1. 项目背景

上海教育宽带网以各高校和17个区县教育网为主，主干构架在教育专用光缆之上，拥有自主管理权，可以根据设备不同、应用需求的变化、以及发展的需要，方便灵活地进行提升带宽。

一方面，上海教育宽带网使用静态路由的方式以20Gbps的带宽实现与中国教育科研网的互通；通过2Gbps带宽实现与上海交互中心互通，1Gbps带宽与上海电信运营商互联，1Gbps带宽与上海移动互联，1Gbps带宽与东方有线互联。

另一方面，宽带网上目前开展了各类应用，如MPLS VPN，视频会议，跨校认证，无线通等，不同的业务对网络的需求目标并不一样，有些业务需要低延迟，有些业务需要高带宽，宽带网的性能是否足够是支撑各种业务是否能够顺利展开的前提条件。

网络性能测量矩阵平台，可以实现对全网的延迟、抖动、QoS、可用带宽等进行测量，依照平台的部署点构建测量矩阵，以便运维人员和宽带网用户及时方便的了解网络性能，为提高业务的服务质量提供相应的依据。

1. 项目总体目标
2. 平台能支持多种测量方式，一方面可以通过主动向网络中注入探测数据包，并收集、分析探测包在网络中传输后发现的状态变化，从而计算出要测量的网络性能参数。具体对带宽、拥塞、延迟、抖动、丢包、可用性等进行测量，能够支持ICMP、TCP、UDP等多种方式完成测量；另一方面可以在测量点上部署相应的探针/采集器，使用SNMP对网络信息进行收集，从而分析出网络的性能参数，具体包括往返时延、端到端丢包、带宽、吞吐量、CPU及内存等。
3. 平台提供网络性能测试，在每个点部署三种大小的文件供用户进行TCP下载测试（基于html5）；同时提供三种大小文件上传测试；
4. 平台能够结合被动与主动测量，形成各节点间的的性能测量矩阵；如基于ICMP/TCP/UDP的时延、抖动、丢包、可用性、可用带宽等多指标矩阵图；
5. 平台可以支持分布式部署，可以按照规模发展进行灵活的部署和撤销部署。
6. 平台支持中文界面；需提供可编程接口进行配置；支持可读性良好的故障报告和趋势报告；
7. 项目建设特点

3.1技术方案概述

网络性能测量矩阵平台，可以实现对全网的延迟、抖动、QoS、可用带宽等进行测量，用户众多，数据量大 ，所以我们在技术上实现如下目标：

* 1. 平台能够可持续稳定的运行，能够支持大量并发的访问，能够处理海量数据。
  2. 平台支持多个节点分布式部署，充分考虑节点与节点之间的关系，对节点之间的网络状况进行高效测量。
  3. 系统支持统一身份认证借口。
  4. 系统完全采用模块化设计，做到模块清晰，借口开放，利于进行二次开发和系统扩展。
  5. 界面支持中文，友好简便，操作人性化。

3.2技术框架

网络性能测量矩阵平台是基于不同平台，不同系统和不同运行环境的测量平台。我们建议采用C++开发运行框架，以便于在Windows、Linux、Unix和Solaris多种操作系统中进行系统的部署。

3.3技术特点

1. 数据集中管理 各个节点之间的数据逻辑关系清晰，预先考所有数据标准的兼容，以及与其他节点之间的数据共享，以便实现数据的及时性和一致性；
2. 高度模块化 平台由多个子系统组成，子系统相互联系又有很大的独立性，方便管理员有选择有步骤的实施项目；
3. 高度参数化 平台中的很多参数和指标都是由用户自定义，方便各个节点和平台随时根据网络状态的变化来调整系统参数。
4. 易用性 平台操作流程清晰，操作简单，用户不需要进行复杂的业务培训就可以轻松的使用系统；
5. 跨平台性 平台采用C++开发技术和多层应用体系结构，因此可以运行于多种操作系统平台上；
6. 高效缓存 为提高平台运行速度，降低时延和开销，加强多用户测试时平台的反应能力，我们采用了灵活的缓存技术，有效提高系统的运行性能问题。

3.4平台安全问题

网络性能测量矩阵平台的安全问题可以分为运行环境的安全问题和平台自身的安全问题。

运行环境的安全问题主要是指网络性能测量矩阵平台所部属的服务器的安全问题。服务器需要放置计算机病毒和黑客入侵，数据库需要做定期备份以防数据丢失。

平台自身的安全问题主要是平台中不同用户具有不同的管理权限，放置越权操作行为的发生，以免引起数据和管理上的混乱。我们在网络性能测量矩阵平台中采用高的是ACL权限控制方式，具有很大的灵活性。各个节点的管理员可以根据节点的实际情况灵活的设计平台使用过程中的安全策略。

3.5平台整合能力

在网络信息化日益重要的时代，构建系统时需要考虑到第三方的系统兼容问题。以避免信息孤岛的产生。由于网络性能测量矩阵平台是部署在上海教育网的节点上的，需要将各个节点，用户的带宽、拥塞、延迟、抖动、丢包、可用性信息进行整合，保证不同系统在数据上保持同步，从而实现系统简单的数据共享。

1. 网络测量平台方案

4.1平台概述

网络性能测量矩阵平台从结构上可以分成测量模块，处理模块，数据模块和系统模块。测量模块能够自主的选择进行主动测量，被动测量或者上传下载测试，同时可以对参数进行配置，支持ICMP、TCP、UDP等多种方式测量网络的带宽、拥塞、延迟、抖动、丢包和可用性。处理模块则是向网络中注入数据包和收取数据包，对收集到的数据进行处理分析得到结果图，并且可以根据一段时间的数据进一步的生成故障报告和趋势报告。数据模块用于存放缓存数据包，用于测试上传下载的文件以及SNMP等各类文件，用于被动测量，带宽测试以及生成故障报告和趋势报告。系统服务提供了用户、权限、参数设置以及安全方面的功能。其示意图如下：

**前端模块**

**登录界面**

**测量界面**

**测量模块**

**主动测量（ICMP、TCP、UDP）**

**被动测量（SNMP）**

**带宽**

**拥塞**

**延迟**

**抖动**

**丢包**

**其他**

**处理模块**

**测量矩阵**

**生成图表**

**故障报告**

**趋势报告**

**数据模块**

**测试文件**

**缓存数据**

**S**

**NMP**

**用户数据**

**系统服务**

**用户管理**

**角色权限**

**数据管理**

**参数设置**

图1

4.2系统设计

本系统是一个部署在多个测量点上的分布式系统，服务于上海教育宽带网以各高校和17个区县教育网。系统采用B/S架构，用户可以再不同的系统上通过浏览器访问实现各种功能。根据每一个用户权限的不同，可以提供不同的网络测量服务，其示意图如下：

图2

**管理员**：可以向服务器发送主动测量和被动测量请求，对整个网络中任意节点之间的带宽，延迟，抖动等指标进行测量，并按照生成网络测量矩阵、故障报告、趋势报告，可以对测量指标进行配置；

**普通用户：**可以通过上传下载文件测试确定自身所处的网络到目标节点的网络状况，生成网络测量矩阵、故障报告等。

4.3系统功能

4.3.1测量模块

测量模块我们采用CSS语言编写网络前端的网站界面，用户可以通过各种浏览器方便的访问服务器获取各种网络测量服务。

同时，我们采用C++语言来编写后台程序部署在服务器上，根据用户的命令进行向网络中注入或者提取数据包，分析数据，提交结果等操作，以下是程序中重要的几个类。

Thread类是网络测量矩阵平台中十分重要的一个类，他把POSIX标准中的多线程机智封装起来，为系统提供了一个简单易用的线程模型，是系统能够实现多线程并行操作的核心。

class Thread {

public:

Thread(void);

virtual ~Thread();

// start or stop a thread executing

void Start(void);

void Stop(void);

virtual void Run(void) = 0;

// wait for this or all threads to complete

void Join(void);

static void Joinall(void);

void DeleteSelfAfterRun(void) {

mDeleteSelf = true;

}

// set a thread to be daemon, so joinall won't wait on it

void SetDaemon(void);

// returns the number of user (i.e. not daemon) threads

static int NumUserThreads(void) {

return sNum;

}

static nthread\_t GetID(void);

static bool EqualID(nthread\_t inLeft, nthread\_t inRight);

static nthread\_t ZeroID(void);

protected:

nthread\_t mTID;

bool mDeleteSelf;

// count of threads; used in joinall

static int sNum;

static Condition sNum\_cond;

private:

static void\* Run\_Wrapper(void\* paramPtr);

}; // end class Thread

void Thread::Start(void) {

if (EqualID(mTID, ZeroID())) {

// increment thread count

sNum\_cond.Lock();

sNum++;

sNum\_cond.Unlock();

Thread\* ptr = this;

// pthreads -- spawn new thread

int err = pthread\_create(&mTID, NULL, Run\_Wrapper, ptr);

FAIL(err != 0, "pthread\_create");

}

}

void Thread::Stop(void) {

if (!EqualID(mTID, ZeroID())) {

// decrement thread count

sNum\_cond.Lock();

sNum--;

sNum\_cond.Signal();

sNum\_cond.Unlock();

nthread\_t oldTID = mTID;

mTID = ZeroID();

// exit thread

// use exit() if called from within this thread

// use cancel() if called from a different thread

if (EqualID(pthread\_self(), oldTID)) {

pthread\_exit(NULL);

}

else {

// Cray J90 doesn't have pthread\_cancel; Iperf works okay without

pthread\_cancel(oldTID);

}

}

}

void \*Thread::Run\_Wrapper(void\* paramPtr) {

assert(paramPtr != NULL);

Thread\* objectPtr = (Thread\*)paramPtr;

// run (pure virtual function)

objectPtr->Run();

#ifdef HAVE\_POSIX\_THREAD

// detach Thread. If someone already joined it will not do anything

// If noone has then it will free resources upon return from this

// function (Run\_Wrapper)

pthread\_detach(objectPtr->mTID);

#endif

// set TID to zero, then delete it

// the zero TID causes Stop() in the destructor not to do anything

objectPtr->mTID = ZeroID();

if (objectPtr->mDeleteSelf) {

DELETE\_PTR(objectPtr);

}

// decrement thread count and send condition signal

// do this after the object is destroyed, otherwise NT complains

sNum\_cond.Lock();

sNum--;

sNum\_cond.Signal();

sNum\_cond.Unlock();

return NULL;

}

void Thread::Joinall(void) {

sNum\_cond.Lock();

while (sNum > 0) {

sNum\_cond.Wait();

}

sNum\_cond.Unlock();

}

void Thread::SetDaemon(void) {

sNum\_cond.Lock();

sNum--;

sNum\_cond.Signal();

sNum\_cond.Unlock();

}

Socket 类的主要功能是封装了socket文件描述符、此socket对应的端口号，以及socket接口中的listen, accept, connect和close等函数，为用户提供了一个简单易用而又统一的接口。同时作为其他派生类的基类。

#ifndef SOCKET\_H

#define SOCKET\_H

#include "headers.h"

#include "SocketAddr.hpp"

/\* ------------------------------------------------------------------- \*/

class Socket {

public:

// stores server port and TCP/UDP mode

Socket(unsigned short inPort, bool inUDP = false);

// destructor

virtual ~Socket();

protected:

// get local address

SocketAddr getLocalAddress(void);

// get remote address

SocketAddr getRemoteAddress(void);

// server bind and listen

void Listen(const char \*inLocalhost = NULL, bool isIPv6 = false);

// server accept

int Accept(void);

// client connect

void Connect(const char \*inHostname, const char \*inLocalhost = NULL);

// close the socket

void Close(void);

// to put setsockopt calls before the listen() and connect() calls

virtual void SetSocketOptions(void) {

}

// join the multicast group

void McastJoin(SocketAddr &inAddr);

// set the multicast ttl

void McastSetTTL(int val, SocketAddr &inAddr);

int mSock; // socket file descriptor (sockfd)

unsigned short mPort; // port to listen to

bool mUDP; // true for UDP, false for TCP

};

void Socket::Listen(const char \*inLocalhost, bool isIPv6) {

int rc;

SocketAddr serverAddr(inLocalhost, mPort, isIPv6);

// create an internet TCP socket

int type = (mUDP ? SOCK\_DGRAM : SOCK\_STREAM);

int domain = (serverAddr.isIPv6() ?

#ifdef IPV6

AF\_INET6

#else

AF\_INET

#endif

: AF\_INET);

mSock = socket(domain, type, 0);

FAIL\_errno(mSock == INVALID\_SOCKET, "socket");

SetSocketOptions();

// reuse the address, so we can run if a former server was killed off

int boolean = 1;

Socklen\_t len = sizeof(boolean);

// this (char\*) cast is for old headers that don't use (void\*)

setsockopt(mSock, SOL\_SOCKET, SO\_REUSEADDR, (char\*)&boolean, len);

// bind socket to server address

rc = bind(mSock, serverAddr.get\_sockaddr(), serverAddr.get\_sizeof\_sockaddr());

FAIL\_errno(rc == SOCKET\_ERROR, "bind");

// listen for connections (TCP only).

// default backlog traditionally 5

if (!mUDP) {

rc = listen(mSock, 5);

FAIL\_errno(rc == SOCKET\_ERROR, "listen");

}

}

int Socket::Accept(void) {

iperf\_sockaddr clientAddr;

Socklen\_t addrLen;

int connectedSock;

while (true) {

// accept a connection

addrLen = sizeof(clientAddr);

connectedSock = accept(mSock, (struct sockaddr\*) &clientAddr, &addrLen);

// handle accept being interupted

if (connectedSock == INVALID\_SOCKET && errno == EINTR) {

continue;

}

return connectedSock;

}

}

void Socket::Connect(const char \*inHostname, const char \*inLocalhost) {

int rc;

SocketAddr serverAddr(inHostname, mPort);

assert(inHostname != NULL);

// create an internet socket

int type = (mUDP ? SOCK\_DGRAM : SOCK\_STREAM);

int domain = (serverAddr.isIPv6() ?

#ifdef IPV6

AF\_INET6

#else

AF\_INET

#endif

: AF\_INET);

mSock = socket(domain, type, 0);

FAIL\_errno(mSock == INVALID\_SOCKET, "socket");

SetSocketOptions();

if (inLocalhost != NULL) {

SocketAddr localAddr(inLocalhost);

// bind socket to local address

rc = bind(mSock, localAddr.get\_sockaddr(), localAddr.get\_sizeof\_sockaddr());

FAIL\_errno(rc == SOCKET\_ERROR, "bind");

}

// connect socket

rc = connect(mSock, serverAddr.get\_sockaddr(), serverAddr.get\_sizeof\_sockaddr());

FAIL\_errno(rc == SOCKET\_ERROR, "connect");

}

主动测量方面，我们通过主动向网络中注入探测数据包，并收集、分析探测包在网络中传输后发现的状态变化，从而计算出要测量的网络性能参数。具体对带宽、拥塞、延迟、抖动、丢包、可用性等进行测量，需要支持ICMP、TCP、UDP等多种方式完成测量。

测试TCP带宽的原理是两个服务器端建立连接，然后发送端发送一定大小的数据，并记下发送的时间， 或者在一定的时间内发送数据，并记下发送的总数据。带宽的大小等于发送的总数据除以发送的总时间。对另一端来说，就是在连接建立时间内，接收的总数据除以所花时间即为这一端端所测得的带宽。MSS的大小通过TCP内核接口函数直接获得。



图3

测试UDP的性能，一端可以指定UDP数据流的速率。发送端发送数据时将根据客户提供的速率计算数据报发送之间的时延。另外客户还可以指定发送数据报的大小。每个发送的数据报包含一个ID号，用来惟一的标识该报文。接收端则根据该ID 号来确定数据报丢失和乱序。当把UDP报文大小设置可以将整个报文放入I P 层的包(packet)内时，那么UDP所测得的报文丢失数据即为IP层包的丢失数据。这提供了一个有效的测试包丢失情况的方法。数据报传输延迟抖动(Jitter)的测试由接收器端完成，客户发送的报文数据包含有发送时间戳，服接收端根据该时间信息和接收到报文的时间戳来计算传输延迟抖动。传输延迟抖动反映传输过程中是否平滑。由于它是一个相对值，所以并不需要客户端和服务器端时间同步。



图4

ICMP协议的优先级较低，所以测得的数据有一定的局限性，时间进度只能在毫秒级。源主机可以通过发送“回显请求”报文等待目标主机回应来测试目的主机的可达性，同时可以根据发送和应答的时间间隔测量网络的双向时延。 同时，ICMP还可以以类似于TCP和UDP的方式或者与TCP和UDP相结合测试网络的带宽，拥塞和丢包率。



图5

被动测量方面，需要在测量点上部署相应的探针/采集器，使用SNMP对网络信息进行收集，从而分析出网络的性能参数。具体包括往返时延、端到端丢包、带宽、吞吐量、CPU及内存等。



图6

4.3.2数据处理模块

结合被动与主动测量，形成各节点间的的性能测量矩阵；如基于ICMP/TCP/UDP的时延、抖动、丢包、可用性、可用带宽等多指标矩阵图。

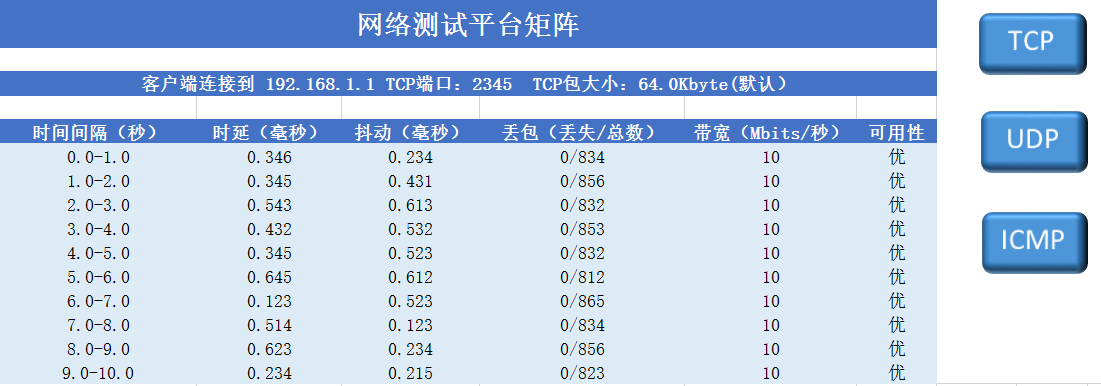


图7

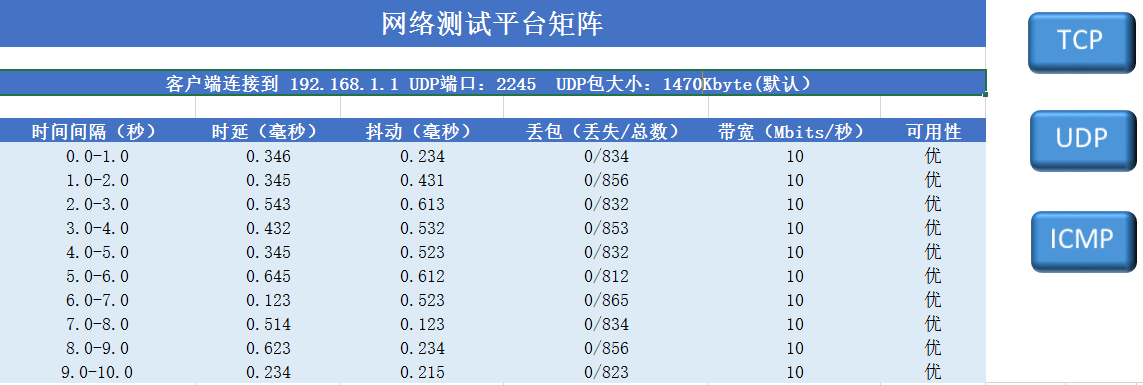


图8



图9



图10

4.3.3数据模块

我们给所有的用户提供到制定端点的上传下载测试服务。可以提供3种不同大小的文件下载以供测试。



图11



图12

4.3.4系统模块

我们提供了用户身份验证，可以针对不同权限的用户提供不同种类的网络测试服务。



图13

同时，当用户测试遇到故障需要反馈时，可以通过故障报告快速的将自身所遇到的状况报告给指定端口的管理员，管理员将实时的接收到这些信息并针对性地进行处理。



图14

当用户想知道一段时间内的网络状况时候，还可以选择生成趋势报告，如下图所示。

图15

图16

1. 项目实施建议

5.1需求分析

鉴于本公司还没用对网络测量矩阵平台的需求进行深入了解，对各个节点之间的处理谢协同方面也没有机会进行深入的研究。这些都需要我们精心深入的分析和调查研究。本公司将派专门人员向相关部门了解细致的需求和需求相关的情况，服务器管理员的意见等。

5.2系统概要设计

根据现有的建设和规划的布局，勾画出各流程在网络中的运行过程。编制《概要设计文档》并交付，并根据反馈意见对文档进行修改和最终确认。

5.3详细设计

在开发环境和部署规定已经确定的情况下，根据概要设计的内容，对系统各种逻辑实现进行详细设计。根据经双方确认的《概要设计说明书》、《需求说明书》和《详细》设计，设计个表单、视图等格式。

本公司项目开发组编制《详细设计》并上交给项目管理组。

5.4编码和单元测试

* 1. 编制所有模块的程序代码：根据经双方确认的《详细设计》编制所有的模块的程序代码，本公司项目开发组定期告知上海交通大学项目管理组编码的进展情况。
  2. 进行单元测试。

5.5综合测试

1. 制定测试计划：本公司项目开发组将于计划的测试时间前一星期向项目管理组递交《产品测试计划》，以利于委托方准备测试案例和反馈意见。
2. 准备测试样例：为了测试的准确性本公司项目开发组会要求管理组提供真实的案例。如果资料涉及文件安全问题，本公司简易双方签订文件保密协议以解决问题。
3. 功能测试：功能测试将验证系统是否实现了用户的功能需求目标，本公司的项目开发组根据《需求说明书》和涉及文档编制测试案例，进行功能测试。为了产品功能测试更有效，本公司项目开发组要求项目管理委员会派终端用户配合本公司项目开发组一起进行功能测试。
4. 系统测试：系统测试是检测系统性能、承受压力情况等。本公司项目开发组根据《需求说明书》和涉及文档编制测试案例、进行系统测试。为了产品系统测试更有效，本公司项目开发组要求项目管理委员会派系统管理员配合本公司项目开发组一起进行系统测试。

5.6编制用户文档

1. 编制《用户手册》，本公司项目开发组在产品交付使用前向项目管理组递交《用户手册》草稿，终端用户如有不明之处可在两天内向本公司项目开发组提出，正式的《用户手册》将随产品一起交付。
2. 编制《管理员手册》，本公司项目开发组在产品交付使用前向项目管理组递交《管理员手册》草稿，管理员如有不明之处可在两天内向本公司项目开发组提出，正式的《管理员手册》将随产品一起交付。

5.7安装和培训

1. 培训：本公司项目开发组提供一下两种以《用户手册》和《管理员手册》为主要教材的培训：
2. 终端用户培训：目标在于使终端用户掌握系统的各项功能操作，大概了解系统整个工作流程。
3. 管理员培训：管理员培训的目标在于使管理员详细掌握系统工作流程，系统管理操作。