**毕业设计（论文）题目： 虚拟网络流量可视化技术研究**

**学 院： 信息与电子学院**

**专 业： 电子科学与技术**

**班 级： 05411202**

**姓 名： 王本亮**

**指导教师： 苏京霞**

目 录

[第1章 绪 论 1](#_Toc169323422)

[1.1 研究背景和意义 1](#_Toc169323423)

[1.2 研究历史和现状 2](#_Toc169323424)

[1.2.1 发展历史 2](#_Toc169323425)

[1.2.2 研究现状 1](#_Toc169323426)

[1.2.3 总结与分析 1](#_Toc169323427)

[1.3 研究内容和结构安排 1](#_Toc169323428)

[第2章 涉及的理论与技术基础 1](#_Toc169323429)

[2.1 引言 1](#_Toc169323430)

[2.2 虚拟网络组成与构建 5](#_Toc169323434)

[2.2.1 虚拟网络的组成 5](#_Toc169323435)

[2.2.2 虚拟网络的路由 6](#_Toc169323436)

[2.3 流量监听技术与实践方法 5](#_Toc169323434)

[2.3.1 流量监听 5](#_Toc169323435)

[2.3.2 实践方法 6](#_Toc169323436)

[2.4 数据存储方法 5](#_Toc169323434)

[2.5 服务器与客户端双向通信方法 5](#_Toc169323434)

[2.5.1 Websocket技术简介 5](#_Toc169323435)

[2.5.2 Websocket应用场 6](#_Toc169323436)

[2.6 可视化技术理论基础 5](#_Toc169323434)

[2.6.1 引言 5](#_Toc169323435)

[2.6.2 动态页面设计方法 5](#_Toc169323435)

[2.6.3 Echart服务框架简介 6](#_Toc169323436)

[2.7 逻辑-视图-模版（MVT）框架 5](#_Toc169323434)

[2.7.1 MVC框架介绍 5](#_Toc169323435)

[2.7.2 MVT框架介绍 6](#_Toc169323436)

[2.8 消息队列功能 5](#_Toc169323434)

[第3章 系统设计 9](#_Toc169323437)

[3.1引言 9](#_Toc169323438)

[3.2 系统总体设计 9](#_Toc169323438)

[3.2.1 技术路线与设计原则 9](#_Toc169323439)

[3.2.2 目标与功能需求 10](#_Toc169323440)

[3.2.3 系统总体结构 12](#_Toc169323441)

[3.3 流量监听模块 13](#_Toc169323442)

[3.3.1流量监听模块功能与性能要求 14](#_Toc169323443)

[3.3.2 技术框架选取 26](#_Toc169323446)

[3.4 通信服务器模块 28](#_Toc169323448)

[3.4.1 通信服务器功能 28](#_Toc169323449)

[3.4.2 通信服务器技术框架 28](#_Toc169323450)

[3.4.3 数据库功能与需求分析 29](#_Toc169323451)

[3.5 数据可视化展示模块 28](#_Toc169323448)

[3.5.1 数据展示要求 28](#_Toc169323449)

[3.5.2 数据展示技术架构 28](#_Toc169323450)

[第4章 系统实现 38](#_Toc169323456)

[4.1主要功能需求说明 38](#_Toc169323457)

[4.1.1 系统工作流程 28](#_Toc169323449)

[4.1.2 流量监听的硬件与软件环境 28](#_Toc169323450)

[4.1.3 流量监听性能指标 28](#_Toc169323449)

[4.1.4 通信服务器功能需求分析 28](#_Toc169323450)

[4.1.5 数据展示模块而功能需求分析 29](#_Toc169323451)

[4.2关键功能模块实现 38](#_Toc169323458)

[4.2.1流量监听模块 38](#_Toc169323459)

[4.2.2数据存储与读取功能模块 38](#_Toc169323460)

[4.2.3 通信服务器模块技术实现 38](#_Toc169323460)

[4.2.4 数据展示模块功能需求分析 38](#_Toc169323460)

[4.2.5 数据展示模块的逻辑实现 38](#_Toc169323460)

[4.3 系统功能与稳定性测试 43](#_Toc169323462)

[4.3.1 系统功能测试 38](#_Toc169323460)

[4.3.2 系统稳定性测试 38](#_Toc169323460)

[4.3.3 测试环境和条件 38](#_Toc169323460)

[4.3.4 评价方法 38](#_Toc169323460)

[4.3.5 结果与分析 38](#_Toc169323460)

[4.4 小结 43](#_Toc169323462)

[第5章 结束语 48](#_Toc169323463)

[5.1 全文总结 43](#_Toc169323462)

[5.2 工作展望 43](#_Toc169323462)

[致 谢 49](#_Toc169323464)

[参考文献 50](#_Toc169323465)

[附 录 51](#_Toc169323466)

# 摘 要

# Abstract

**第1章** **引** **言**

## 1.1 研究背景和意义

随着信息产业技术的发展，各类计算机和移动终端的使用越来越普遍，而网络应用是其中最主要的应用。所以，当今社会人们对于网络信息的交流依赖度越来越高，带动了信息数量的增加和传播速度的上升。但与此同时，我们也不得不为此感到担忧，因为伴随着网络的扩张和信息量的增多，必将使网络基础设施负载更高的压力，影响网络性能的稳定性，这些都将会给人们在网络管理、分析和安全方面带来不小的挑战。

因此，网络提供者与相关的管理用户迫切希望能够在不影响网络性能和可靠性的前提下，通过对流量的收集和分析，更好的监测和控制网络。然而，网络流量的捕捉和分析却是个非常复杂的流程，当中涉及到流量的提取、分类和优先等级的分配，然后还要分发至网络管理、分析和安全等有关的工具。长久以来，传统的网络流量分析方法一直是通过大规模添加工具与系统，变更以太网[交换机](http://www.vsharing.com/module/27720.html)的用途，借助镜像端口复制流量，以及通过网络TAP（分路器）分拆流量。对各种用户来说，一个创新的流量可视化方案是他们亟需渴望的。因为这将为企业用户的IT部门和多种监测分析工具带来智能化的可视性，集中监控、简化运作，优[化工](http://www.vsharing.com/industry/1617.html" \t "_blank)具性能，节约成本等实质性的好处。

## 1.2 研究历史和现状

### 1.2.1 发展历史

虚拟化技术在 20 世纪 60 年代首次开发，当时是为了对大型机硬件进行分区以提高硬件利用率。30 多年前，IBM 率先实施虚拟化，作为对大型机进行逻辑分区以形成若干独立虚拟机的一种方式。这些分区允许大型机进行“多任务处理”：同时运行多个应用程序和进程。由于当时大型机是十分昂贵的资源，因此设计了虚拟化技术来进行分区，作为一种充分利用投资的方式。在 20 世纪 80 年代和 90 年代，由于客户端-服务器应用程序以及价格低廉的x86 服务器和台式机成就了分布式计算技术，虚拟化实际上已被人们弃用。20 世纪 90 年代 Windows 的广泛使用以及 Linux 作为服务器操作系统的出现奠定了 x86 服务器的行业标准地位。x86 服务器和桌面部署的增长带来了新的 IT 基础架构和运作难题：基础架构利用率低；物理基础架构成本日益攀升；IT 管理成本不断攀升；故障切换和灾难保护不足；最终用户桌面的维护成本高昂。

1999 年，VMware 推出了针对 x86 系统的虚拟化技术，旨在解决上述很多难题，并将 x86 系统转变成通用的共享硬件基础架构，以便使应用程序环境在完全隔离、移动性和操作系统方面有选择的空间。x86 计算机与大型机不同，它在设计上不支持全面虚拟化，因此必须克服难以解决的难题才能在 x86 计算机上开发出虚拟机。在大型机和 PC 中，大多数 CPU 的基本功能都是执行一系列存储的指令（即软件程序）。x86 处理器中有 17 条特定指令在虚拟化时会产生问题，从而导致操作系统显示警告、终止应用程序或直接完全崩溃。因此，这 17 条指令是在 x86 计算机上首次实现虚拟化时的严重障碍。为应对 x86 体系结构中会产生问题的这些指令，VMware开发了一种自适应虚拟化技术。在生成这些指令时此技术会将它们“困住”，然后将它们转换成可以虚拟化的安全指令，同时允许所有其他指令不受干扰地执行。这样就产生了一种与主机硬件匹配并保持软件完全兼容性的高性能[虚拟机](http://www.vmware.com/cn/technology/virtual-machine.html" \t "_blank)。

在VMware逐渐在企业级市场中被广泛的接受后，Xen也逐渐在互联网领域崭露头角。在成熟的服务器操作系统当中，Novell SUSE Linux Enterprise 10是第一个采用Xen技术的。当时的Xen还很不成熟，但是在2006年，RHEL 5.0发布的时候，红帽公司决定也将Xen加入到自己的默认特性当中，一时之间，在Linux服务器领域，Xen似乎成为了VMware之外的最佳虚拟化选择。但是，作为一项Linux平台上的虚拟化技术，Xen在很长一段时间内一直没有被接受到Linux内核的代码当中，这对于Xen的维护者而言，不仅意味着要多做很多工作，还意味着用户在废了半天劲装好Xen之后可能遇到意想不到的问题。

2008年9月，红帽收购了一家名叫Qumranet的以色列小公司，由此入手了一个叫做KVM的虚拟化技术（KVM，全称Kernel-based Virtual Machine，意为基于内核的虚拟机）。

2010年，IBM系统软件部市场与销售副总裁，同时也是开放虚拟化联盟委员会成员之一的Inna Kuznetsova公开表示：KVM已经对数据中心做好了准备。

为了进一步降低桌面虚拟化的投入成本，Citrix在2011年5月发布了IntelliCache技术。在过去，VDI项目都是使用共享存储，服务器的本地存储设备被闲置，是一个不小的浪费，IntelliCache技术可以把虚拟机的主镜像文件传送到服务器的硬盘上，使虚拟机对存储的读写由原来的共享存储，转向对服务器本地存储设备的读写，既节省了共享存储的投入成本，又有效利用了现有的投资，是一个双赢的结果。

在虚拟化技术大规模应用于生产之后，随着科学技术的不断发展,计算机网络技术的应用也日益普及.在目前的计算机网络安全工作中,虚拟网络技术是比较重要的一类,并且在很多方面都发挥了较大的积极作用。从成本、占地空间、硬件资源管理等各个方面来看，虚拟网络技术的应用是必然的路径。考虑到今后的计算机网络拥有较大的空间,在应用虚拟网络技术的过程中,需考虑不同的环境和应用标准,以此来实现计算机网络安全的巩固。

然而，在使用方便的同时，软件资源管理的问题也日益突出。

### 1.2.2 研究现状

目前数据可视化的研究主要偏向于统计数据的展示呈现，往往通过新的理念模式或者算法对海量的信息数据进行处理与计算得到统计学结果或者计算出二维乃至多维的数据相关性，再通过图表的形式展现出来。这种可视化技术，注重的是对大数据的分析，注重的是结果，而且可视化技术在该研究中往往属于非主要部分，即展现统计结果。

### 1.2.3 总结分析

目前的可视化技术主要存在三个问题：展示界面不够友好，由于常规的可视化技术更注重根据数据和算法所获得的计算结果的准确性，所以计算结果通常在命令行中用块儿状或者折线状视图展示，展示工作不够美观；实时性不够高，将可视化技术应用在流量分析工作中，对展示结果往往有着较高的实时性要求，传统的计算方法要花费较长的时间进行离线计算；占用过多的系统资源，传统的可视化技术在计算过程中，一方面由于要读取大量的数据，所以有较多的磁盘I/O，占用时间，占用内存，妨碍其他程序在服务器上高校运行，另一方面，由于算法大多为计算密集型程序，所以同样会占用较多的CPU资源，耗电，耗费计算节点的计算资源。

本文研究的虚拟网络流量可视化技术，在由于采用了新的通信协议，降低了服务器资源的损耗；采用消息队列的机制，保障了系统的实时性；使用开源的前端美化框架，使监控结果看起来更加清楚，更加美观。

## 1.3 研究内容和结构安排

### 1.3.1 研究内容

本问介绍了一套高性能、高可靠、高可扩展的虚拟网络流量监控系统。系统分为三个模块，分别为：虚拟网络流量监听及流量处理模块；系统通信服务器模块儿；流量数据可视化展示模块。

其中，虚拟网络流量监听及流量处理模块和系统通信服务器模块的设计架构为C/S架构；流量数据可视化展示模块和系统通信服务器模块的设计架构为B/S架构。两端的S部分为集成的系统通信服务器程序，便于程序的监控管理和后期的功能扩展以及个性化功能的定制。论文的主要研究内容包括以下几个方面：

1. 介绍虚拟网络流量监听常用技术与具体实现的方法，并根据监听到的流量数据提取对应参数，为后期研究提供数据储备和为线上部署做好准备；
2. 介绍Web应用中消息队列的原理与实现方法。使用Redis数据库作为数据存储仓库和消息队列实现的软件基础；从消费者和生产者模型入手，详细介Redis的消息的发布/订阅（Pub/Sub）模式，讲述二者的区别，以及使用Pub/Sub的原因；
3. 介绍Web程序的B/S架构设计模式中客户端与服务端的双向通信方式即采用WebSocket协议，WebSocket是区别于HTTP的应用层协议；本文详细介绍了基于Tornado框架的WebSocket协议的代码实现过程，并且将消息队列应用于双工通信中；
4. 介绍MVC设计模式的原理，总结MVC设计模式的优点，研究使用Tornado框架来提高系统的稳定性的方法，并研究其在对应开发框架的变化和应用；
5. 简述html5的基本使用，介绍Echart开源服务框架的使用方法，以及个性化定制技术；
6. 基于以上技术实现的网络流量可视化系统的开发过程和其他相关技术的介绍；
7. 将网络流量可视化系统布置在虚拟网络环境下的自动化程序。

### 1.3.2 结构安排

论文共分为5章。第1章是引言部分，主要对虚拟化网络流量可视化技术的研究背景、研究意义做简单介绍；回顾虚拟化技术的发展历史，介绍虚拟化网络中流量可视化技术的研究现状，同时对国内外研究的现状作简要分析。第2章论述理论基础和相关的技术，包括虚拟化网络组建、虚拟化网络流量的监听技术、使用Redis数据库实现的消息队列、WebSocket协议的原理与使用方法、Tornado服务框架的原理与使用、Echart服务框架的使用与个性化定制等。第3章阐述了虚拟化网络流量可视化系统的总体设计的技术路线和设计原则，分析了目标与功能需求，详细描述了流量监听模块、通信服务器模块和数据可视化展示模块的技术架构选取原则与方法。第4章具体展示了虚拟网络流量可视化系统的各个模块的实现细节，并介绍了系统的测试方法，提供了一部分测试所得数据及结果分析。第5章为全文做了总结，并介绍了下一步的工作展望。

**第2章** **涉及的理论与技术基础**

## 2.1 引言

本系统分为三个模块，应用了一组C/S架构，一组B/S架构。涉及了虚拟网络组建、流量监听、NoSQL数据库、WEBSOCKET协议、可视化技术、MVC设计模式、消息队列等理论和技术。

## 2.2 虚拟网络组成与组建

### 2.2.1 虚拟网络的组成

虚拟网络是指在少量性能强大的服务器上才用虚拟化技术，构建起多台虚拟主机，但是其网络组建是各自独立的，虚拟主机在网络中的地位与物理主机并无差异；同构建实体网络一样，在设置好网络ID之后，为各个终端，即虚拟主机分配主机ID。为各个终端设置好IP地址、网关与子关掩码后，只需将虚拟主机所在的宿主机接入我们配置的网络，并在虚拟机管理程序中进行少量配置就可以把虚拟主机接入网络。虚拟主机的使用与物理主机使用并无差异，但是有方便管理、容灾性好、可以实时备份或者恢复等有点，同时，在批量配置虚拟主机时还可以使用镜像直接操作，方便快捷。

### 2.2.2 虚拟网络的路由

关于虚拟网络的路由，从虚拟主机的角度来看，其特性与物理主机并没有差异，其虚拟特性对自身来讲是透明的；但是虚拟主机的所有设备文件，都是基于虚拟化技术虚拟产生的，包括CPU、硬盘、内存等，这里当然包括网卡，但是主机想要进行网络I/O又必须需要物理网卡，所以由虚拟主机的宿主机来完成这一任务。

虚拟主机的网络配置基本分为三种，分别为桥接、NAT、Host-Only，目前市场上主流的虚拟软件，包括VMware Station，Virtual Box,，Virtual PC 以及企业级应用vSphere都支持这三种虚拟网络配置模式。

1. 桥接：桥接网络是指本地物理网卡和虚拟网卡通过VMnet0虚拟交换机进行桥接，物理网卡和虚拟网卡在拓扑图上处于同等地位，那么物理网卡和虚拟网卡就相当于处于同一个网段，虚拟交换机就相当于一台现实网络中的交换机,所以两个网卡的IP地址也要设置为同一网段。所以当我们要在局域网使用虚拟机，对局域网其他pc提供服务时，例如提供ftp、ssh、http等服务，那么就要选择桥接模式。
2. NAT模式：该模式就是让虚拟机借助NAT(网络地址转换)功能，通过宿主机器所在的网络来访问公网。NAT模式中，虚拟机的网卡和物理网卡的网络，不在同一个网络，虚拟机的网卡，是在vmware提供的一个虚拟网络。
3. Host-Only：在Host-Only模式下，虚拟网络是一个全封闭的网络，它唯一能够访问的就是主机。其实Host-Only网络和NAT网络很相似，不同的地方就是Host-Only网络没有NAT服务，所以虚拟网络不能连接到Internet。

本系统组建的虚拟网络的路由模式是才用桥接和NAT来年两种网络模式，拓扑图如图2-1所示：



图2-1 虚拟网络拓扑图

## 2.3 流量监听技术与实践方法

基于以太网的网卡，有两种工作模式。一种模式称为一般模式，即网卡正常工作，在接收数据帧时，网卡只关心与自己有关的数据包，这时候，网卡仅接受数据帧中目的地址与自身网络接口地址相同的数据帧，然后交给操作系统或者相关应用程序进行处理；另外一种为混杂模式，即网卡监听自身所在网络内的所有数据帧，接收所有数据帧，而不管这些数据帧的目的地址是哪里，全部交给自身上层的程序进行处理。

考虑到程序的稳定行，我们将在Linux下部署监听程序，因此系统的开发环境也是Linux。在Linux系统下进行网络流量的监听，首先需要将网卡设为混杂模式，以便于网卡监听自身所在网络内的所有数据帧；以太网上的数据帧监听，主要关心TCP/IP协议簇中的IP协议，ARP协议，以及TCP协议和UDP协议。

传统的数据包捕捉平台主要是基于网络中但节点的操作系统。最早被云公用的是Raw Socket原始套接字，但这种技术只能捕捉IP包，对ARP包无效；BSD Packet Filter（BPF）机制相对来说也是基于FreeBSD下使用较多的捕捉平台。它对数据包的处理简单易行，但是对于高流量数据包的处理表现并不好。一方面，考虑到本系统对实时性的要求较高；另一方面，由于过度依赖UNIX内部的一些机制，频繁地使用系统调用，不便于移植，因此并不合适。美国洛仑兹伯克利国家实验室所编写的专用语数据包截获功能的API函数库Libpcap的设计目标，就是要使得数据包监听程序在不同的操作系统平台上更加容易地进行移植，它有强大的性能和良好的移植性使我们实现流量监听程序的首选，另一方面，很多编程语言都提供了封装了pcap的库，使我们在编程实现程序时有更多的选择[[[1]](#endnote-1)]。

## 2.4 数据存储方法

在获取流量监听节点监听及处理后的数据之后，系统将把数据进行存储和展示，我们的系统使用Redis数据库和MongoDB这两种NoSQL数据库。

首先，我们的系统的目标是准确、实时展示各个计算节点的流量情况，而不关心数据帧的源地址和目的地址的关系，所以我们使用NoSQL数据库MongoDB进行存储数据帧的详细信息，包括数据帧的参数信息、捕获时间、存储时间，由于该数据库有灵活的set和list数据结构，在后期我们可以进行统计学计算，包括指定节点的流量情况，特定的数据帧类型等。这里我们只关心每一条记录，而并不关心各个记录之间的关系，MongoDB有存取速度快、查找方便、并发性能好、灾难恢复容易等诸多有点。

另一方面，我们在进行流量可视化的展示的时候，并不可以把监听节点的数据直接交给前台模块展示，因为考虑到网络I/O，硬盘I/O等问题，展示模块收到的每一条记录很可能不是流量监听节点捕获的流量信息的正确顺序。系统需要使用一个速度快，性能好的中间数据库作为缓冲。这里使用Redis数据库，按照时间顺序接受并存储流量监听节点处理后的数据内容，并按照其存储的先后顺序，将数据发送给通信服务器程序，最终交由前台展示页面将数据信息通过可视化技术展现出来。Redis有队列以及双端队列等数据结构，其底层细节用C语言的字符串数据结构来实现，效率很高，同时又提供了多种查询方法，适合对实时性准确性要求较高的应用。

## 2.5 服务器与客户端实时通信方法

### 2.5.1 传统的实时应用解决方案

通常情况下，Web应用都使用Http协议来进行工作，但是http协议是无状态的协议，即服务器每次返回给客户端对应内容之后，就关闭连接。这样以来，客户端就没有办法接受新的数据并展示了。常规的解决办法是在客户端使用无限轮询的策略。所谓轮询就是客户端以一定的时间间隔向服务器发出请求，以频繁的HTTP请求的方式来不断刷新客户端呈现的信息。这种刷新是异步刷新页面的展示情况，通常使用的是Ajax技术。Ajax，是异步J啊vaScriptjeXNL技术的简称。才用Ajax技术方法，客户端利用JavaScriptdeXMLHttpRequest对象以异步的方式向服务器发送HTTP请求，在不重载整个页面的情况下完成与服务器的数据交换。利用Ajax的这一特性，可以避免对于客户端页面的频繁刷新。基于Ajax的长轮询技术是目前Web实时应用中使用频率较高的方式，这种方式才用Ajax方法向服务器发送请求，在服务器没有信息更新时，服务器会一直维持这个请求连接，直到服务端有新的数据产生，则服务器将新数据返回，否则服务器将一直维持这个连接直到这个连接被判定为连接超时，服务端才会关闭这个连接。基于Ajax的长轮询的具体工作过程如下：客户端发送请求，与服务端建立一个连接；然后服务端将这个连接保持一段时间，这个时间通常是几秒也可能是一分钟或者更长；当服务端检测到当前的客户端请求的数据有更新时，即有必要将新产生的数据发往客户端时，会立即通过正在维持的连接将数据发送给客户端，作为客户端此次请求的响应。=，然后关闭该连接；如果服务端在维持这个连接的整个过程中都没有新的数据发送到客户端，会在维持时间超过设定时间后关闭连接。无论是否有数据更新，在上一次连接关闭后，客户端会立即重新发出一个新的请求，再次与服务端建立连接，这个过程会一直循环下去，确保服务端的新数据能够及时的发往客户端，如果轮询间隔较长，那么系统的实时性无法保证。在基于Ajax的长轮询方案中，当服务端的数据更新速度较快时，长轮询将退化为普通的轮询（在我们的系统中，由于流量数据的量非常大，这种情况几乎是必然会发生的），这样将大大降低其性能，并且对服务器的内存资源造成浪费，也会给服务器的处理器带来较大的压力。另外，由于应用ajax技术的解决方案本质上仍然是才用HTTP作为通信协议，而HTTP连接的建立和关闭都有一定的资源和时间消耗。这不仅会导致服务器的资源浪费，也会使在连接建立和关闭过程产生的新数据无法及时发送到客户端，可能导致客户端数据丢失。每一次http请求和响应都会带有完整的http消息头，这就增加了每次实时信息更新时的传输数据量，造成了网络带宽的浪费。[[2]](#endnote-2)

### 2.5.2 WEBSOCKET协议简介

WebSocket协议[[3]](#endnote-3)并不是标准的socket协议，是HTML5标准中提出的一种新的协议，可以实现浏览器与服务端的双向通信。不同于传统的HTTP方式，在使用Websocket时，一旦连接建立，客户端与服务端就处于同等的地位，可以随时互相发送消息。Websocket协议本质上是一个基于TCP的协议，该协议采用的是单socket双向通信来进行推送或者接收信息，在客户端与服务端之间建立类TCPSocket的持续、双向、有状态的通信模式，实现真正意义上的长连接模式服务器推送，满足高时效性要求。这不仅可以避免轮询的问题，也可以避免在客户端实现的javascript代码在不同浏览器平台上的可移植性问题。WebSocket连接的建立过程与TCP连接的建立相似，因为websocket协议本身就是基于TCP/IP的底层协议进行实现的[[4]](#endnote-4)。

其连接过程为：浏览器通过Javascript向服务器发出建立websocket连接的请求，服务器依据HTTP首部判别是否为WebSocket请求。识别成功后，服务器将把该请求升级为一个WebSocket连接，并向客户端返回应答信息，然后建立连接，在连接建立起来之后，服务端和客户端就可以通过TCP连接直接进行数据交换。除了具备双向通讯的能力，WebSocket洗衣的传输数据格式相对于HTTP而言更加简洁。Websocket在传输信息时，相对于HTTP大大降低了数据帧中头信息的占用字节数，从而降低了传输的数据量，节省了带宽资源，降低了网络负载，同时提高了数据的传输效率，增强了应用的实时性。

## 2.6 可视化技术理论基础

### 2.6.1 引言

一幅图胜过千言万语，人门从外界获得的信息约有80%以上来自视觉系统[[5]](#endnote-5)，当数据以直观的可视化的图形形式展示在观察者面前时，观察者往往能够一眼洞悉数据背后隐藏的信息并转换成知识以及智慧。可视化技术，是数据分析的一种重要结果形式。

### 2.6.2 动态页面设计方法

动态页面是与静态页面相对应的一种网页设计方法。动态页面的URL不固定，可以通过后台程序与用户进行交互来完成请求、数据提交等动作[[6]](#endnote-6)。常用的语言有Python、PHP、ASP、JSP等。

动态页面有如下特点：

1. 动态页面的URL不固定，其后缀名一般为.asp、.jsp、.php等，与静态页面相比，动态页面的URL一个很大的特点就是它的URL中有个标志性的符号“？”[[7]](#endnote-7)；

2. 动态页面以数据库技术为基础，能够很大程度的减少网站维护的工作量；

3. 动态页面具有很好的交互性，当用户在页面上操作后，页面会根据用户的操作类型实时改变和响应。由于交互性很好，和静态页面相比动态页面能实现更丰富的功能，例如账号注册、账号信息管理、在线问卷调查等等；

4. 动态页面不是独立存在于服务器上的网页文件，只有当用户请求服务时，经过后台程序处理服务器才会返回一个完整的页面；

5. 由于动态页面的URL通常含有“？”，一般情况下搜索引擎不能访问一个网站数据库的所有页面，搜索引擎搜索时不会检索URL中“？”后面的内容，所以动态页面不容易被搜索引擎检索。

系统采用B/S架构，本质上是一个Web动态应用。系统采用Python作为后台程序编程语言，使用HTML、CSS、JavaScript、JSON等进行前端界面开发。

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它是基于JavaScript的一个子集。 虽然JSON采用一种完全独立于语言的文本格式，但是也使用了一些类似于C语言的习惯。正是因为这些特性使得JSON成为理想的数据交换语言。

### 2.6.3 ECHART服务框架简介

Echart是百度公司开发并进行开源的一个Web前端服务框架。它一个纯 Javascript 的图表库，可以流畅的运行在 PC 和移动设备上，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari等），底层依赖轻量级的 Canvas 类库 [ZRender](https://github.com/ecomfe/zrender" \t "_blank)，提供直观，生动，可交互，可高度个性化定制的数据可视化图表。

系统的可视化模块的前端逻辑是依靠Echart进行二次开发完成的。

## 2.7 逻辑-视图-模版（MVC）设计模式

### 2.7.1 MVC结构介绍

MVC(Model View Control)是一种框架模式，它将应用程序的输入、处理和输出分开[[8]](#endnote-8)，使用MVC框架的应用程序被分为模型、视图和控制器三个核心部件。如图 2‑1所示，M是指业务模型，表示应用程序的核心，在应用程序中用于处理应用程序数据逻辑；V是指用户界面，在应用程序中处理数据显示；C是控制器，处理数据输入，在应用程序中控制器主要用于处理用户交互。为了使同一个程序可以使用不同的显示形式，使用MVC将应用程序的模型和视图两个核心部件的实现代码分离。控制器作为三大核心部件之一是为了保证模型和视图的同步，视图随着模型的改变同步更新[[9]](#endnote-9)。



图 2‑2 MVC部件关系图

在MVC框架中，模型（Model）主要负责应用程序业务状态/流程的处理，并制定业务规则。业务处理的过程对于其他模块来说是透明的，模型接收视图发送的请求数据。如果模型发生变化，视图会得到通知，并且视图可以通过模型查询模型相关状态[[10]](#endnote-10)。同时，控制器可以通过模型访问封装在模型内部的应用程序功能。

视图（View）用来组织模型的内容，它负责提供用户交互界面。视图从模型那里获取数据并将这些数据按照制定的格式显示出来。当模型发生变化的时候，视图负责维持数据表现的一致性，并且将用户要求告知控制器。

控制器（Control）定义了应用程序的行为，它的主要作用是接收用户的请求。控制器对来自视图的用户请求进行解释，并把这些请求映射成相对应的行为，然后通过模型来实现这些行为。在一个Web应用程序中，用户请求的有很多种，比如来自网页的GET、POST或HTTP请求等[[11]](#endnote-11)。模型可以实现的行为分为两种，一种是修改模型的状态，另外一种是处理业务，控制器根据用户请求和模型行为的结果选择一个视图作为对用户请求的应答。通常情况下一个控制器负责一组相关的功能集。

MVC框架的处理用户请求包含三个过程[[12]](#endnote-12)：1）控制器接收用户请求，并根据用户请求的类型决定调用哪个模型来处理用户的请求；2）模型根据用户请求进行相应的业务逻辑处理，并返回数据；3）控制器调用相应的视图格式化模型返回的数据，并通过视图呈现给用户。

MVC框架具有如下几个优点：1）在一个应用程序中同一个模型可以被不同的视图重用，这样大大提高了代码的重复利用率；2）MVC的三个模块相互独立，其中某一个模块的改变不会影响其他两个模块，根据这种设计思想能构造良好的松耦合的构件；3）控制器大大提高了应用程序的灵活性和可配置性，控制器可以用来连接不同的模型和视图去完成用户的请求，控制器为构造应用程序提供了强有力的手段。

### 2.7.2 MVT结构介绍

MVT框架是基于MVC的设计理念变化而来，是Model-View-Template设计模式的简写。在MVC设计模式的基础上，一些Web开发框架的结合自身语言的特点进行变化，形成了MVT框架。其中Model为数据模型模块；View是页面展示的后端的逻辑，根据需求从数据模型中提取数据或者调用对应方法筛选数据，返回给Template模块进行展示；Template是展示模块，其本质一些包含了HTML、Javascript的静态页面和标签与数据展示程式的规则，比较常用的有Jinja2。

MVT与MVC都是Web开发设计模式，其意在提供模块化的web开发建议，不同模块间相互分离，各自开发，自身实现细节对其他模块是透明的，这种设计模式也是面向对象思想在Web开发中的一种体现。

## 2.8 消息队列功能

### 2.8.1 消息中间件的要素

本系统中为了进行有序实时展现监听模块监听与分析后的数据，需要使用消息中间件模块。作为一个消息中间件模块，需要确定五个要素：

1. 消息模型。根据业务场景，需要确定消息中间件的类型，消息中间件分为两种类型一种是生产者消费者模型；一种是发布订阅模型，，客户端与服务端建立连接后，消息中间件把新获取的数据，传给服务器，服务器把数据内容推送给每一个与服务器建立了socket连接的客户端。
2. 消息投递可靠性。一个消息在传递过程中可能出现如下几种情景：At-most-once(消息丢失)， 消息被投递0或者1次，消息可能被丢失；At-least-once(消息重复)，消息可能被投递多次，直到收到ack，会造成消息重复，开启消息持久化及ack机制，可以保证此投递策略，确保消息投递成功，但可能造成消息重复；exactly-once(消息仅且投递一次)，消息通过事务等机制保证对方能成功接收，而且不会重复接收。
3. 消息持久化。即消费者是否可以接收离线时的消息。
4. 消息优先级。消息是否支持优先级。
5. 消息回溯。消息是否可以重复消费

### 2.8.2 Redis的Pub/Sub模型

本系统采用Redis的Pub/Sub模型来提供消息队列的服务。Pub/Sub功能（means Publish, Subscribe）即发布及订阅功能。基于事件的系统中，Pub/Sub是目前广泛使用的通信模型，它采用事件作为基本的通信机制，提供大规模系统所要求的松散耦合的交互模式：订阅者(如客户端)以事件订阅的方式表达出它有兴趣接收的一个事件或一类事件；发布者(如服务器)可将订阅者感兴趣的事件随时通知相关订阅者。在本系统中，采用单队列单消息分发器的模式，其原理如图2-2所示：



图 2-2 Redis的Pub/Sub模型图

pub/sub功能可以有三个非耦合：时间非耦合，也就是发布者和订阅者不必同时在线，它们不必同时参与交互；空间非耦合，也就是发布者和订阅者不必相互知道对方所在的位置。发布者通过事件服务发布时间，订阅者通过事件服务间接获得事件。发布者和订阅者不需要拥有直接到对方的引用，也不必知道有多少个订阅者或者是发布者参与交互；同步非耦合，发布/订阅是异步模式。发布者可以不断的产生事件，而订阅者(通过一个回调)则可以异步地得到产生事件的通知。

**第3章** 系统设计

## 3.1 引言

本章将介绍系统的功能要求，技术路线与设计原则，系统的结构三个方面阐述本系统的设计思想。另一方面，本章也详细介绍了系统的三个功能模块各自的功能要求、性能要求以及技术架构设计选择。

## 3.2 系统总体设计

### 3.2.1 技术路线与设计原则

根据功能需求，才用单线流程，即工作流决定程序模块功能的设计思想，本系统采取了一组C/S架构，一组B/S架构，对系统的功能进行模块化设计。秉着高内聚低耦合的设计原则，在保证系统的各个功能正常实现的基础上，减少不必要的功能、资源的使用，以减少CPU占用率、系统内存及其他相关硬件的使用。系统采用数据耦合的方式设计为三个模块，分别是使用http和websocket通信协议进行通信，按模块进行开发，方便调试和系统扩展。

### 3.2.2 目标与功能需求

目标系统要求能够准确、实时监控指定虚拟网络的流量情况，提取相关参数，并通过可视化技术展现为动态页面。系统功能需求如下：

1. 系统的流量监控模块能够稳定、实时监控虚拟网络的流量情况，并提取出数据帧类型、源IP地址、目的IP地址、源Mac地址、目的Mac地址以及数据帧的长度这些参数，通过http请求的POST方法发送到通信服务器模块，不需要接受反馈以及重复发送。
2. 可视化展示页面能够通过浏览器打开，在页面上能看到虚拟网络中被监听节点的数量，实时显示各节点之间的流量变化情况。
3. 流量监听节点所获取的数据需可持续化存储，为后期开发场景复现功能留下功能接口和数据记录。

系统的性能参数包括：

1. 可视化页面的设计需要兼容包括IE 7/8/9/10、Chrome、Firefox在内的多种浏览器。
2. 能够监听大于50个虚拟主机节点，CPU占用率小于5%，占用内存小于400MB。
3. 能够支持超过500个客户端打开可视化展示页面。
4. 在100Mb/s的有限网络中，可视化展示页面与监控系统的时间误差不超过3秒。

### 3.2.3 系统结构

系统的结构被分为两个部分，三个模块。第一部分由流量监听节模块和通信服务器模块组成，是C/S架构；第二部分由可视化数据展示页面和通信服务器模块，为B/S架构。下面详细介绍三个模块的功能与输入输出：

1. 流量监听模块。该模块负责监听网络节点的流量，并对流量数据帧进行处理，提取出相关参数，其输入内容为被监听节点的流量，经过自身程序的分析和计算，输出内容为流量的相关参数。
2. 通信服务器模块。作为两组架构的S端，通信服务器有着至关重要的作用，该模块连接着流量监听模块和数据展示模块，与流量监听程序进行通信时，接受流量监听模块的处理结果，把接收到的数据进行存储并进行进一步处理；与数据可视化模块进行协同工作时，监听服务器的指定端口，接受并建立合法的websocket连接，把从流量监听模块接收到的数据，稳定有序地传递给数据展示模块进行可视化操作，即把数据发布给已经与服务器建立了websocket连接的所有客户端。
3. 数据可视化模块，作为系统第二部分的B端，这个模块的运行环境即为客户端的浏览器。在进行此部分的程序的开发的过程中，要着重考虑程序运行环境的兼容性和可移植性。

系统的架构示意图，如图2-3所示：



图2-3 系统总体架构示意图

## 3.3 流量监听模块

### 流量监听模块功能与性能要求

1. 功能要求:实现一个存在于虚拟网络中的动态流量检测程序，在网络流量出口交换机放置嗅探器，分析出流量对应的数据包的类型，只采集和监听包括但不限于ICMP/TCP/UDP/HTTP类型的数据包；通过程序处理提取指定的参数，包括源IP地址、源Mac地址、目的IP地址、目的Mac地址、数据包类型、数据包长度、捕获日期，提供可选参数‘数据包内容‘为日后功能升级留下接口。
2. 性能要求：(1) 监测节点的数目大于50个；(2) 节点流量检测模块的CPU占用量小于5%，内存占用小于400MB；(3) 监测节点的网络流速不小于10Mb/s；(4) 支持的应用协议类型大于10种，实时数据显示延迟时间小于3秒；

### 技术框架选取

系统采用Libpcap来进行流量监听模块的开发。Libpcap 的英文意思是 Packet Capture library，即数据包捕获函数库。该库提供的 C 函数接口可用于需要捕获经过网络接口（通过将网卡设置为混杂模式，可以捕获所有经过该接口的数据报，目标地址不一定为本机）数据包的系统开发上。

Libpcap 提供的接口函数主要实现和封装了与数据包截获有关的过程。这个库为不同的平台提供了一致的编程接口，在安装了Libpcap的平台上，以 Libpcap 为接口写的程序，能够自由的跨平台使用。在Linux 系统下，Libpcap 可以使用BPF（Berkeley Packet Filter）分组捕获机制来获得很高的性能。这个库为不同的平台提供了一致的编程接口，在安装了 Libpcap 的平台上，以 Libpcap 为接口写的程序、应用，能够自由的跨平台使用。操作系统所提供的分组捕获机制主要有三种：BPF（Berkeley Packet Filter），DLPI（Data Link Provider Interface），及Linux下的=SOCK\_PACKET 类型套接口。基于BSD的系统使用BPF，基于SVR4 的系统一般使用DLPI。从文献上看BPF比DLPI 性能好很多，而SOCK\_PACKET 更弱。Libpcap主要由两部分组成：Network Tap、Packet Filter。Network Tap从网络设备驱动程序中收集数据拷贝，PacketFilter 决定是否接受该数据包。目前很多优秀的网络数据包捕获软件都是以Libpcap为基础，如著名的Tcpdump、Ethereal 等。[[13]](#endnote-13)

## 3.4 通信服务器模块

### 通信服务器模块功能

通信服务器根据其在系统的两部分架构中的角色不同可以分为如下两点：

1. 在C/S架构部分，作为http服务器的server构建起一个restful的web系统，用于接收流量监听模块获取并分析的流量数据的结果；并把获取到的数据存到MongoDB数据库中作为实验数据保存，以及用作流量参数和动态情况的离线使用；把数据加入到Redis做的Pub/Sub消息队列中，留待其他程序调用。
2. 在B/S架构部分，作为一个S端。与客户端(即数据可视化模块的用户)通过类TCPsocket方法建立连接，完成初始化，并且可以持续与客户端进行全双工非阻塞同步通信。在与数据可视化模块的程序建立起websocket连接之后，一直监听Redis数据库提供的消息队列服务，并且从Redis的消息队列中，按照一定的规则，获取消息，经过逻辑程序的处理后，发布给正在与服务端进行websocket通信的所有客户端，并确认状态，在客户端丢失或关闭连接后，通信服务器需要关闭该连接，回收服务器的相关资源。

### 通信服务器技术框架

通信服务器的结构如图2-4所示：



图2-4 通信服务器的结构图

通信服务器模块才用Linux操作系统，使用Nginx作为代理程序，使用Tornado Framework作为服务端的开发框架，其提供了两部分逻辑处理程序：一部分用于与流量监听模块进行交互，将收到的Post数据存入数据库，并且加入到消息队列中去；另一部分，与数据可视化的模块通过websocket进行通信，并随时把消息队列中出队的消息发送给数据可视化模块进行展示。

### 数据库功能与需求分析

如图2-4所示，在通信服务器这个模块当中有两个地方用到了数据库程序。一个作消息队列使用，另一个作数据存储使用。其中，使用Redis 的Pub/Sub机制做消息队列的原因这里不再赘述。另一方面，我们选取了MongoDB数据库来实现数据存储。

MongoDB的主要特点是开源设计、高性能、易部署、易使用，存储数据方便，它的面向集合存储、模式自由、支持动态查询、支持完全索引[[14]](#endnote-14)。从需求出发，我们只需要存储每一条数据的详细信息即可。包括源IP地址、源Mac地址、目的IP地址、目的Mac地址、数据包类型、数据包长度、捕获日期。因此，我们不需要为虚拟机节点建立数据模型，在当前阶段我们也不需要了解各个节点之间的通信关系和通信次数等高级信息，所以无需使用表结构相对复杂的关系型数据库，比如MySQL或PostgreSQL等，与这些数据库相比MongoDB的响应速度要快的多；另一方面，系统对这一部分的功能要求仅仅是提供高效率的存储，所以对数据库的事务性等高级功能不做要求，所以使用NoSQL数据库是更好的选择。

## 3.5 数据可视化展示模块

### 数据展示要求

对浏览器中呈现的数据可视化结果页面的展示有如下要求：

1. 要求能观察到各个监听节点的流量接受和发送情况，从抓取、计算、传递到页面展示，延迟不低于3s；
2. 兼容包括IE 7/8/9/10、Chrome、Firefox在内的至少三种浏览器。

### 3.5.2 数据展示技术架构

数据可视化模块的主要包括这几个模块：Websocket的客户端代码，用于与服务器建立websocket连接和接受新的数据信息；基于Echart框架进行二次开发的动态展示代码，包括被监听节点的展示，虚拟网络有流量变化时页面内行为动作的展示；大方美观的背景图片设计；使用JQuery技术，实现页面内其他操作操作的功能组件。JQuery是一个优秀的跨浏览器的JavaScript框架，提供了许多可用的页面组件，他是一个快速、简洁、轻量的JAVASCRIPT库，利用它我们可以方便的处理，展示页面上的documents、events属性时间，丰富页面的展示效果[[15]](#endnote-15)。

**第4章** 系统实现

## 系统的设计

系统的工作流程为：部署好虚拟网络，在作为虚拟路由的节点上部署好虚拟流量监听程序和处理程序；启动通信服务器模块；虚拟网络流量监听程序将处理后的数据发往通信服务器，由服务器进行存储、并加入到消息队列中；客户端通过浏览器向通信服务器发起websocket连接，服务端在与客户端建立连接后，开始从消息队列中逐条提取数据，并发送给客户端，客户端浏览器根据接受到的数据进行渲染和展示。

## 流量监听模块

本部分用于对指定的网卡进行流量的监听，使用libpcap进行实现。首先，确定我们的程序的运行环境环境如下：操作系统为Ubuntu 14.04.1 LTS；编程语言指定为Python；通过apt(Advanced Packaging Tool )软件安装python-pcapy；在程序编写过程中调用该库文件。

模块程序中需要加入代码：

import socket # 用于创建socket对象

from struct import \* # 字节流，组包拆包实现

import datetime # 关于日期

import pcapy # python中封装了libpcap库功能的接口

import sys # 提供系统相关功能

import requests # 提供发送http请求功能

该程序有四个功能函数组成，parse\_packet()函数接受一个数据包对象作为输入，返回一个字典对象，其内容包括数据包的类型、报文长度、源IP、目的IP、源MAC地址、目的MAC、数据包捕获的时间；eth\_addr()函数接受一个6位的十六进制表示的mac地址，并把它转换成标准格式返回；send\_to\_server()函数把parse\_packet的返回值作为参数，将其进行url编码，并且利用requests模块的相关功能接口，将编码后的内容发送到通信服务器；main()函数，作为该程序的入口，按顺序进行监听网卡、捕获数据包、解析数据包、转换相关内容的格式、发送其他函数捕获和处理之后的信息到通信服务器。

另一方面，使用系统命令” ifconfig eth0 promisc” 开启网卡的混杂模式，其中eth0是我们的目标监听网卡。至此，该程序可以正常监听虚拟网络中的流量， 并提取相关参数了。

## 4.3 通信服务器模块

### 4.3.1 环境配置

该模块的程序的开发和运行环境仍然是操作系统Ubuntu 14.04.1 LTS下。该模块的本质是一个Web应用程序，包括代理软件（Nginx）、wsgi/websocket、web应用程序、数据库。

### 4.3.2 数据库

数据库的功能有两部分。一部分是存储流量监听模块传递过来的相关数据，我们使用MongoDB数据库来实现该功能。首先，使用操作系统提供的软件包管理程序apt来进行MongoDB的安装，在操作系统的命令行中执行命令”apt-get install mongodb”，操作系统会自动安装程序和该程序所需要的相关依赖程序和库文件。在安装失败时，需要从一下几个方面考虑：当前用户是否有进行安装软件这一操作的权限；是否为网络连接断开或网络拥堵情况严重；是否有相关的依赖程序无法进行安装或者已经存在的依赖程序的版本与mongodb所需要的版本有冲突。考虑到安全性问题，mongodb数据库在安装的时候必须配置为使用账户和密码进行登录指定的数据库，虽然数据量很大，但是在系统进行工作的过程中，该数据库主要负责写入操作，所以不需要构建主从数据库。在系统的命令行中输入“mongo“指令，成功进入mongodb的指令操作界面，标明mongodb数据库的安装成功完成。在该操作界面中创建用户，和我们需要使用的一个数据库即可。

另一部分数据库需要提供的功能是消息队列功能，我们redis数据库。其安装的流程与Mongodb类似，不再赘述。由于Redis是基于内存访问的数据库，这也是Redis数据库的读写效率特别高的原因之一。我们需要设置它的内存分配策略，其配置文件位于”/proc/sys/vm/overcommit\_memory”，该部分的可选值为可选值：0、1、2：

0， 表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许；否则，内存申请失败，并把错误返回给应用进程。

1， 表示内核允许分配所有的物理内存，而不管当前的内存状态如何。

2， 表示内核允许分配超过所有物理内存和交换空间总和的内存。

redis在dump数据的时候，会fork出一个子进程，理论上child进程所占用的内存和parent是一样的，比如parent占用的内存为8G，这个时候也要同样分配8G的内存给child,如果内存无法负担，往往会造成redis服务器的down机或者IO负载过高，效率下降。所以这里比较优化的内存分配策略应该设置为 1（表示内核允许分配所有的物理内存，而不管当前的内存状态如何）。接着，开启访问redis数据库的端口，默认为6379，这里不作更改，仅修改系统的防火墙配置文件，进行如下三步操作：

1．使用vim编辑器打开配置文件：vi /etc/sysconfig/iptables 。

2．加入端口配置：-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 6379 -j ACCEPT 。

3．重新加载规则，在系统命令行中执行命令：service iptables restart 。

在系统的命令行中启动redis服务，输入“redis-server /etc/redis.conf “指令，成功执行后，再使用”redis-cli”命令，能进入redis的操作界面即可。

### 4.3.3 web应用程序

通信服务器模块中的web应用程序使用Python编程语言进行编写，使用开源的Tornado Framework进行二次开发。功能分为两部分，第一部分为作为服务端响应服务器接收到的http请求，用于接收流量监听模块发送过来的数据，并进行存储和其他处理；第二部分用于响应数据可视化模块向服务器发起的建立websocket连接的请求，并且在连接建立之后，从redis数据库提供的消息队列中获取消息，并且传输给数据可视化展示模块。

第一部分用于响应http请求的代码，需要使用利用Python操作redis与mongodb的两个第三方功能库，分别为reids和pymongo，在操作系统的命令中执行“pip install 包名（redis或pymongo）”命令进行安装。该部分的核心代码需要使用Tornado服务框架中的web包和ioloop包，其在程序中的使用方法如下面的代码所示：

该模块程序中，实现两个函数，函数save\_mongo()接受一条json数据作为参数，并把这条数据存储到mongodb数据库中，函数push\_redis()接受同样的数据并把该数据作为消息加入到redis数据库提供的消息队列中；新建一个继承自tornado.web.RequestHandler的类，名为MainHandler，其内部实现一个post方法，用于处理post请求，即流量监听模块处理后向通信服务器模块发送的请求。

import tornado.web # 提供web程序响应请求的示例

import tornado.ioloop # 提供操作网络io的相关方法

import redis # 操作redis数据库

import pymongo # 操作mongodb数据库

第二部分用于与数据可视化展示模块进行协同工作，其在服务端的程序需要实现两个类。类Task，实例化时返回一个redis的消息队列对象，包含从当前时间节点开始redis消息队列中的所有消息和后续被加入到消息队列中的消息，实现一个方法pop()，每次调用时则返回一条消息。类WebsocketHandler是继承自tornado.websocket.WebSocketHandler的，该类在每个客户端向服务器发起websocket连接时创建一个示例，实现四个方法，open()方法在websocket连接创建时执行，在客户端加载数据可视化展示所需要的静态文件和javascript代码；on\_message() 在接收到客户端发送的信息时执行，由于本系统不需要处理从客户端接收到的信息，所以保持默认操作即可；write\_message()方法，用于向客户端发送消息，在这里调用Task类，实现一个实例，一旦消息队列中有新的数据就把数据发往客户端，数据格式为json；on\_close()在客户端关闭页面和由于其他原因服务器或客户端断开连接时执行，可以不做任何操作，tornado框架会自动销毁此次连接中创建的相关对象，释放内存。

## 数据可视化展示模块

该模块主要包括的内容是静态html文件，图片，和需要加载的javascript代码。值得一提，此模块的程序代码也是放在通信服务器的web程序的目录下的，相关代码和文件，会在通信服务器响应客户端的建立websocket的请求时发给客户端，在客户端的浏览器进行展示和执行。其中的核心代码即为在客户端利用javascript建立websocket对象，其核心代码为：

//在客户端创建一个websocket对象

connect : function(){

\_this.ws = new WebSocket(\_this.socketAddress);

\_this.ws.onerror = \_this.error;

\_this.ws.onopen = \_this.onopen;

\_this.ws.onmessage = \_this.onmessage;

\_this.ws.onclose = \_this.onclose;

},

客户端在打开指定页面，加载完成静态文件，并成功执行javascript代码之后，就可以实时观察虚拟网络的流量展示情况。

## 4.5 系统功能与稳定性测试

### 4.5.1 测试环境和条件

本系统的测试，采取动态测试的方法。先进行单元测试，在根据使用多种客户端浏览器进行工作流测试。所需软件及相关参数如下图所示

表 4‑1实验环境

|  |  |
| --- | --- |
| 操作系统 | Linux Ubuntu Server(X64) |
| 处理器 | 2.4GHz×8 |
| 内存 | 8GHz |

表 4‑2实验软件

|  |  |
| --- | --- |
| 软件名 | 版本号 |
| APScheduler | v3.0.1 |
| Python | v2.7 |
| Tornado | v4.0.2 |
| Redis | v2.8.4 |
| MongoDB | v2.4.9 |
| Python-pcapy | v0.10.10 |
| Python-redis | v2.10.5 |
| pymongo | v3.2.1 |
| six | v1.8.0 |

测试环境的网络拓扑结构如图4-1所示：



其中，通信服务器为部署通信服务器模块和数据可视化展示模块程序的服务器，流量监听程序部署在虚拟路由器上，虚拟路由器通过物理网卡桥接连到虚拟网络进行流量监听和相关参数的提取；通信服务器，虚拟路由器均处于内网中，用户通过代理服务器对可视化页面进行访问，代理服务器使用Squid作为web服务代理，同时在Squid中设置防火墙规则iptables来控制访问以保证通信服务器以及虚拟网络的安全。[[16]](#endnote-16)

### 4.5.2 评价方法

系统共进行五项测试，分别是可部署性测试、并发性测试、稳定性测试、兼容性测试、功能完整性测试。这五项测试分别对应五个评价指标：系统部署耗时(t)、平均响应时间（）、持续稳定运行时间、支持浏览器种类数、系统功能完成度。

### 4.5.3 测试过程

1. 系统可部署性测试

在系统功能测试和性能测试之前首先进行系统部署，在系统部署时记录开始部署系统的时间和完成系统部署的时间，从而得到系统部署耗时。

1. 系统并发性测试

使用Microsoft 的Web Application Stress Tool（WAS）这个工具软件进行并发性测试，WAS能有效测试一个网站的负载性能，这个软件可以通过脚本模拟数百个强并发用户的访问，并模拟实际用户的一些点击操作。实验使用不同的并发用户数进行了8次测试，每次测试设置的并发用户数分别为100、200、300、400、500、600、650、700，测试中记录每一次的响应时间并计算平均响应时间，同时记录CPU和内存的占用率。

1. 系统可靠性测试

实验期间连续运行系统，记录系统稳定运行的最长时间。

1. 系统兼容性测试

在电脑上安装几类主流的浏览器，分别通过这些浏览器访问系统Web服务器，并在系统界面上进行操作来测试系统的兼容性。测试的浏览器包括IE、FireFox、Google Chrome、Sogou Explore、Safari、360chrome、360SE、Baidu Explore。

1. 系统功能完整性测试

访问系统主页，对系统设计的功能逐项进行测试，记录每项功能是否达到系统设计要求。

### 4.5.4 测试结果

1. 系统可部署性测试

整个系统部署的过程耗时2小时36分，结果表明系统可以在3小时内完成部署，具备高可部署性，测试结果详情如表 4-3所示。

表 4-3系统部署时间记录

|  |  |  |
| --- | --- | --- |
| 序号 | 事件 | 时间点 |
| 1 | 开始部署系统 | 2016年5月10日18:36 |
| 2 | 完成系统必备软件安装 | 2016年5月10日19:09 |
| 3 | 完成系统安装 | 2016年5月10日19:30 |
| 4 | 完成虚拟网络部署 | 2016年5月10日20:03 |
| 5 | 完成代理服务器设置 | 2016年5月10日20:26 |
| 6 | 完成通信服务器部署 | 2016年5月10日20:50 |
| 7 | 完成流量监听程序 | 2016年5月10日21:12 |

1. 系统并发性测试

测试结果表明，随着并发用户数的增加，平均响应时间、CPU占用率、内存占用率都会上升。当并发数在500以内时，平均响应时间在1s内，CPU占用率和内存占用率均保持在50%以内。并发数超过500后，平均响应时间、CPU占用率和内存占用率迅速上升。并发数达到700时，平均响应时间超过3s，CPU占用率超过75%，内存占用率超过70%。当响应时间在1s以内时，用户体验最好。实验结果表明系统满足支持超过500个用户并行操作的性能要求，详细测试结果如表 4-4所示。

表 4-4并发性测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 并发用户数 | 平均响应时间 | CPU占用率 | 内存占用率 |
| 1 | 100 | 0.26s | 12% | 17% |
| 2 | 200 | 0.31s | 19% | 23% |
| 3 | 300 | 0.45s | 28% | 29% |
| 4 | 400 | 0.68s | 39% | 38% |
| 5 | 500 | 0.96s | 50% | 47% |
| 6 | 600 | 1.51s | 61% | 56% |
| 7 | 650 | 2.43s | 73% | 63% |
| 8 | 700 | 3.78s | 82% | 72% |

1. 系统可靠性测试

系统持续稳定运行时间超过15天。

1. 系统兼容性测试

测试结果表明，系统兼容测试的8类浏览器，满足兼容3类以上浏览器的性能要求，详细测试结果如表 4-5所示。

表 4-5兼容性测试结果

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 浏览器名称 | 浏览器内核 | 是否兼容 |
| 1 | IE | IE | 兼容 |
| 2 | FireFox | Gecko | 兼容 |
| 3 | Google Chrome | Webkit | 兼容 |
| 4 | Sogou Explorer | IE | 兼容 |
| 5 | Safari | AppleWebKit | 兼容 |
| 6 | 360chrome | IE | 兼容 |
| 7 | 360SE | IE | 兼容 |
| 8 | Baidu Explore | IE | 兼容 |

1. 系统功能完整性测试

实验结果表明，系统设计的功能均已实现，系统功能完整性为100%，测试详情如图4-2所示。

**此处待插入效果图**

## 4.6 小结

**第5章** 结束语

## 5.1 全文总结

## 5.2 工作展望

1. [] Lnux下的高流量数据包监听技术 [↑](#endnote-ref-1)
2. 基于WebSocket的实时Web应用解决方案\_温照松 [↑](#endnote-ref-2)
3. W3C. TheWebSocketAPI [EB/OL]. (2011.9.29) [213.10] https://www.w3.org/TR/2011/WD-websockets-20110929/ [↑](#endnote-ref-3)
4. 基于WebSocket的网络实时通信\_薛陇彬 [↑](#endnote-ref-4)
5. Yuan XR. Big Data visualization. IEEE Computer Graphics and Applications, 2013,33(4):20-21. [doi:10.1109/MCG.2013.54] [↑](#endnote-ref-5)
6. 柳杨,范冰冰. 一种通用的动态Web应用加速方案[J]. 计算机工程,2012,38(11):247-250. [↑](#endnote-ref-6)
7. 田红玉. 静态网页制作技术与动态网页制作技术的比较[J]. 黑龙江科学,2013,4(9):116. [↑](#endnote-ref-7)
8. 刘祺彬. 基于开源技术的MVC框架的构建[J]. 信息技术,2013,(8):125-129. [↑](#endnote-ref-8)
9. 张宇,王映辉,张翔南等. 基于Spring的MVC框架设计与实现[J]. 计算机工程,2010,36(4):59-62. [↑](#endnote-ref-9)
10. 赵玉伟,肖敏,熊前兴等. 一种简单MVC框架的实现方案[J]. 交通与计算机,2006,24(4):136-138. [↑](#endnote-ref-10)
11. 刘亚鹏. 基于MVC架构的电子商务平台的设计与实现[D]. 华中科技大学,2011. [↑](#endnote-ref-11)
12. 于海雯,娄芳. 用MVC处理Web系统的逻辑编程和页面设计的分离问题[J]. 电脑知识与技术（认证考试）,2004,(8):69-71. [↑](#endnote-ref-12)
13. Libpcap 数据包捕获机制剖析与研究 江苏信息职业技术学院 平震宇 [↑](#endnote-ref-13)
14. MongoDB数据库的应用研究和方案优化\_王光磊 [↑](#endnote-ref-14)
15. 基于JQuery框架的Web查询视图设计与实现\_杨旭士 [↑](#endnote-ref-15)
16. 周奇 李震阳. LiNuX 系统网络服务器组建、配置和管理实训教程.[M]. [↑](#endnote-ref-16)