

Project Specifications for IEG4180 Network Programming and System Design

Department of Information Engineering

The Chinese University of Hong Kong

September 2009

Introduction

This document provides detail specifications of the course projects in IEG4180. The course projects are designed to strengthen students' understanding of concepts and theories introduced in the lectures, and to gain hands-on experiences on network software design and programming.

Marks for the projects are distributed as follows:

Project #1 – NetProbe Console Edition	10%	(Individual)
Project #2 – NetProbe Client-Server Edition	10%	(Individual)
Project #3 – SuperNetProbe and JavaNetProbe	10%	(Team of 2)
Project #4 – On-Demand Video Streaming System	10%	(Team of 2)
Total:	40%	

Tools and Platform

The following tools and platforms are to be used in the projects:

Tool/Platform	Name
Operating System	Microsoft Windows and Java (J2SE 5.0 or above)
Network API	Winsock 2 and Java (J2SE 5.0 or above)
Language	C/C++ and Java
Development Tools	Microsoft Visual C++ and Netbeans IDE (or equivalent Java IDE)
Protocol Stack	TCP/IP

Attention is drawn to University policy and regulations on honesty in academic work, and to the disciplinary guidelines and procedures applicable to breaches of such policy and regulations. Details may be found at <http://www.cuhk.edu.hk/policy/academichonesty/>. With each assignment, students will be required to submit a statement that they are aware of these policies, regulations, guidelines and procedures.

Project 1 – NetProbe Console Edition (10%)

A. Overview

In this project, the students will develop a simple software for network testing and benchmarking. Key concepts applied in this project include: winsock programming basics, blocking I/O, socket configuration and control, simple protocol design and implementation. The NetProbe is a very useful tool for diagnosing network problems, as well a good tool for benchmarking performance of various programming models. This is an individual project.

B. Specification

The NetProbe tool can be operated in three modes, namely Send Mode, Receive Mode, and HostInfo Mode. Both the send and the receive mode are to be implemented using blocking I/O with multi-threading. The HostInfo mode is also to be implemented using blocking mode. Use of thread is optional.

The command line arguments are specified below.

Specification of Command-line Arguments:

NetProbe [mode] <more parameters depended on mode, see below>

[mode] : “s” means sending mode; “r” means receiving mode ;

“h” means host information mode. In this mode, NetProbe will try to resolve and find out all the IP addresses and names for a given hostname.

If [mode] = “s” then

NetProbe [mode] [refresh_interval] [remote_host] [remote_port] [protocol] [packet_size] [rate] [num]

else if [mode] = “r” then

NetProbe [mode] [refresh_interval] [local_host] [local_port] [protocol] [packet_size]

else if [mode] = “h” then

NetProbe [mode] [hostname]

end if

Parameter Definitions:

[refresh interval] : The refreshing interval of statistics display (in ms).

[remote_host | local_host] : Hostname or IP address for remote | local host.

[remote_port | local_port] : Port number for remote | local host.

[protocol] : “tcp” means to use TCP; “udp” means to use UDP.

[packet_size] : Size in bytes of a datagram (UDP), or size of I/O transaction (TCP).

[rate] : The rate (in bytes per second) at which to transmit data (sending mode only). ‘0’ means send as fast as the transport allows.

[num] : Total number of packets to send before disconnecting and terminate. ‘0’ means send forever until break by the user.

[hostname] : Name of host to lookup, assume local host if not specified.

C. Experiments

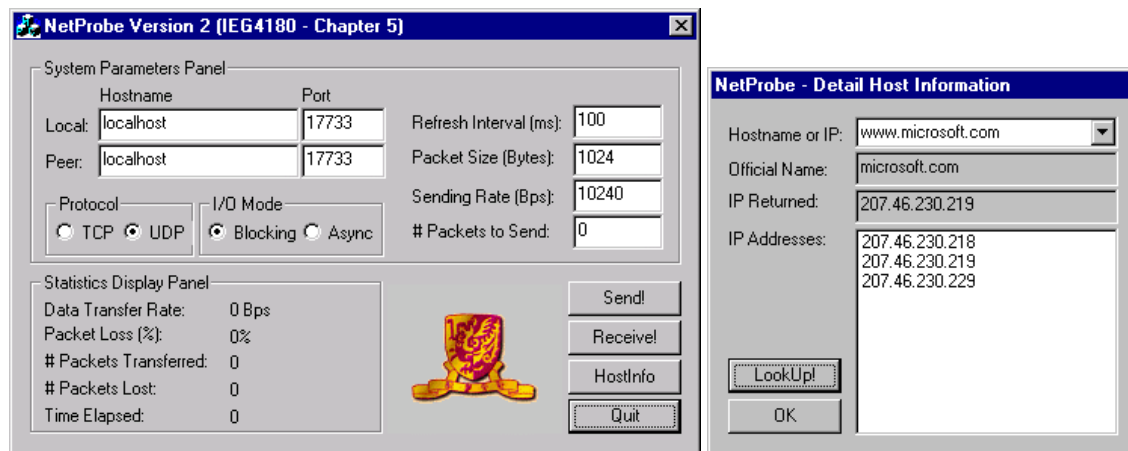
1. Measure the maximum transmission rate and CPU utilization versus packet size (2^n bytes, $n=6, 8, 10, 12, 13, 14, 15$) using the TCP protocol.
2. Repeat (1) using the UDP protocol. Plot also the packet loss rate versus packet size.

Project 2 – NetProbe Client-Server Edition (10%)

A. Overview and Specifications

In this project, we extend the NetProbe application developed in Project #1 into two applications, namely a client and a server application.

- (a) The NetProbe Server will listen for incoming connections originating from a NetProbe Client, receiving the operating parameters (e.g., packet size, transmission rate, etc.) and then begin transmitting data to the NetProbe Client. Additionally, the NetProbe Server will be extended to become a **concurrent server** and thus, will be able to establish more than one NetProbe Sessions with multiple NetProbe Clients. The concurrent server model is to be implemented using **multi-threading**. No GUI is required for the NetProbe Server.
- (b) The NetProbe Client will be re-implemented with a **graphical user interface** (GUI) for parameter configuration and statistics display, such as the example below. In addition, the NetProbe Client will also implement the **message-driven I/O model** for receiving data in addition to the threaded blocking I/O model used in Project #1.



B. Experiments

1. Compare the achievable throughput, loss, and CPU utilization under message-driven I/O with multithreaded blocking I/O.
2. Compare the achievable throughput versus the number of client sessions, under both TCP and UDP.
3. (Optional) Test the achievable throughput over a wireless network (e.g., Wi-Fi), under both TCP and UDP.

Project 3 – SuperNetProbe and JavaNetProbe (10%)

A. Overview and Specifications

Project 3 is divided into three parts: Web Console for NetProbe Server, SuperNetProbe and JavaNetProbe. This is a team project with two students per group.

The NetProbe Server will be extended to support a web-based console for controlling and monitoring its operations. The web console supports two functions. First, a user can use a web browser to connect to the NetProbe Server using HTTP, and then the NetProbe Server will return a web page (a form) for the user to configure the transmission parameters. Once the user press a start button on the returned web page the parameters will be sent to the NetProbe Server via HTTP and then the NetProbe Server will begin transmission accordingly. Second, a user can use a web browser to connect to the NetProbe Server using HTTP, and then the NetProbe Server will return a web page showing the latest transmission statistics (e.g., number of concurrent connections, statistics for each connection such as client IP address/hostname, transmission rate, packet size, bytes transmitted, time elapsed, etc.).

SuperNetProbe extends the NetProbe Client from Project #2 to implement a new I/O model - **alertable overlapped I/O**, for receiving data. JavaNetProbe is a reimplementation of the NetProbe Client using **Java**. Network I/O is to be implemented using the New I/O (NIO) API. This Java version should employ the *same protocol* as used in Project #2 so that JavaNetProbe can *interoperate* with the Winsock version of the NetProbe Server.

B. Experiments

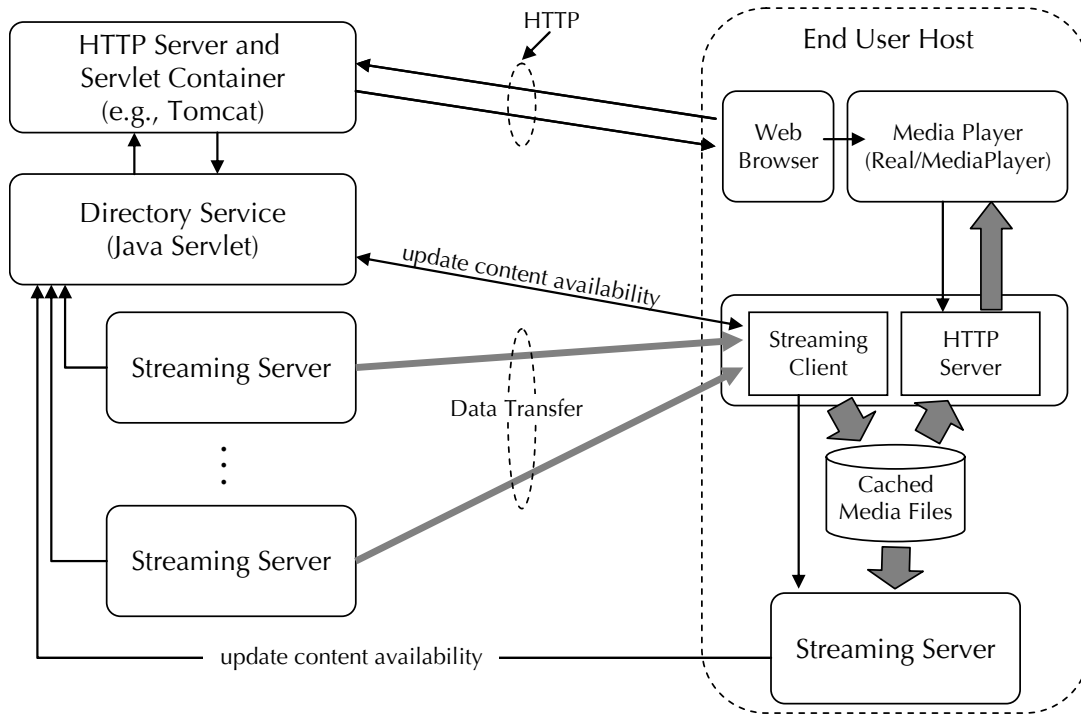
1. Study performance of the Java implementation and compares it to the winsock version.
2. Study network I/O performance under different platforms (e.g. Windows and Linux).
3. Test your ISP's bandwidth, loss rate, etc.

Project #4 – Distributed Video Streaming System (10%)

A. Overview

This project expands upon Project #3 into a system to stream video contents in a distributed manner. This is a team project with two students per group.

There are three components in the system, namely the Streaming Server, the Client, and the Directory Service.



B. System Components

The Streaming Server and Streaming Client support multi-source video streaming. Specifically, the client can request data of the same media stream from multiple servers. Each server will send only the requested portions of the media data to the client, thus the workload is divided among the servers. The client upon receiving data from the servers will need to resequence the data into proper order before passing to the media player for playback. Note that the user should be able to configure the maximum connection/bitrate at which a Streaming Server can accept concurrently to control resource utilization and to avoid congesting the access network.

After playback the received media data will be cached by the Streaming Server. The user can specify a maximum cache size so that when the cache is full, some of the old media data will be deleted to make room for new media data. The Streaming Server will keep the Directory Service updated of its cached contents (i.e., file ID and range of bytes cached) as well as its available streaming capacity using a control protocol.

The Directory Service is a new component to be developed as a Java Servlet. Its function is to collect media file availability information from all running Streaming Servers to build a central directory for content search and location purposes. A user will use a web browser to connect to the Directory Service, which then presents a web-based interface to the user for locating and selecting media files for playback. When a user select a media file for playback (e.g., using HTTP Post) the Directory Service will redirect the browser to an URL that triggers the media player to begin HTTP streaming through the Streaming Client.

Note that if the same contents are cached in multiple servers, the Directory Service will attempt to balance the load across Streaming Servers when assigning servers to a client. After a streaming session has been started, the Streaming Client will maintain a control connection with the Directory Service to receive updates on media data availabilities.

Notes:

- (a) The Streaming Server can be run in the same host as the Streaming Client.
- (b) The protocol between the Streaming Server and the Streaming Client is to be designed by the student. However it is recommended to adopt HTTP when TCP is chosen as the transport, as this will enable the Streaming Server to be tested with existing video players such as Windows Media Player or RealVideo Player. The Streaming Client Proxy can then also be tested with a HTTP server.
- (c) Flow control will be needed between the Streaming Server and the Streaming Client. When TCP is used for data transfer then flow control can be implicit. When UDP is used then an explicit flow control mechanism will be needed.
- (d) When using UDP for data transfer, implementation of packet loss recovery is optional.
- (e) The Client does not need to implement the full HTTP specification. It only needs to enable the media player to perform HTTP streaming through it (e.g., using localhost in the URL).