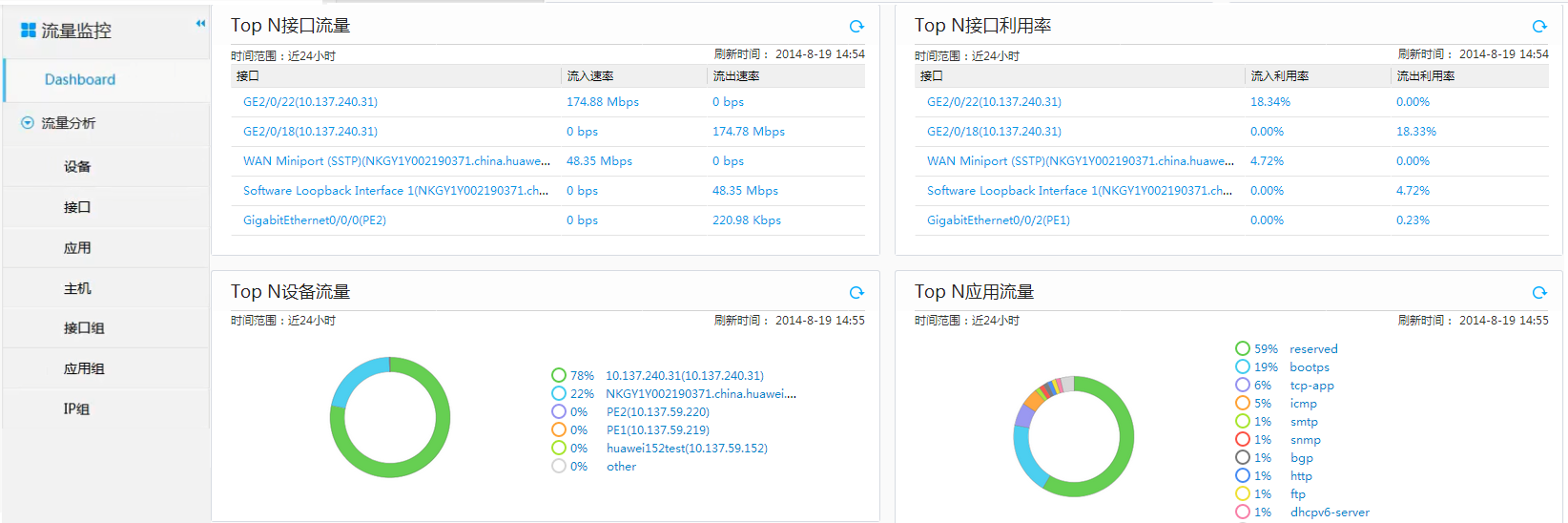


图4.7 流量概览界面

对于除接口维度意外的其他维度，点击对应维度菜单进入TOP N流量趋势界面，以应用维度的TOP N流量趋势界面进行展示的效果如图4.8所示。

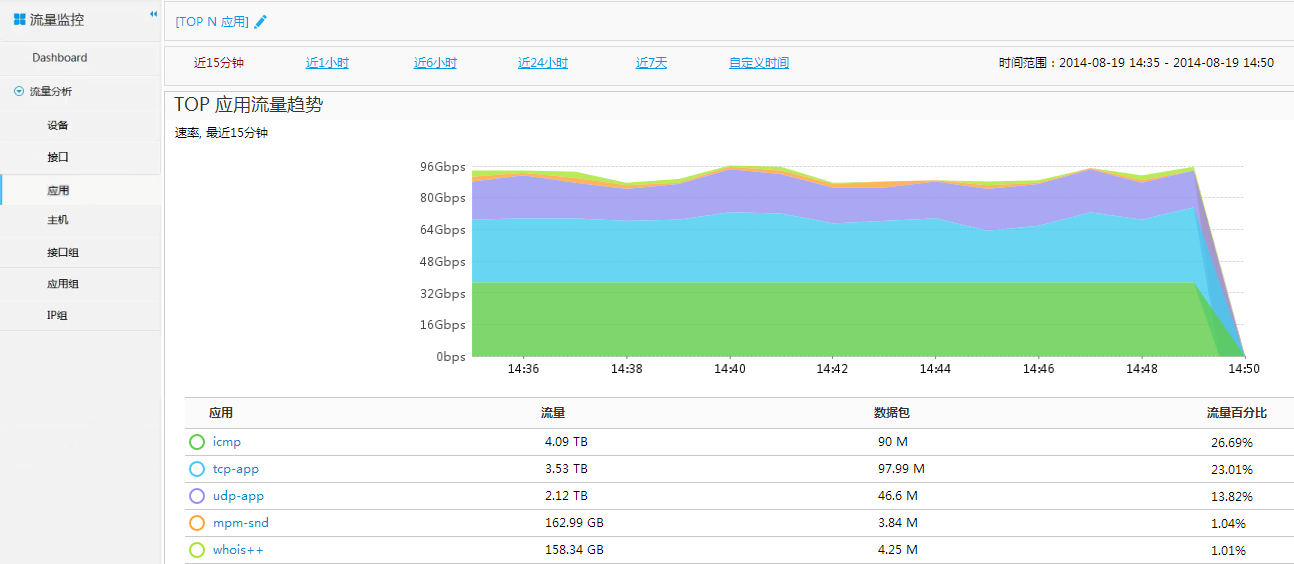


图4.8 应用维度TOP N流量趋势界面

而点击接口维度后，进入的是接口分析列表界面，而不是TOP N流量趋势界面。此界面提供全部接口的接口名称、别名流入或者流出利用率以及该接口对应的设备等信息，如图4.9所示。此界面提供对全部接口的查询，可以根据接口名称、接口别名、设备名称等条件来搜索来找到想要查看的接口，并点击接口链接则进入单个接口的具体流量分析界面，如图4.10所示。

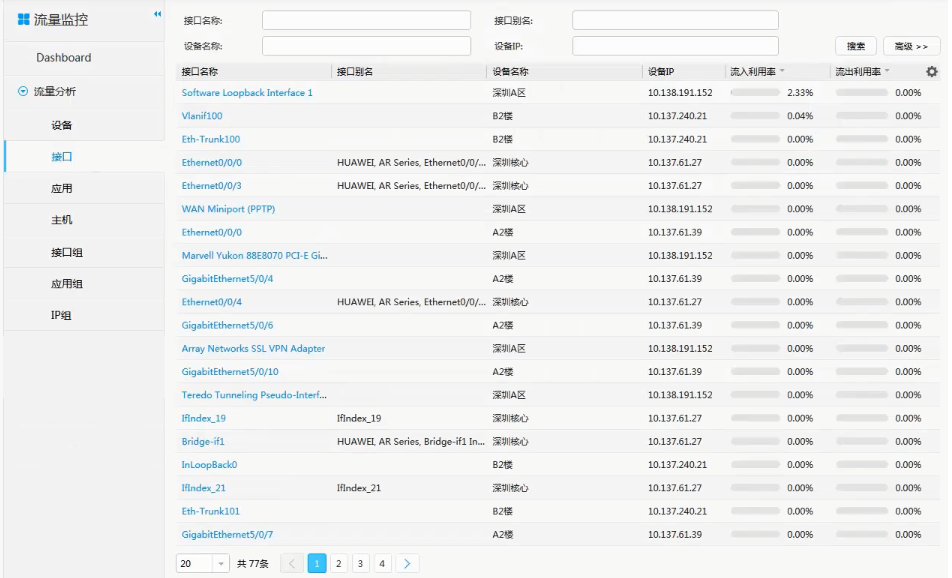
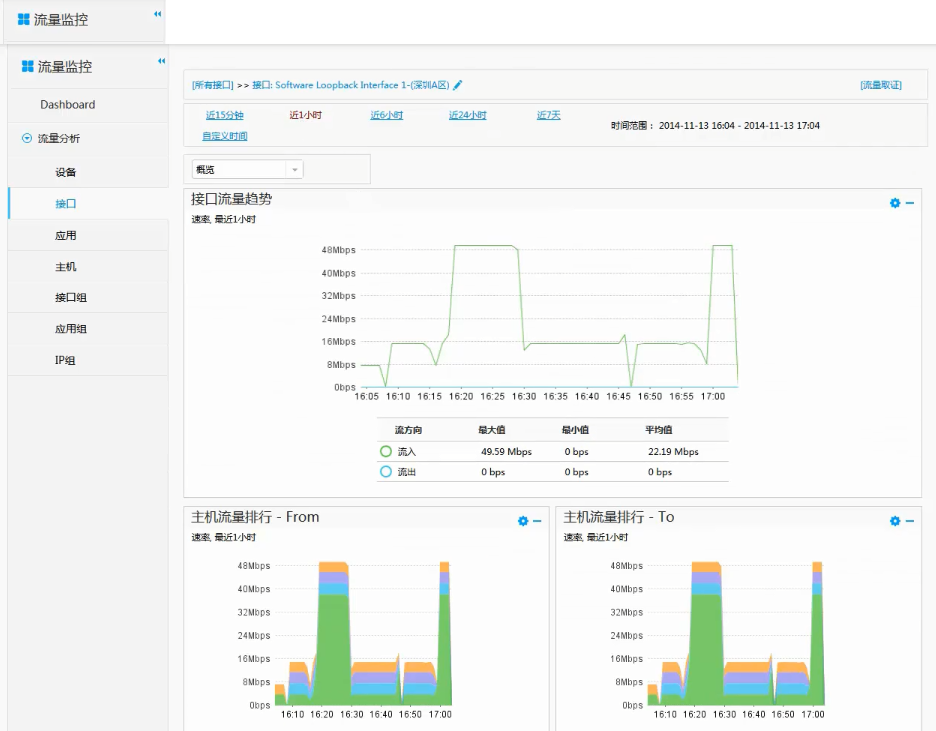


图4.9 接口分析列表界面



4.10 单个接口流量分析界面

## 4.3 流量监测配置功能的实现

控制台NMC通过图形化界面的方式给用户提供对设备及其监控信息的管理操作。网络流量监测配置功能的前台表示层使用AngularJS框架进行开发，AngularJS是一款非常优秀的前端JavaScript框架，其支持单页面的动态开发，使开发过程更为简单以及程序测试的成本，提供更为灵活的开发方式。公司在AngularJS基础上开发了一套内部eview组件，该套组件在AngularJS原生控件的基础上扩展了组件的更多功能，并优化了组件的美观度。为了满足公司要求，保证系统前台界面风格一致以提升美观度及易用性，表示层统一使用eview控件进行开发，以下是一般界面都会使用的按钮组确定和取消：

<div ng-controller="ButtonGroupController" id="buttongroupController">

<eview:button-group id="buttonGroup">

<eview:button id="confirm" click="confirm" text="{{i18n.confirm}}">

</eview:button>

<eview:button id="cancle" click="cancel" text="{{i18n.cancel}}">

</eview:button>

</eview:button-group>

</div>

这个按钮组拥有一个ID唯一的ng-controller，按钮的所有事件均实现在此controller内。使用AngularJS的另一好处是数据双向绑定，{{i18n.confirm}}和JS文件中引用的按钮国际化资源进行双向绑定，如要修改按钮文字，则只需修改国际化资源里的内容，不需要修改前台html代码即可，简化了表示层的开发。

前台请求框架使用RESTful架构，此架构使代码逻辑更为简洁，代码层次感更强，RESTful样式的服务都有一个独立的url，通过此url调用后台对应的服务。包括GET、POST、PUT、DELETE四种请求。

网络流量配置的三种配置里面的菜单代码架构基本类似，故仅挑选一个维度的配置架构进行介绍，以下为应用维度的各个url配置：

var urls=

{

appQuery:"/rest/enterprise/app/nas/manager/appcfg/quaryApp",//查询应用

appCreate:"/rest/enterprise/app/nas/manager/appcfg/createApp",//创建应用

appModify:"/rest/enterprise/app/nas/manager/appcfg/modifyApp",//修改应用

appDelete:"/rest/enterprise/app/nas/manager/appcfg/deleteApp"//删除应用

}

应用查询的RESTful的GET请求的基本格式为：

var urlVal=constant.urls.appQuery;

iemp.external.get({

url:urlVal,

dataType:"json",

data:{param:JSON.stringify(condition)},//查询条件

success:function(data)

{

initPageData(data);

},

error:function(jqXHR)

{

}

})

后台请求控制部分通过OSGI服务对前台请求进行监听：

<osgi.activator>

enterprise.app.nas.manager.as.communication.BaseinfoBundleActivator

</osgi.activator>

注册好RESTful服务将应用的查询请求映射到以下方法：

@Path({"enterprise/app/nas/manager/appcfg"})

@Target("appcfg")

@ROAEncode(true)

public class AppROAService

{ @GET

@Path({"quaryApp"})

public ROAResult<AppQueryBackDataModel> doGet(@QueryParam("param")

String param)

{ //查询条件

AppQueryDataModel condition = new AppQueryDataModel();

condition = JSONUtil.strToObj(param, AppQueryDataModel.class);

//进入实现层进行应用查询

AppQueryBackDataModel app = queryAppCommon(condition, true);

//返回结果

ROAResult<AppQueryBackDataModel> result = new ROAResult();

result.setData(app);

return result; }

}

业务DAO层实现通过Spring，采用最基本的Javabean调用请求对应的具体业务实现。以应用查询请求举例，根据请求的condition生成filterMap进行数据库查询操作，该方法对应的实现如下：

public List<ApplicationModel> queryApplication(ApplicationFilter filter,

int pageBegin, int pageLength)

{ ApplicationModel model = ApplicationFilter.convert2AppModel(filter);

String nameSqlID = "queryall.nasmgr.tbl\_nas\_application";

//转换查询条件

Map<String, Object> filterMap = new HashMap();

AppBuildParam.addFilterMap(filterMap, model);

DataSet dataSet = null;

//数据库查询

dataSet = DataBaseCommonEnt.excuteQuery(nameSqlID,

filterMap, pageBegin, pageLength);

List<ApplicationModel> models = new ArrayList();

ApplicationModel tmpModel = null;

while (dataSet.next())

{ tmpModel = (ApplicationModel)DataObjectMapper.dataMapObject(

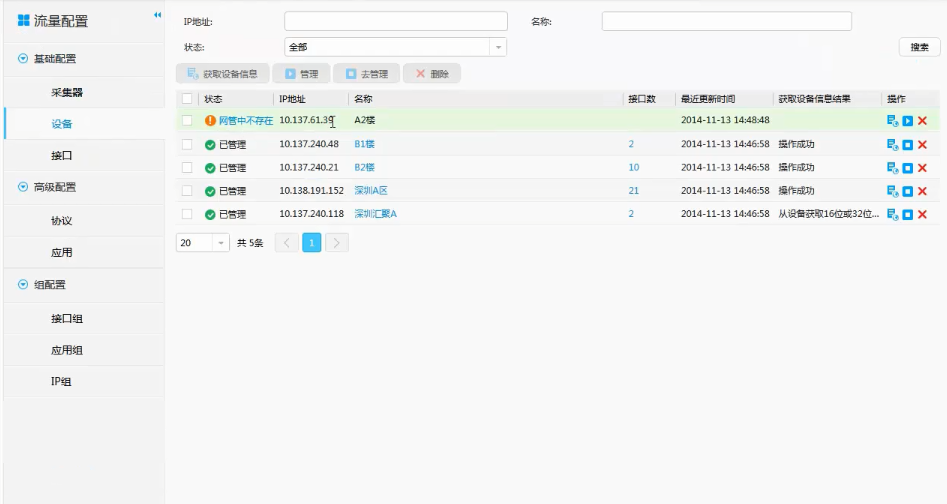
dataSet, ApplicationModel.class);

models.add(tmpModel); }

return models;

}

网络流量监测的第一步就是在系统中添加设备，并且有网络流量流经该设备，采集器采集到该设备的流量后会发送消息给NMC控制台，将该条设备存入数据库，但是该条设备并不会自动管理，需要在设备配置界面点击该条设备将其加入到系统中进行管理，此时才会发送消息通知采集器进行该条设备的采集操作。其中设备的配置界面如图4.11所示。



4.11设备配置界面

当设备添加到网络流量监测系统里后，将自动将该设备的接口入库，并显示在接口配置界面，此时设备会按照默认的1000比一的采样比进行报文的统计输出，用户可以通过此界面对接口的采样比进行修改，也可以对接口进行监控或者去监控、删除等操作，其中接口配置的界面如图4.12所示，修改接口采样比的界面如图4.13所示。

对于组配置，接口组、应用组、IP组的界面框架类似并且流程相同，三种组配置均提供对应组的搜索、创建、修改、删除等操作，以下使用IP组的配置进行展示，其中IP组配置主界面如图4.14所示。

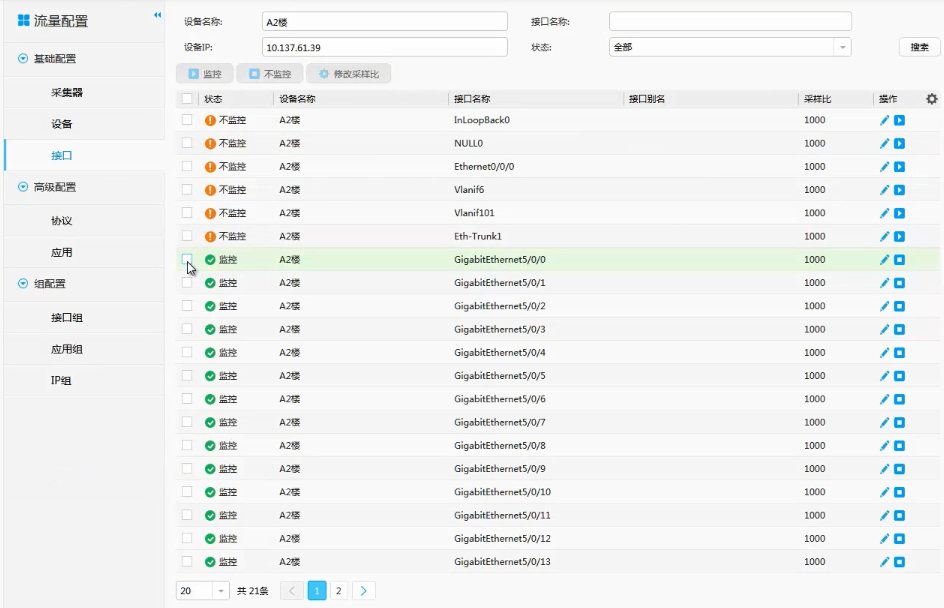


图4.12 接口配置界面

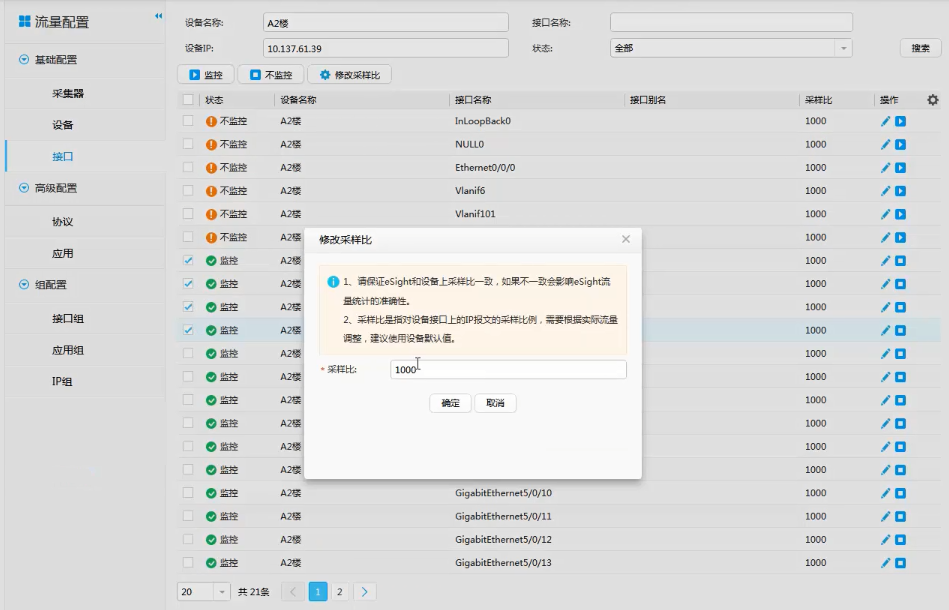


图4.13 接口配置修改采样比界面

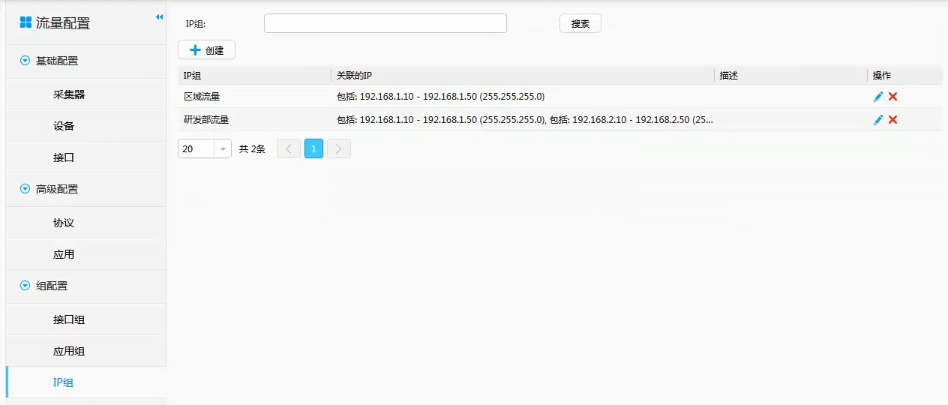


图4.14 IP组配置界面

用户在配置界面可以将某一区域或者部门内的IP地址定义为一个IP组，创建完成后，网络流量分析模块将会开始对该IP组进行流量数据的分析，其分析效果如图4.15所示。



图4.15 IP组流量分析界面

## 4.4 系统测试

### 4.4.1 测试目的

网络流量监测系统的功能测试要求保证用户按照正确要求搭建好网络流量监测系统之后，配置模块能够正常的对设备及其接口、以及其他功能进行配置，分析模块能够在要求的时间内做出响应并按照实际的用户配置显示正确的监测结果。同时在满足基本功能正常的情况下，对比较极端的情况下，比如服务器CPU持续冲高、内存不足的情况下，要求系统能够进行正常工作，并且保证流量不失真。

### 4.4.2 测试环境

不同的硬件平台对管理能力有不可避免地有所影响。测试选用比全量组件安装的最低配置要求稍高的PC服务器。其中分为两种场景：

**（1）同机部署**

同机即采集器NTC与控制台NMC安装在同一台服务器，其配置要求为CPU 2\*四核2GHz以上，内存8GB，硬盘空间120GB，数据库为MySQL 5.6，操作系统为windows 7，选用PC Sever。

**（2）分机部署**

分机部署为采集器独立安装在与控制台相独立的服务器，不需要安装数据库，其配置要求为CPU 1\*四核2GHz以上，内存4GB，硬盘空间120GB，操作系统为windows 7，选用PC Server。

测试工具选用发流工具，此工具为公司内部开发，可以任意配置发流的IP地址以及端口。高级选项可以配置会话信息，包括会话源IP地址与目的IP地址、流入接口ID、流出接口ID，也可以发送自定义应用类型的流。

### 4.4.3 测试过程及结果

系统功能测试过程如下：

* 解压并安装网络流量监测系统的全量安装包，并填写数据库的基本信息。
* 安装完毕后打开并登陆系统，此时打开测试工具，填写此NMC控制台的大网IP地址，默认的监听端口是9995与9996。
* 在高级配置里面创建会话信息点击发送，等待三分钟采集器采集到数据后，进入网络流量配置基础配置界面，发现新增了一台新发现的但是NMC控制台中并不存在的设备，此状态的设备数据并不会进行汇聚操作。
* 点击该设备，将此设备添加到NMC控制台中，此时采集器开始进行数据聚合。
* 等待三分钟查看网络流量分析界面，查看各维度是否有数据，显示的流量是否与配置工具配置相同。

具体的测试过程和测试结果如表4.1所示，该表的测试结果显示系统的基本功能正常可用。

表4.1 测试过程与测试结果

|  |  |  |  |
| --- | --- | --- | --- |
| **功能模块** | **测试内容** | **前置条件** | **测试结果** |
| 网络流量配置 | 添加设备 | 系统正确安装，对采集器发流 | 成功添加设备 |
| 删除监控设备 | 设备已添加到NMC控制台系统 | 成功删除监控设备 |
| 采集器添加 | 采集器绑定该NMC控制台IP | 成功添加采集器 |
| 协议、应用监控 | 该条协议、应用没有被监控 | 成功监控该条协议、应用 |
| 协议、应用去监控 | 该条协议、应用已经被监控 | 成功对该条协议、应用去监控 |
| 创建IP组 | 系统正确安装 | 成功创建IP组 |
| 修改、删除IP组 | 已经创建了IP组 | 成功修改、删除该IP组 |
| 创建接口组 | 系统正确安装 | 成功创建接口组 |
| 修改、删除接口组 | 已经创建了接口组 | 成功修改、删除该接口组 |
| 创建应用组 | 系统正确安装 | 成功创建应用组 |
| 修改、删除应用组 | 已经创建了应用组 | 成功修改、删除该应用组 |
| 采集器删除、修改 | 采集器已经成功添加 | 成功删除、修改采集器 |
| 网络流量分析 | 查询DashBoard流量概览界面 | 系统正确安装，汇聚数据表中已存在数据 | 显示正确 |
| 查询接口维度分析界面 | 系统正确安装，接口汇聚数据表中已存在数据 | 显示正确 |
| 查询应用维度分析界面 | 系统正确安装，应用汇聚数据表中已存在数据 | 显示正确 |
| 查询IP组维度分析界面 | 系统正确安装，配置了IP组且有流量流经该IP组 | 显示正确 |
| 查询主机维度分析界面 | 系统正确安装，主机汇聚数据表中已存在数据 | 显示正确 |
| 查询接口组分析界面 | 系统正确安装，配置了接口组且有流量流经该接口组 | 显示正确 |
| 查询应用组分析界面 | 系统正确安装，配置了应用组且有流量流经该应用组 | 显示正确 |
| 查询设备维度分析界面 | 系统正确安装，汇聚数据表中已存在数据 | 显示正确 |

# 第五章 工作总结与展望

## 5.1 工作总结

本论文基于NetStream技术实现了可以实时监控电商数据仓库IDC之间网络流量的系统，该系统能够对环境中的网络流量进行统计并且通过直观的图形化界面详细、准确的显示出网络流量的详细组成，帮助用户实现网络流量可视、故障可查、规划可依的网络透明化管理目标，并且网络管理员也可以及时了解网络流量动态，确保网络带宽得到充分合理的使用，保障网络长期稳定高效运行，为优化网络设计和规划，实现以最小的网络运营成本达到最佳的网络性能和可靠性提供了坚实基础。

本文首先说明了选择这个课题的背景以及实现这样一套系统的现实意义，结合国内外网络流量监测的发展现状对本论文所要实现的内容进行了额阐释。而后对NetStream技术做了基本的介绍，以及NetStream系统的各个组成以及作用进行了详细说明，更进一步的解释了系统整体的运行机制，包括系统的流量统计、采样方式以及几种流老化的方式和处理结果的输出格式。紧接着对网络流量监测系统的总体架构以及各个功能点进行了详细设计。并对网络流量监测系统的功能进行实现，并贴出实现过程中涉及到的代码的核心部分。最后系统进行功能性测试并得出相应结果。

## 5.2 工作展望

本系统实现了对网络流量的监测，可以显示出流量的详细信息。但是目前网络发展如此迅速，流量将会越来越庞大，对带宽的要求也会越来越高，从系统测试环节可以看出，系统的正常运行对服务器配置要求比较高，况且系统可以监控的数据只能达到几百台，对于现在一些比较大型的场景的几千甚至几万台设备无法胜任。目前对该系统存在一些可以改进的方面：

（1）系统所有模块的操作产生的结果最终保存到数据库里面，并且所有的显示也是要经过数据库的多次查询，某些界面存在众多控件，加载一个页面存在太多次的数据库查询或者修改、删除等操作。这将导致系统的性能下降，同时也会浪费不必要的资源，界面也会出现卡顿等状况。所以该系统可以在数据库查询方面进一步优化，可以设计更为高效的表结构，简化查询流程，防止冗余查询，也可以设计更高效的查询方法提高查询的速率和性能。

（2）对于日后更加智能的时代，系统也应该趋于智能化，目前系统仅仅提供数据的显示，以及一些基本组成、速率等的分析，对网络状况的总体状况比如哪些是异常流量，哪些应用的流量偏高并没有分析功能，所以系统可以对分析功能进行加强，提供一些比较好推测的分析结果。

# 致 谢

当下正值毕业季，经过了这段时间的努力，毕业论文到了收尾部分，毕业论文的完成也预示着我的大学生涯即将结束，在这四年的时间里，跟随众多优秀的老师的步伐，我学到了很多知识与技能，大学四年的生活让我成长了一大步。感谢这四年一直陪伴着我的老师和同学，学习生活没有他们的帮助会很艰难。

首先，感谢李伟导师在这段时间以来极其细心又严谨的指导，老师每次主动找我们交流，无私的给我们提供指导和帮助，从任务书、开题报告到毕业设计论文，全部内容老师层层把关，对我的论文提出了很多改进建议，让我的论文一点点的变得完善，真的对李伟老师非常的感激，可以说，没有李伟老师的指导，本篇论文我自己很难完成。

接着感谢企业导师张冰老师对我实习期间的无微不至的教导，初次进入公司工作的我，陌生的环境致使我需要花费大量时间去熟悉公司文化。作为我的企业指导老师，从技术到工作态度，尽管他自己一直很忙，但是他还是不厌其烦的对我提出的问题进行解答，帮助我更快的适应了公司的生活。同时也感谢其他同事，他们的关怀让我的工作更有效率。

感谢学校给我提供这次实习机会，让我不再停留在理论阶段，真正的进入实践中去运用知识。同时这样的实习也让我们以后可以更快的融入企业工作生活。让我们少走了许多弯路，同时也感谢学校提供给我的学习环境，让我可以无忧无虑的享受学习生活。

感谢所有的同学，论文的很多不懂之处也是经由同学们的帮助下才解决。意外感谢我的家人，在我的学习生活上一直支持着我。

由于学术水平有限，本论文还有很多不完善的地方，如有不足，烦请老师和同学指正。

# 参考文献

1. 陈春霞. 基于容器的微服务架构的浅析[J]. 信息系统工程, 2016 (3): 95-96.
2. Fielding R T. Architectural styles and the design of network-based software architectures[D]. University of California, Irvine, 2000.
3. 唐文宇. 面向 SOA 架构微服务的安全系统的设计与实现[D]. 南京大学, 2016.
4. Abrams S, Kunze J, Loy D. An emergent micro-services approach to digital curation infrastructure[J]. International Journal of Digital Curation, 2010, 5(1): 172-186.
5. Namiot D, Sneps-Sneppe M. On micro-services architecture[J]. International Journal of Open Information Technologies, 2014, 2(9).
6. 徐鹏, 陈思, 苏森. 互联网应用 PaaS 平台体系结构[J]. 北京邮电大学学报, 2012, 35(1): 120-124.
7. Newman S. Building microservices[M]. " O'Reilly Media, Inc.", 2015.
8. 王克苑, 张维勇, 王建新. SSL 安全性分析研究[J]. 合肥工业大学学报: 自然科学版, 2004, 27(1): 87-91.
9. 王珊, 王会举, 覃雄派, 等. 架构大数据: 挑战, 现状与展望[J]. 计算机学报, 2011, 34(10): 1741-1752.
10. Brooks F P, Li Q. 人月神话[M]. 中国电力出版社, 2003.