Design and Implementation of
WinTrace
An Internet Path Characterization Tool

A GRADUATE PROJECT TECHNICAL REPORT

Submitted to the Faculty of
The Department of Computing and Mathematical Sciences
Texas A&M University-Corpus Christi
Corpus Christi, TX

In Partial Fulfillment of the Requirements for the Degree of
MASTER of SCIENCE in COMPUTER SCIENCE

BY

Rui Shen

Fall, 2001

COMMITTEE MEMBERS

Dulal Chandra Kar, Ph.D.
Chairperson
David Thomas, Ph.D.
Member
Mario Garcia, PhD
Member
ABSTRACT

This project is the design and implementation of a MS-Windows-based network utility, WinTrace. WinTrace is able to perform three important functions: pinging a host and finding out the round-trip time, tracing the Internet path from a sending host to a destination host and finding out all the routers in between, and estimating the bandwidth of each link along the Internet path from a host to a destination host. WinTrace can perform all these three functions by sending ICMP (Internet Control Message Protocol) probe packets from a single source and then measuring round-trip times from responses received from destination hosts on probe packets. WinTrace can be used to find out the characteristics of the Internet links along a path and to detect the bottleneck link among all links. The results on the characteristics of Internet links can be very useful for future planning and extension of network services.
Table of Contents ........................................................................................................ II

ABSTRACT .............................................................................................................. I
LIST OF FIGURES ...................................................................................................... IV

1. BACKGROUND AND RATIONALE ................................................................. 1
   1.1 Background .................................................................................................... 1
   1.2 Rationale ...................................................................................................... 1
   1.3 Previous Work .............................................................................................. 2

2. NARRATIVE ...................................................................................................... 4
   2.1 Links and Bandwidth .................................................................................... 4
   2.2 Functions of WinTrace ................................................................................ 5
   2.3 Users of WinTrace ...................................................................................... 6
   2.4 User Interface ............................................................................................. 7
       2.4.1 WinTrace Options ................................................................................ 8
       2.4.2 Using WinTrace .................................................................................. 10
       2.4.3 Error Messages .................................................................................. 14
       2.4.4 Windows Functions and Help .............................................................. 15

3. ENVIRONMENT ............................................................................................... 16
   3.1 Minimum Hardware Required .................................................................... 16
   3.2 Minimum Software Required ..................................................................... 16

4. PROCEDURE .................................................................................................... 17
   4.1 ICMP .......................................................................................................... 17
   4.2 Socket Programming in WinTrace .............................................................. 18
       4.2.1 Non-blocking receive and the “select” function ................................. 20
       4.2.2 Basic Socket Structures in Windows Socket .................................. 22
       4.2.3 Implementation of Ping ................................................................. 24
       4.2.4 Implementation of Trace Routers .................................................. 28
       4.2.5 Implementation of Find Bandwidth .................................................. 31
   4.3 Algorithm used to Estimate the Bandwidth .............................................. 31
       4.3.1 Finding the Routers On the Way ..................................................... 31
       4.3.2 Finding the Min RTT for Each Router .............................................. 33
       4.3.3 The Relationship Between the “Slope / Intercept” and “Bandwidth / Propagation delay” .......................................................... 34
       4.3.4 The “Least Square Fit” Algorithm .................................................... 35
       4.3.5 Finding the Bandwidth Between any Contiguous Routers ............... 36
List of Figures

Figure 1: Links and Nodes ................................................. 4
Figure 2: Bottleneck Links ................................................. 5
Figure 3: WinTrace User Interface ..................................... 8
Figure 4: WinTrace Options Dialog Box ......................... 9
Figure 5: “Ping” Output ................................................. 11
Figure 6: “Trace Routers” Output .................................... 12
Figure 7: “Find Bandwidth” Normal Output ..................... 13
Figure 8: “Find Bandwidth” Verbose Output ................. 14
Figure 9: Probes......................................................... 32
Figure 10: RTT vs Packet Size For a Single Link ............ 35
Figure 11: RTT vs Packet Size For Multiple Links ............ 37
Figure 12: Flow Chart ................................................. 46

Table 1: Comparison of WinTrace with UNIX-Based Tools .... 48
1. BACKGROUND AND RATIONALE

1.1 Background

Internet is one of the most important media for people worldwide. It is an extensive source of information for education, entertainment, and communication. As more and more people are using Internet, the amount of data transferred over Internet is becoming more and more enormous day by day. Fast, efficient, and reliable data transfer between hosts is very important to maintain the quality of services over Internet. A single bottleneck link along a path can be a major cause for poor performance for data communications between two hosts on Internet.

1.2 Rationale

Although so many people are using Internet and so many data packets are travelling back and forth in Internet all the time, Internet has no central control or administration. In terms of ability to grow and scale, this is a strength, but it might be a weakness if someone wants to diagnose and fix problems that come along with fast growth [1]. In this situation we need some automated tools that can be used to measure the performance of various Internet links and to detect problems along a path without disrupting any ongoing services being provided over the path.

Some programs have been developed to characterize links and detect problems along any Internet path. Most of them are originally developed to work on a UNIX platform. As more and more people are using MS-Windows, it has become necessary to develop such useful utilities for Windows environment as well. That is why some useful UNIX networking probing tools like Ping and Tracerouter have been re-implemented for Microsoft Windows.
In this project, a MS-Windows-based network performance measuring tool, WinTrace, has been developed. By sending ICMP (Internet Message Control Protocol) packets from a single source and measuring round-trip times, WinTrace is able to perform the following three functions: 1) pinging a host and finding out the round-trip time, 2) tracing the path to a host and finding out all the routers in between, 3) and tracing the path to a host and estimating the bandwidth of each link along the path. Some similar programs have been developed by others to operate on UNIX environment [1], [4], [5]. But it is not easy to just port such programs from UNIX to MS-Windows since the programming Application Programming Interfaces (APIs) are quite different on these two different platforms.

This project was chosen for the following good reasons that can be ascribed to learning and developing skills in network protocols and programming:

- It requires a lot of research on similar UNIX tools.
- It requires performance analysis of different algorithms used in those tools and the consideration of possible changes to improve those algorithms’ efficiency and accuracy.
- It requires complete understanding of the related network protocols: UDP (User Datagram Protocol) and ICMP (Internet Control Message Protocol).
- It requires good Visual C++ programming skill.
- It requires good windows socket programming skill.
- The program will benefit the users who use MS-Windows environment.

1.3 Previous Works

In 1997, Van Jacobson from LBL's Network Research Group developed a tool, called Pathchar[2], to estimate performance characteristics of each link along an Internet path from a source to a destination host. Later, some problems with Pathchar were reported by
a research group from Osaka University in a paper “Improving Bandwidth Estimation for Internet Links by Statistical Methods” [3]. The paper points out that the reliability of the estimation of Pathchar needs to be improved. In 1999, another tool, called Pchar, was written by Bruce A. Mah based on the algorithm of the Pathchar [4]. Pchar improves the reliability of Pathchar by using a nonparametric approach. Clink [5], written by Allen Downey from Colby College, is another variant of Pathchar. The algorithm and interface of Clink are based on Pathchar, but it differs from Pathchar in the method used to estimate the bandwidth of a link. Clink also improves the estimation accuracy in the case of routing instability. All the tools mentioned above are designed for UNIX or UNIX-like systems.
2. NARRATIVE

2.1 Links And Bandwidth

The physical components of Internet are links and nodes (Figure 1). A node can be a desktop workstation, a network switch or a router. A node is also called a host. A link connects a pair of nodes. Links can be wires, cables and even radio waves. A set of links that connects a host with another one on Internet constitutes a path.

![Figure 1: Nodes and Links](image)

Performance of a network, a link, or a path on Internet can be described in terms of two important parameters: bandwidth and latency (also called delay) \[6\] \[7\]. The bandwidth of a network is defined by the number of bits that can be transmitted over the network in a certain period of time. For example, a network might have a bandwidth of 10Mbps, meaning that it is able to deliver 10 million bits every second. A link with the smallest bandwidth along a path is considered as the bottleneck link. The second performance metric, latency, corresponds to how long it takes a message to travel from one end of a network to the other. There are many situations in which it is more important to know how long it takes to send a message from one end of a network to the other end and back. We call this latency as the round-trip time (RTT) of the network.

Bottlenecks exist in Internet. A message sent from a host travels along a number of links along a path before it reaches the destination. The slowest link along a path largely determines the transfer time of a message. An example situation is shown in Figure 2.
Figure 2: Bottleneck Links

As can be seen in Figure 2, except the two middle links, the capacity of all other links is 10 Mbps. The links at the middle with 1 Mbps bandwidth can severely limit the performance of the network during communications of a host from the network part with router R1, R2, and R3 to a host in the network part with routers R5, R6, and R7.

2.2 Basic Mechanism of WinTrace

The implementation of WinTrace is based on the “Internet” layer as described in the TCP/IP (Transmission Control Protocol/Internetworking Protocol) Layering Model [7]. The TCP/IP Layering Model, which is also called the Internet Layering Model or the Internet Reference Model, contains five layers:

- Application (Layer 5)
- Transport (Layer 4)
- Internet (Layer 3)
- Network Interface (Layer 2)
• Physical (Layer 1)

The “Internet” layer specifies the format of packets sent across Internet as well as the mechanisms used to forward packets from a computer through one or more routers to a final destination.

WinTrace sends several ICMP packets (probes) to a remote host to measure the round trip time. A packet is a data unit sent over a packet-switched network. In the IP header of an ICMP packet, there is a field called TTL (Time to Live). The TTL field contains an allowed maximum number of links (hops) a packet can travel. The TTL field was initially included to eliminate infinite looping of a packet on the Internet. When a router gets a packet, it decreases the TTL value by one. Once a router finds a packet with the TTL value of zero, it sends an ICMP “Time exceeded message” back to the sender and discards the packet. To detect the first router, WinTrace sends a probe packet with the TTL value to 1. To detect others, it successively increases the TTL value by one before sending each time. By collecting the ICMP “Time exceeded message” from each router, WinTrace can identify all the routers on the way to the remote host. Based on round trip times measured for many different sized probe packets sent to a router, the bandwidth of the corresponding link ending to that router can be estimated using some estimation algorithm. WinTrace can estimate the bandwidth of each link along the path to a destination host.

2.3 Users of WinTrace

WinTrace is a tool with multiple functions and hence can be used in many different ways. It can be used to find whether some host is alive or not. It can be used to trace a path to a host and to find out the identity of all routers along the path. Most importantly, it can be used to find the bandwidth of each link along a path to a host. Users of WinTrace could be:
• Internetworking researchers
• Internet service providers
• Intranet designers
• Some advanced Internet users
• Any common user who is interested in finding Internet path characteristics

From the set of bandwidths obtained for all links along a path, the bottleneck can be identified. This can be a valuable information for networks designers, researchers, and network service providers for planning and growth.

2.4 User Interface

WinTrace has a user interface typical to any common Windows application. It contains menu options, command buttons, option buttons, etc., as required to run and interact with the tool for different functions. By selecting appropriate menu options, option buttons, and command buttons, a user can run to ping a host, to trace a route to a host, or to find bandwidths of all links to a host.
2.4.1 WinTrace Options

As WinTrace is a MS-Windows-based application, the interface looks very similar to a typical Windows application as in Figure 3.

![WinTrace User Interface](image)

**Figure 3: WinTrace User Interface**

As can be seen in Figure 3, in the host name textbox, one can enter the domain name or the IP address of a remote host such as www.tamu.edu or 165.95.8.68. By clicking on
the Tools menu option, one can open a window to choose different options as shown in Figure 4.

![WinTrace Options Dialog Box](image)

**Figure 4: WinTrace Options Dialog Box**

These options are described as follows:

**Trace Type:** WinTrace has three functions: ping, trace routers and find bandwidth. The default function is ping. A user can choose the trace type by clicking on any one of the radio buttons provided in the interface for different trace types.
**Ping Size**: A user can specify the packet size of each ICMP echo-request message. The default size is 32 bytes of data plus 36 bytes of ICMP and IP headers, altogether 68 bytes.

**Ping Times**: A user can specify how many times to send the ICMP echo-request message to the specified host. The default value is 8 times.

**Maximum Routers**: A user can specify the maximum number of routers to trace when using the trace routers function. The default value is 50.

**Number of Probes**: An user can enter the number of probe packets to be sent for each packet size. The default is 16. More accurate estimate of the bandwidth can be obtained by entering a larger number in this field.

**Find Bandwidth Output Mode**: A user can specify the output mode when using the find bandwidth function. The verbose mode dumps all the probing messages and any intermediate results on bandwidth calculation to the output box. The default mode just sends all the verbose output to the status bar while the output box shows clean and concise results.

### 2.4.2 Using WinTrace

WinTrace can be used in MS-Windows 95/98/Me/2000 platform (NT is not tested). Since the application uses raw sockets, a Windows 2000 user has to login as the administrator in order to use the tool. WinTrace has three functions: ping, trace routers and find bandwidth.

**Ping**

The default function of WinTrace is “ping”. By default, WinTrace sends eight ICMP request packets to the specified host. By collecting the ICMP echo (reply) packets,
WinTrace can measure the round trip time to that host. The packet size and the number of packets can be specified by the user through the option dialog box from the “Tool” menu. The output is shown in Figure 5.

![WinTrace Window](image)

Figure 5: “Ping” Output

**Trace Routers**

Through the option dialog box from the “Tool” menu, user can specify which function WinTrace needs to perform. The default function is “ping”. If the user selects the “Trace router” function, WinTrace will send ICMP packets to the specified host with different TTL value starting with 1. By collecting the ICMP “Time exceeded message” packets, WinTrace can get all the IP addresses of the routers on the way. The user can specify the
number of maximum routers to trace in the option dialog box. The output is shown in Figure 6.

![WinTrace Window](image)

**Figure 6: “Trace Routers” Output**

**Find Bandwidth**

If a user select the “find bandwidth” function from the option dialog box, WinTrace will first traces the route to find all the routers on the way, then send different sized echo-request ICMP messages to each router to measure the roundtrip time. By measuring the round trip time, the bandwidth of each path can be estimated using some “Least Square Fit Algorithm” and eventually the link bandwidth can be calculated from the path bandwidth. The Wintrace function “find bandwidth” can provide two possible kinds of
output. The default one is the normal output, in which only the routers and bandwidths are shown in the output box, the detail information are shown in the status bar (Figure 7).

![WinTrace Output](image)

**Figure 7: “Find Bandwidth” Normal Output**

The “verbose output” checkbox in the option dialog can be checked to tell WinTrace to dump all the detailed information to the output box as shown in Figure 8. By selecting the “Save as” option in the File menu, a user can save all the trace information to a single text file which can be used for future analysis.
2.4.3 Error Messages

Error messages are displayed in the output box when WinTrace fails. Error messages that Wintrace can display in various problematic situations are given below:

Message: Host not found.
Explanation: Either the Internet connection is faulty or the host name is not right.

Message: Unable to open a raw socket.
Explanation: Failed to open a raw socket.
Message: Receivefrom() - WSAError: 10042
Explanation: Socket errors.

Message: Unable to set the recvfrom FD.
Explanation: The “select” function can not set the file descriptor.

2.4.4 Windows Functions and Help

Since WinTrace is a windows-based tool, it has all the basic functions like any typical windows applications. From the menu option “File”, a user is able to save the results as a text file. The user can get help concerning WinTrace from the menu option “Help Topics”. The help file shows the usage of WinTrace and any other details a user needs to know to use the tool. It also lists all the available options with detailed descriptions for each option. An “About” submenu option can also be found in the “Help” menu to show some information concerning the program such as author, version, etc.
3. Environment

3.1 Minimum Hardware Requirements

WinTrace requires a Pentium class or equivalent computer with a minimum of 16 MB of random access memory and 500 MB of free hard disk space. A Pentium III with 64 MB RAM is recommended. Since the execution of a command might take several minutes, a stable, fast Internet connection is required.

3.2 Minimum Software Requirements

WinTrace is able to run under MS-WINDOWS 95/98/Me/2000. When WinTrace is used in Windows 2000, the user must login as the administrator. Internet Explore 4.0 or above is required to display the help file correctly.
4. Procedure

4.1 ICMP

ICMP stands for the Internet Control Message Protocol. It is a network layer protocol along with Internetworking Protocol (IP). ICMP is used in WinTrace to get internetworking control messages.

ICMP is a “support protocol” that does not transport data. Instead, as its name implies, it delivers control, error, and informational messages between Internet hosts [8]. The Internetworking Protocol (IP) is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment. The ICMP messages typically report errors in processing of datagrams. To avoid the infinite regress of messages about messages, no ICMP messages are sent about ICMP messages. There are many types of ICMP messages. Most of them affect an application’s operation quite visibly. An ICMP message can be retrieved by some specific Windows Socket functions. For example, if a returned value of the function WSAGetLastError is “WSAHOSTUNREACH”, it indicates the receipt of the ICMP “host unreachable” error message.

ICMP messages are sent using the basic IP header. The first byte of the data portion of an ICMP packet is a ICMP “Type” field; the value of this field determines the format of the remaining data. The second byte is the “Code” of that ICMP Type. Then the two bytes checksum is followed. The checksum is the 16-bit one’s complement of the one’s complement sum of all the 16-bit consecutive blocks of the ICMP message starting with the ICMP message type field. For computing the checksum, the initial checksum field should be zero.

Three types of ICMP messages are used in WinTrace:

- **Type 0 and 8, Code 0**: echo request and echo reply messages
WinTrace sends an ICMP echo request packet to a host. After the host receives the echo request packet, it sends an ICMP echo reply packet back. The echo request packet sent by WinTrace has a timestamp of the sending time of that packet. When WinTrace receives the echo reply packet, it records another timestamp. The difference of the two timestamps will be the round trip time (RTT) of that packet on that particular Internet path.

- **Type 11, Code 0: Time to live exceeded in transit**

Before sending an ICMP packet, WinTrace sets the TTL value in the IP header of the ICMP packet. Then, each router on the way to the destination decreases the TTL value by 1. The router which gets a “0” on the TTL value sends an ICMP “time to live exceeded in transit” message back to WinTrace.

### 4.2 Socket Programming in WinTrace

All the three functions “Ping”, “Trace routers” and “Find bandwidth” require socket programming. A socket is an abstraction used by network application to perform most network communications. A socket is an endpoint of communication, created in software, equivalent to a computer’s network interface. It allows a network application to “plug into” the network. Typically there is only one physical network interface on a computer, but the number of sockets can be far more. There is a one-for-many correspondence between a network interface and sockets, it is called multiplexing. Sockets were first developed on UNIX at the University of California at Berkley. Sockets were designed so that most network communications between applications could be performed in the same way the applications would read and write files. Making a socket connect to another application does require a different set of information than opening a file. To open a socket connection, we need to know the computer on which the other application is running and the port on which it is listening. A port is like a phone extension, and the computer address is like the phone number. The basic socket operations include: create a socket, make a connection, send and receive messages, and close the connection.
Similar to Berkley sockets, Microsoft developed a set of sockets called “WinSock” that can be used in windows application development. WinSock2 is used to develop WinTrace. Windows Sockets (WinSock) are open interfaces for network programming under Microsoft Windows. WinSock resulted from a cooperative effort among a number of network software vendors. The Windows Sockets API (WSA) consists of a collection of functions, data structures, and conventions. The WSA provides standard access to the network services of an underlying protocol stack to any Microsoft Windows application. Unlike many other APIs, WinSock provides binary compatibility as well as source code compatibility. Consequently, programs require no changes at all when their executable versions are moved from one network system vendor to another.

WinTrace uses WinSock2 which uses the socket paradigm that was first popularized by Berkeley Software Distribution (BSD) for UNIX. It was later adapted for Microsoft Windows in Windows Sockets 1.1. One of the primary goals of WinSock2 has been to provide a protocol-independent interface, fully capable of supporting emerging networking capabilities, such as real-time multimedia communications. WinSock2 is an interface, not a protocol. As an interface, it is used to discover and utilize the communications capabilities of any number of underlying transport protocols. Because it is not a protocol, it does not in any way affect the bits on the wire, and does not need to be utilized on both ends of a communications link.

Actual implementations of the “ping”, the “trace routers” and the “find bandwidth” functions are heavily depended on the windows socket programming. A number of problems were encountered during the implementation. Most of them will be discussed in the next sections. In order to understand the socket programming, several built-in structures will be discussed in section 4.2.2. The detailed socket implementation of the three functions in WinTrace will be described later.
4.2.1 Non-blocking “receive” and “select” function

The windows function “recvfrom” can be used to receive data from a socket. By default, “recvfrom” function works in blocking mode, which means that if no incoming data is available at the socket, the “recvfrom” function blocks and waits for data to arrive. WinTrace sends an ICMP echo request message to a remote host and sets the TTL field in the IP header to any arbitrary value, and then use the “recvfrom” function to wait for the returning ICMP “Time exceeded” message. If the default “blocking mode” is used, the program might sit there indefinitely waiting for a lost packet. There are two ways to solve this problem. One way is to set the “recvfrom” to a non-blocking mode using the “ioctlsocket” function:

```
ioctlsocket(srcFd, FIONBIO, &nb);
```

“srcFd” is a file descriptor for a socket which is ready to receive any information. The “FIONBIO” command followed by a nonzero u_long value will set any of the socket operation of “srcFd” to a non-blocking mode. After the setting, if we use the “recvfrom” function to receive data from the “srcFd” socket, the function will try to listen to a particular socket port. If nothing is received during a system timeout, it will quit and return an error code. The error code can be retrieved by the “WSAGetLastError” function.

The other way to solve the blocking problem is by using the “select” function. The Windows “select” function determines the status of one or more sockets, waiting if necessary, to perform synchronous I/O. The prototype of the “select” function is shown below:

```
int select (  
    int nfds,  
    fd_set FAR * readfds,  
    fd_set FAR * writefds,  
```

20
The set of sockets for which a given status is requested is indicated by an FD_SET structure. The FD_SET contains a set of file descriptors. The select function returns the number of sockets meeting the conditions. A set of macros is provided for manipulating an FD_SET structure. The following macros can be used to operate the file descriptors in the FD_SET:

- **FD_CLR**(*s*, *set*):
  It is used to remove the descriptor *s* from *set*.

- **FD_ISSET**(*s*, *set*):
  It is used to test whether *s* is a member of the *set*, return nonzero if *s* is a member of the *set*. Otherwise, zero.

- **FD_SET**(*s*, *set*):
  It is used to add the descriptor *s* to *set*.

- **FD_ZERO**(*set*):
  It is used to initialize the *set* to the NULL set.

In WinTrace, we have to include the descriptor of the socket to an FD_SET to receive data in non-blocking mode. The following example shows how to include descriptor srcFd in set readfds:

```c
fd_set FAR * exceptfds,
const struct timeval FAR * timeout
);

fd_set readfds;
FD_SET(srcFd, readfds);
select(1, &readfds, NULL, NULL, &Timeout);
```
Since we only use the select function to set the timeout for a read *fd_set*, the write *fd_set* field and exception *fd_set* field are set to NULL.

### 4.2.2 Basic Socket Structures in Windows Sockets

There are three basic structures in Windows Socket API: *SOCKADDR_IN*, *IN_ADDR* and *HOSTENT* which are used in WinTrace. Their usages are described below:

**SOCKADDR_IN**

The *SOCKADDR_IN* structure is used by Windows Sockets to specify a local or remote endpoint address to which to connect a socket and the structure is defined as:

```c
struct sockaddr_in{
    short    sin_family;   // Address family (must be AF_INET)
    unsigned short sin_port;   // IP port
    struct   in_addr  sin_addr;  // IP address
    char      sin_zero[8];  // Paddings
};
```

The source and destination host address can be defined as a *sockaddr_in* structure as follows:

```c
sockaddr_in saDest, saSrc;
```

**IN_ADDR**

This structure is used to store the IP address of a host. The structure is defined as:
struct in_addr {
    union {
        struct {
            unsigned char s_b1,
            s_b2,
            s_b3,
            s_b4;
        } S_un_b; // IP address stored as four characters
        struct {
            unsigned short s_w1,
            s_w2;
        } S_un_w; // IP address stored as two short values
        unsigned long S_addr; // IP addr stored as long
    } S_un;
};

The sin_addr field of a sockaddr_in structure is an in_addr structure that stores the IP address.

HOSTENT

The HOSTENT structure contains the information of the host name. The details of HOSTENT structure are described below:

struct hostent {

    char FAR * h_name;
    char FAR * FAR * h_aliases;
    short h_addrtype;
    short h_length;
    char FAR * FAR * h_addr_list;
}
Given a host name such as \texttt{www.tamu.edu}, we must obtain its corresponding IP address which can be used to initialize the socket. The “\texttt{gethostbyname}” function is used to obtain the IP address in a HOSTENT structure:

\begin{verbatim}
HOSTENT host;
host = gethostbyname ("www.tamu.edu");
\end{verbatim}

Then the \texttt{h_addr} field of the HOSTENT structure can be cast and assign to the \texttt{s_addr} IP address field of the destination address structure as follows:

\begin{verbatim}
saDest.sin_addr.s_addr = *((u_long FAR *)(host
->h_addr));
\end{verbatim}

The \texttt{h_name} field of the HOSTENT structure can be fetched to obtain the official name of the host.

\subsection*{4.2.3 Implementation of Ping}

The Ping function sends an ICMP echo request packet to a host. To construct and send an echo request packet, and to receive and interpret an echo reply message, the following structures are used:

\textbf{IP Header}:

\begin{verbatim}
typedef struct tagIPHDR {
    u_char       VIHL;   // Version and IHL
    u_char       TOS;    // Type Of Service
    short        TotLen; // Total Length
} IPHDR;
\end{verbatim}
short ID;
short FlagOff; // Flags and Fragment offset
u_char TTL;     // Time To Live
u_char Protocol; // Protocol
u_short Checksum; // Checksum
struct in_addr iaSrc; // Source IP Address
struct in_addr iaDst; // Destination IP Address
}IPHDR;

ICMP Header:

typedef struct tagICMPHDR {
    u_char Type;     // Type
    u_char Code      // Code
    u_short Checksum; // Checksum
    u_short ID;      // Identification
    u_short Seq;     // Sequence Number
}ICMPHDR;

ICMP Echo Request:

typedef struct tagECHOREQUEST {
    ICMPHDR icmphdr;     // ICMP header
    DWORD dwTime;         // Sending time stamp
    char cData[REQ_DATASIZE]; // packet data
}ECHOREQUEST;

ICMP Echo Reply:

typedef struct tagECHOREPLY {
    IPHDR ipHdr;
}ECHOREPLY;
The following steps explain the implementation of the “Ping” function:

**Step 1:** Declare two variables to store host addresses:

```c
struct sockaddr_in saFrom, saDest;
```

The “`sockaddr_in`” structure is the address structure for an Internet host address. Then set the fields in the structure variable “`saDest`”:

```c
saDest.sin_family = AF_INET;
saDest.sin_port = 6000;  // port number set to 6000
```

**Step 2:** Create a raw socket using the following statement:

```c
sd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

“`sd`” is a socket descriptor which is used to refer to this socket. The argument `AF_INET` specifies the Internet address family. The second argument `SOCK_RAW` means that this socket will be used to send raw data. The last argument `IPPROTO_ICMP` specifies that the socket will be used to communicate ICMP messages.

**Step 3:** Create an ECHOREQUEST and an ECHOREPLY objects, and then initialize the value as follows:

```c
ECHOREQUEST echoReq;
ECHOREPLY echoReply;
echoReq.icmpHdr.Type = 8;
echoReq.icmpHdr.Code = 0;
```
echoReq.icmpHdr.Checksum = 0;

**Step 4:** Get the system time stamp:

```c
echoReq.dwTime = GetTickCount();
```

**Step 5:** Calculate checksum:

```c
echoReq.icmpHdr.Checksum = ip_checksum((u_short*)
    &echoReq,size);
```

**Step 6:** Send the echo request packet:

```c
Nret = sendto(sd,              // socket descriptor
    (LPSTR)&echoReq,  // buffer to send
    size,            // packet size
    0,               // flag
    (LPSOCKADDR)saDest,// destination address
    sizeof(SOCKADDR_IN));// address length
```

**Step 7:** Receive the echo reply packet:

```c
nRet = recvfrom(sd,
    (LPSTR)&echoReply, // buffer to hold incoming pkt.
    sizeof(ECHOREPLY), // size of buffer
    0,                 // flags
    (LPSOCKADDR)saFrom, // From address
    &nAddrLen);        // pointer to address len
```

**Step 8:** Find the current system time minus the `dwTime` field found in the echo reply packet. The result will be the round trip time (RTT) of the packet for that path.

```c
DwElapsed = GetTickCount() - echoReply.echoRequest.dwTime
```
The Ping function by default repeats all the above steps eight times. However, a user can change the ping times as well as the ping packet size from the “option” dialog box of the WinTrace menu.

4.2.4 Implementation of Trace Routers

The "trace routers" function sends packets with some specific TTL value for each probe to discover all the routers along a path. In order to discover a router, the "trace routers" function needs to get the ICMP “Type 3, Code 0, time to live exceeded in transit” message from the router. The detailed steps to get the ICMP “TTL exceeded” message in MS-Windows environment are described as follows:

**Step 1:** Declare two variables to store host addresses:

```c
struct sockaddr_in saFrom, saDest;
```

**Step 2:** Set the fields of the address from which the ICMP message will be received:

```c
saFrom.sin_family = AF_INET;
saFrom.sin_port = htons(0);
saFrom.sin_addr.s_addr = htonl(INADDR_ANY);
```

The “htons” and “htonl” functions convert a 32-bit number from the host byte order to TCP/IP network byte order (which is big-endian). Using INADDR_ANY as the IP address allows the sending host to receive ICMP messages from any of its multiple interfaces available on it.

**Step 3:** Create two raw sockets, one is for sending raw ICMP echo request message, the other for receiving ICMP “TTL exceeded” message:
srcFd = socket (AF_INET, SOCK_RAW, IPPROTO_ICMP);
destFd = socket (AF_INET, SOCK_RAW, IPPROTO_ICMP);

Step 4: Bind the receiving socket to the address:

bind(srcFd, (LPSOCKADDR)&saFrom,
     sizeof (SOCKADDR_IN));

Step 5: Set the receiving socket descriptor to non-blocking mode as discussed in section 4.2.1:

ioctlsocket(srcFd, FIONBIO, &nb);

Step 6: Set the TTL field in the IP header:

Setsockopt (destFd, IPPROTO_IP, IP_TTL,
           (const char*) &ttl, sizeof(ttl));

Step 7: Fill the sending buffer:

send_buf = (ICMPHeader*) new char[packet_size];
send_buf->type = 8;
send_buf->code = 0;
send_buf->checksum = 0;
send_buf->id = (USHORT) GetCurrentProcessId();
send_buf->seq = 0;
send_buf->timestamp = GetTickCount();

Step 8: Send the packet:
sendRet = sendto (destFd, (char *)send_buf, packet_size, 0, (LPSOCKADDR)&saDest, sizeof(SOCKADDR));

**Step 9:** Wait for some time:

sleep (700);

**Step 10:** Receive the packet sent by a remote router:

recvRet = recvfrom(srcFd, (LPSTR)recvBuffer, sizeof(recvBuffer), 0, (LPSOCKADDR)&saFrom, &len);

**Step 11:** In an array with the TTL value as the index, store the IP address of the router which sent the ICMP “TTL exceeded” message:

hostIp[ ttl].Format("%s", inet_ntoa(saFrom.sin_addr));

By changing the TTL value in a packet, sending it, and receiving the ICMP “Time Exceeded” message, all IP addresses of the routers on the path are obtained and stored in the array "hostIp". The "hostIp" array can be used in the find bandwidth function.
4.2.5 Implementation of Find Bandwidth

Basically, the find bandwidth function uses the “hostIp” array created by the "trace routers" function as described in the previous section. It sends different sized ICMP echo request messages to each of the address in the “hostIp” array. The returned ICMP echo reply message is timestamped. By subtracting the timestamp in the corresponding ICMP echo request packet from the timestamp of the returned ICMP echo reply packet, the round trip time (RTT) can be calculated. We obtain a series of “RTT vs Packet Size” pairs of data. Using the Least Square Linear Fit algorithm, the link bandwidth between the sending host and any router can be figured out. Based on the definition of the bandwidth, then we can deduce the bandwidth of a particular link. The details of the algorithm are discussed in Section 4.3.

4.3 Algorithm used to Estimate the Bandwidth

4.3.1 Finding the Routers Along an Internet Path

As mentioned in Section 4.2.4, IP packets contain a time-to-live (TTL) field initialized by the original sender, which is decreased by one at each intermediate router. If the field is decremented to zero, the packet is discarded and an error indication packet (an ICMP “TTL exceeded” message) is sent back to the original sender. The source address of the ICMP “TTL exceeded” packet identifies the router that discarded the data packet. Therefore, if a packet is sent to any destination but with the TTL field set to $n$, the router at $n$ hops away along the path is forced to identify itself.
As shown in Figure 9, when the WinTrace program is executed at some host, several varying sized packets are sent to the first router. The TTL field in the packet header is set to 1. When the first router gets the packet, it decreases the TTL value by one, then it finds out that the TTL value has become 0. The router then discards the packet and sends an ICMP message back to the packet sender. As WinTrace is listening on a particular port, therefore, it gets the ICMP “TTL exceeded” message. Since the returning ICMP message contains the source address of the router that discarded its packet, it can easily identify the router. By sending packets with different TTL values such as 1, 2, 3, . . . , n, Wintrace finds all the routers address on the way to the destination.
4.3.2 Finding the Minimum RTT for Each Router

After WinTrace gets the address of all the routers on the way, it starts to send echo request messages to each router. By calculating the difference of the timestamps the roundtrip time to the router is calculated. The bandwidth of the path to the router can be obtained from the packet size and the round trip time as follows:

\[ \text{Bandwidth} = \frac{\text{packet size}}{(\text{RTT}/2)} \]

Since the RTT is the round trip time, in order to calculate the bandwidth, we need to divide RTT by 2.

Internet is a very dynamic environment. During heavy traffic, a router may not be able to forward all the packets immediately it has received. It may need to store packets in a queuing buffer. In this situation, a particular packet might sit in the buffer for a while, therefore the RTT measured will include some queuing delay as well as some propagation delay and hence the straight forward calculation of the path bandwidth will not provide any accurate result. Although the propagation time is constant over path but the queuing delay is variable and depends on traffic condition along the path. Hence, RTT measurements for the same packet size might vary because of the variable queuing time. Another problem WinTrace faces is that when the queuing buffer of a router is full, it discards all the packets it gets at that time. So a packet might be lost on the way also and the RTT to the intended router can not be measured.

Accordingly, WinTrace has to collect several RTT samples on the same link with the same packet size. The minimum of all sample RTT is used to estimate the bandwidth. The packet that has minimum RTT is assumed not to have experienced any queuing delay. By default, WinTrace sends 32 probes for each size of data, then gets the minimum RTT for that particular packet size.
In order to increase the accuracy, WinTrace repeats above probes with different packet sizes. As shown in Figure 5, although the RTT values of each same size packet might be different, the minimum RTTs of varied size packets should form a line.

4.3.3 The Relationship Between the “Slope / Intercept” and “Bandwidth / Propagation delay”

The latency is basically half of the RTT. It contains three components: the propagation delay, the transmission time and the queuing time.

\[
\text{latency} = \text{propagation} + \text{transmission} + \text{queuing} \quad (4.3.3.a)
\]

Since we send multiple probes for each packet size and we assume the minimum RTT does not involve any queuing delay, we could ignore the queuing time. The transmission time is actually the value of the packet size divided by the bandwidth. So the formula turns to be as follows:

\[
\text{RTT/2} = \text{propagation} + \text{packet size / bw} \quad (4.3.3.b)
\]

The above formula is exactly what is presented by the curve in Figure 10. From the curve and the formula, we obtain:

1. The intercept of the linear fit estimates the propagation delay

2. Since the slope of the linear fit represents the value of RTT/2 divided by the packet size, so the inverse of the slope is the bandwidth of the link.
As explained above, the problem of finding the bandwidth becomes the problem of finding the slope of the “RTT vs Packet Size” line. WinTrace collects data on minimum RTTs for different packet sizes and corresponding RTT, uses some “Least Square Linear Fit” Algorithm to fit a straight line, and then obtains the slope of the line. As discussed above, the inverse of the slope is the bandwidth of that link.

4.3.4 The “Least Square Linear Fit” Algorithm

The method of “Least Squares Linear Fit” assumes that the best-fit curve of a given type is the curve that has the minimal sum of the deviations squared (least square error) from a
given set of data. The “Least Square Linear Fit” algorithm can fit all the data in the data set to a straight line. The algorithm can be used to solve our problem as shown in Figure 10. WinTrace collects data on RTT’s for many packet sizes. Since we know it will form a line based on the equation (4.3.3.a), we use the “Least Square Linear Fit” algorithm statistically to find out the slope and intercept of that line.

\[
\text{LeastSqFit}(\text{double } y[], \text{ double } x[], \text{ int } size, \text{ double}& \text{ slope, double}& \text{ intercept})
\]

As shown above, the “Least Square Linear Fit” algorithm used in WINTRACE takes two array parameters to store packet sizes and minimum RTTs. The parameter “size” represents the number of data points. Calculated results on the slope of the line and the intercept of the line are available in the parameters "slope" and "intercept" respectively.

### 4.3.5 Finding the Bandwidth Between any Contiguous Routers

By repeating the steps described in Section 4.3.2 and Section 4.3.3, WINTRACE can obtain all the “RTT/2 vs Packet Size” curves. In Figure 11, the “RTT/2 vs Packet Size” curves of the first seven links are shown together. By observation, the longer the path is, the larger is the intercept and the larger is the slope. The larger intercept means larger propagation delay.

We can find the bandwidth of a path from source host to any router. But the fact is, in order to find the bottleneck the bandwidth between any two contiguous routers is much more important than the overall bandwidth between a host to any router. Lets assume we have found the bandwidth “bw1” of “link1” which is the bandwidth from the source host to the first router and “bw2” of “link2” which is the bandwidth from the source host to the second router. We would like to find out the bandwidth of the link between the first router and the second router. We know the following equations are legal according to the situation described above:
Figure 11: RTT vs Packet Size For Multiple Links

\[ \text{latency}_1 = \text{propagation}_1 + \frac{\text{size}}{\text{bw}_1} \]  \hspace{1cm} (1)

\[ \text{latency}_2 = \text{propagation}_1 + \text{propagation}_2 + \frac{\text{size}}{\text{bw}_1} + \frac{\text{size}}{\text{bw}} \]  \hspace{1cm} (2)

The following equation is also legal for the second link where \( \text{bw} \) is the bandwidth between the first link and the second link:

\[ \text{latency}_2 = \text{propagation}_1 + \text{propagation}_2 + \frac{\text{size}}{\text{bw}_1} + \frac{\text{size}}{\text{bw}} \]  \hspace{1cm} (3)

Subtracting equation (3) from equation (2), we obtain:
\[
\frac{\text{size}}{\text{bw2}} - \frac{\text{size}}{\text{bw1}} = \frac{\text{size}}{\text{bw}} \quad (4)
\]

Since slope is the inverse of the bandwidth, we get:

\[
\text{size} \times \text{slope2} - \text{size} \times \text{slope1} = \frac{\text{size}}{\text{bw}} \quad (5)
\]

Removing the \text{size} factor from both sides, we obtain:

\[
\text{slope2} - \text{slope1} = \frac{1}{\text{bw}} \quad (6)
\]

Eventually we obtain the formula to calculate \text{bw} from \text{slope1} and \text{slope2}:

\[
\text{bw} = \frac{1}{\text{slope2} - \text{slope1}} \quad (7)
\]

Equation 7 is used in WinTrace to calculate the bandwidth of a link between any two contiguous routers.

### 4.4 Implementation of WinTrace in Visual C++

#### 4.4.1 Using the Application Wizard

Implementation starts by using the MFC AppWizard (exe) in Visual studio to create a bare bone WinTrace. In the type of the application dialog, we have to choose “Dialog Based” application since WinTrace has to have a text box to accept a host name string from a user. Neither “Single Document” nor “Multiple Document” type will support this. Unfortunately, a dialog based application does not have any menu templates which both
the “Single Document” type and the “Multiple Document” type have. So we have to add all the menu items from scratch. In the next step, we need to tell the wizard to include the Windows Socket support. One of the items in the last screen of the wizard is very important. It asks a question “How would you like to use MFC library?”. In order to let the WinTrace function properly on different windows platform, the answer “As a statically linked library” must be selected. This selection will bundle all the MFC runtime libraries required by WinTrace to the WinTrace executable file. In this way, WinTrace will not have any problem to find any library to use at runtime. The tradeoff is that the executable file size of WinTrace will be much bigger than the one using the selection, “As a shared library”. But some library may not be present in a particular Windows platform, in that case, WinTrace will not be able to run. Instead, it will prompt an error message such as “XXX.dll is not found”.

4.4.2 Creating the Application Layout

WinTrace has a text box at the top of the application window to get the host name string from the user, and a list control box in the middle to show the output of the trace results. A status bar at the bottom is also very important to tell the user the current status of WinTrace. The “find bandwidth” function of WinTrace usually takes long time. Without a status bar, a user will not be able to know the current status of WinTrace. There are also two command buttons on the window. The “trace” button tells WinTrace that the host name is ready. The “clear” button can clear the contents in the list control box. Implementation of all objects will be explained in more detail below.

Host Name and Status Bar Text Box

The content in this text box is associated with a CString object “m_hostname”. After the “trace” button is pushed, the OnTrace function is triggered. The OnTrace function first uses the UpdateData (TRUE) to get the host name from the host name text box and then calls the startTrace function to start the tracing. The function UpdateData (TRUE) is used
to update the data from interface controls to the program variables. In our case, the contents in the host name text box will be stored in the CString “m_hostname” after the UpdateData (TRUE) is executed. If the data is to be updated from a program variable to an interface control, UpdateData (FALSE) will be used. Actually the OnInitDialog which initializes the WinTrace dialog also uses the UpdateData (FALSE) to initialize the contents of the host name text box to www.tamu.edu, for example.

The status bar is also a text box. The text in the is initialized to “Ready”. When the task of tracing is in progress, WinTrace continuously updates the text in this text box. For the ping and trace routers functions, if trace is in progress, the status bar will display a message similar to “Ping falcon.tamucc.edu”. After the end, the text in the text box is set to “Done”. When WinTrace is asked to find bandwidth, the status bar displays the packet size and the RTT information for each probe if the verbose mode of “find bandwidth” is not selected.

**Output List Control**

A list control object is used in WinTrace to store the data collected and display messages. The following member functions of a list control object are used in WinTrace:

```cpp
m_list.InsertColumn(0,"Trace Results:",LVCFMT_LEFT,550);
```

`m_list` is the name of the list control object. In the dialog initialization stage, a “Trace Results” column is created. The alignment attribute of the column is set to "left" and the width of the column is set to 550.

When we need to store any items, we first use GetItemCount method which returns the number of items in the list control, then we use the InsertItem method to add the item to the last position.

```cpp
int n = m_list.GetItemCount();
```
When the clear button is pushed, it calls the OnClear method. The OnClear method only has one statement:

```cpp
m_list.DeleteAllItems();
```

All the items will be deleted from the list control object after execution of the above statement.

**Menus**

Since the application wizard does not create the menu templates for WinTrace, we have to create the menu from scratch. The menu creation has two steps: create the menu and associate the menu with handlers.

In Visual Studio, it is very easy to create menus. By clicking the resource tab in the workspace pane, then by right-clicking the “menu” fold, one can add a new menu. To add any items to the menu one can just type the item name. In order to ask a menu item to do a certain task, a menu item must be associated with a handler. By using the Classwizard, a handler for each menu item can be added.

For example, WinTrace has an “Option” menu item in the “Tool” menu. The “Option” menu item is associated to “ID_MENU_OPTIONS”. The Classwizard adds a line to the MESSAGE_MAP of the dialog box

```cpp
ON_COMMAND(ID_MENU_OPTIONS, OnMenuOptions)
```

Then the “ID_MENU_OPTIONS” is now associated to the function OnMenuOptions. In the OnMenuOptions function, there is only one statement:
The object `m_dOptDlg` is an instance of the “Option” dialog. The above statement is used to display the “Option” dialog. When a user clicks the “Option” menu item in the “Tool” menu, the “Option” dialog will be displayed.

### 4.4.3 Windows Messages

Windows messages are very important to Windows programming. Everything that happens in a Windows program is mediated by messages. Messages are sent to application windows when the user presses a key, or clicks, releases, or moves the mouse. Messages are also sent from some other places in an application to request for a resource, or to report the status of a resource whether it is available or not, or to tell another part of the application to carry out a certain task.

A message has three parts: a message number, a word parameter and a long parameter. The message number is usually disguised through the use of a `#define` directive. For example, the message generally referred to as `WM_COMMAND` has a numeric value, defined in WINUSER.H as:

```c
#define WM_COMMAND 0x0111
```

Of course, no one ever refers to this command by its hexadecimal number. It is always referred as `WM_COMMAND`. The parameters are passed automatically to the code that “catches”, or deals with the message. Often the word parameter is a pointer to a structure, window, or buffer, while the long parameter is a simple number such as a status code, or the number of bytes in the buffer.

WinTrace uses several messages to control objects. For example, the message `WM_MSG_STATUSBAR` is the message used to tell the status bar to update its text.
In order to use it, it must be first defined in “stdafx.h” as follows:

```
#define WM_MSG_STATUSBAR WM_USER + 0x0101
```

The hexadecimal number 0x0101 is used as an offset to the base number WM_USER to include all used-defined messages.

The message function prototype is to be added in the message map of the header file:

```
LRESULT OnStatusbar(WPARAM wParam, LPARAM lParam);
```

In the message map of WinTraceDlg.cpp, the following will tell windows to trigger the OnStatusbar function when a WM_MSG_STATUSBAR message is received.

```
ON_MESSAGE(WM_MSG_STATUSBAR, OnStatusbar)
```

Now the implementation of the OnStatusbar function will be able to handle the WM_MSG_STATUSBAR messages.

### 4.4.4 Saving Trace Results

In order to let the user analyze the WinTrace results, the `OnFileSaveAs` is implemented to let the user save all the contents in the output list control box to a text file. As described in Section 4.4.2, the `OnFileSaveAs` is associated with the event when a user clicks the “Save As” menu item in the “File” menu. The first thing of `OnFileSaveAs` is to display a “save file as” dialog box. The following statement brings the dialog box:

```
CFileDialog m_fDlg (FALSE, NULL, "trace.txt",
                OFN_OVERWRITEPROMPT, "Text Files (*.txt)|*.txt||",
                this);
```
The “Save File As” dialog box is a CFileDialog object. When the first parameter sets to FALSE, it displays the “save file as” dialog. If it is set to TRUE, it will display the “Open File” dialog. The second parameter suggests a default file name to use. The third parameter is a combination of several flags. The OFN_OVERWRITEPROMPT flag will prompt a “File already exists, do you want to over-write?” message. The fourth parameter tells the dialog box only displays the file with the “txt” extension.

After the user pushes the OK button in the “Save File As” box, WinTrace will create an “ofstream” file object. WinTrace will then retrieve all the items from the list control object and store them in the file object. A sample code for the task is shown below:

```cpp
if (m_fDlg.DoModal() == IDOK) {
    ofstream OutFile(m_fDlg.GetFileName());
    CString s;
    int n = m_list.GetItemCount();
    for(int i = 0; i < n; i++) {
        s = "";
        s = m_list.GetItemText(i, 0);
        OutFile << s << "\n";
    }
    OutFile.close();
}
```

4.4.5 Help Topics

In order to prepare the help topics for WinTrace, a tool named “HTML Help Workshop” (HHW) is used to create the help files. HHW is a tool that comes with the Microsoft Visual Studio. After installation of HHW, a few steps through the "Settings" dialog box in the "Project" menu of Visual C++ IDE are required. Essentially the steps tell the C++ compiler where to find the library to make use of the HHW. In WinTrace,
the OnHelpHtml is called to handle the event when a user clicks the “Help Topics” menu item in “Help” menu. The OnHelpHtml has one statement as follows:

```
HtmlHelp(NULL,"WinTraceHelpHtml.chm",HH_DISPLAY_TOPIC,0);
```

The above statement asks Windows to run the file “WinTraceHelpHtml.chm” that was created and compiled by HHW. To display this help file it requires the support from Internet Explorer version 4.0 or later.
4.5 Flow Chart for WinTrace

The flow chart shown in Figure 12 summarizes the logic used in implementation of WinTrace.

Figure 12: Flow Chart for WinTrace
4.6 Testing

Several testing tasks were conducted to validate the application. Those tasks can be summarized in following four categories:

**Windows Application Behavior:**

- The “save as text file” option in the application menu was fully tested. The contents of the output box can be saved in a normal text file.
- The checking function of data input from the Options dialog was tested, which confirms that the application uses only valid parameters.
- The text in “Help Topics” was checked and it meets the needs of beginner users of the application.
- The keyboard response was tested. The default push button is always the “Trace” button.
- WinTrace was tested on MS-WINDOWS95/98/Me/2000 and was found to work on these Pentium class computers without any problem.

**Application Functionality Validation:**

- The test shows that the parameters set by a user in the Options dialog are successfully passed to the application.
- WinTrace was tested for invalid host names or IP addresses. If an invalid hostname or IP address is entered by a user, a “Host not found” message is displayed.
- Extensive test runs show that all the three functions of WinTrace, such as "Ping", "Trace routers" and "Find bandwidth", work fine for different hosts.
Functions Accuracy:

- Since WinTrace uses a micro second level timer, it can get more accurate measurement of RTT than the “ping” program available in MS-Windows environment.
- The “Trace routers” function was tested on many paths. The results were compared with the results of other applications, such as “tracert” in MS-Windows. It is found that that it can trace correct routers on the way in correct sequence.
- The results of the “Find bandwidth” function were compared with the results obtained using three similar UNIX tools: Pathchar, Clink, and Pchar on a path with three links or hops. The results are shown in the following table. It can be seen that it provides similar results to other UNIX-based tools.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Hop 1 (Ethernet)</th>
<th>Hop2 (FDDI)</th>
<th>Hop3 (Ethernet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pathchar</td>
<td>7.7 Mbps</td>
<td>41 Mbps</td>
<td>9.7 Mbps</td>
</tr>
<tr>
<td>Clink</td>
<td>6.92, 7.48, 7.48 Mbps</td>
<td>32.2, 40.8, 75.3 Mbps</td>
<td>8.7, 8.95, 9.8 Mbps</td>
</tr>
<tr>
<td>Pchar</td>
<td>7.76 Mbps</td>
<td>41.2 Mbps</td>
<td>9.04 Mbps</td>
</tr>
<tr>
<td>WinTrace</td>
<td>7.1 Mb/s</td>
<td>&gt; 10 Mbps</td>
<td>8.9 Mbps</td>
</tr>
<tr>
<td>Actual Bandwidth</td>
<td>10 Mbps</td>
<td>100 Mbps</td>
<td>10 Mbps</td>
</tr>
</tbody>
</table>

**TABLE 1: Comparison of WinTrace with UNIX-Based Tools**

Others:

- Copies of WinTrace were distributed to many typical users such as friends and fellow graduate students. Their feedback was taken into consideration in improving the performance and modifying the graphical user interface of the tool.
4.7 Future Work

The following works can be carried out to improve the accuracy and the usability of WinTrace:

- The “Least Square Linear Fit” algorithm used in WinTrace is not very accurate to measure the bandwidth of a link. The “Least Square Linear Fit” algorithm is used to form a line that balances all the minimum RTTs collected by WinTrace (Figure 10). The line usually falls in between those minimal RTT points. But the line we need is a line that passes through all the true minimum RTT points, leaving no true minimum RTTs points below itself. So the algorithm used in WinTrace to find the bandwidth of a link needs to be improved to get more accurate bandwidth estimation.

- WinTrace assumes that the route to a particular remote host does not change during its running time. The routing path from a source host to a remote host might change in some situations, although it does not happen very frequently. The statistical results show that around every twenty minutes, the routing path to a host might change [10]. For better performance, WinTrace needs to have the ability to check whether the path changes or not during its running time. A possible solution could be, first store the results of the “Trace routers” function to an array, and then, every time before measuring the bandwidth of any link, do another “Trace routers” to make sure the link is still in the original path.

- The interface of WinTrace still needs to be improved. It would be nice to show data and results dynamically in some graphical form during execution of WinTrace. Such visualization of data and results will not only be interesting, but also will make it easy for a user to understand how the algorithm estimates the bandwidth of a link.
5. Conclusion

In this project, a tool, named as WinTrace, to measure the bandwidth of a link along an Internet path, is developed. The tool operates under Microsoft Windows environment. It is fully tested for behavior, function validation, and accuracy. It meets all the requirements as expected. According to the algorithm used in implementation of WinTrace, the accuracy of a bandwidth measurement depends on the network traffic. In heavy traffic condition, it may not get an RTT value with zero queuing delay. In order to get an accurate bandwidth measurement on a link, WinTrace should be run when traffic is light. A comparison with existing, similar UNIX-based tools shows that WinTrace performs as good as them in most situations.

WinTrace is the first known MS-Windows application that can measure the bandwidth of Internet links. In order to make it fully useful to those network engineers who prefer MS-Windows platform, it would require further improvement and validation to guarantee accuracy and reliability of the results obtained in most situations.
6. References and Bibliography

1. Matthieu, Ray. “Pathchar”


   http://www-ana.nal.ics.es.osaka-u.ac.jp/~k-matoba/papers/k-matoba00inet-ImprovingBandwidth.pdf


5. Downey, Allen. “Clink: a tool for estimating Internet link characteristics”


## Appendix A: Socket API used in WinTrace

The basic Windows Sockets API used to develop WinTrace are listed below:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bind</td>
<td>Assigns a local name to an unnamed socket.</td>
</tr>
<tr>
<td>Closesocket</td>
<td>Removes a socket from the per-process object reference table. Only blocks if SO_LINGER is set with a nonzero time-out on a blocking socket.</td>
</tr>
<tr>
<td>Getsockopt</td>
<td>Retrieves options associated with the specified socket.</td>
</tr>
<tr>
<td>Htonl</td>
<td>Converts a 32-bit quantity from host-byte order to network-byte order.</td>
</tr>
<tr>
<td>Htons</td>
<td>Converts a 16-bit quantity from host-byte order to network-byte order.</td>
</tr>
<tr>
<td>Inet_addr</td>
<td>Converts a character string representing a number in the Internet standard “.” notation to an Internet address value.</td>
</tr>
<tr>
<td>Inet_ntoa</td>
<td>Converts an Internet address value to an ASCII string in “.” notation that is, “a.b.c.d”.</td>
</tr>
<tr>
<td>Ioctlsocket</td>
<td>Provides control for sockets.</td>
</tr>
<tr>
<td>Recv</td>
<td>Receives data from a connected or unconnected socket.</td>
</tr>
<tr>
<td>Recvfrom</td>
<td>Receives data from either a connected or unconnected socket.</td>
</tr>
<tr>
<td>Select</td>
<td>Performs synchronous I/O multiplexing.</td>
</tr>
<tr>
<td>Send</td>
<td>Sends data to a connected socket.</td>
</tr>
<tr>
<td>Sendto</td>
<td>Sends data to either a connected or unconnected socket.</td>
</tr>
<tr>
<td>Setsockopt</td>
<td>Stores options associated with the specified socket.</td>
</tr>
<tr>
<td>Socket</td>
<td>Creates an endpoint for communication and returns a socket descriptor.</td>
</tr>
</tbody>
</table>
Appendix B: WinTrace File List

**stdafx.h:** An include file for standard system include files, or project specific include files that are used frequently, but are changed infrequently.

**stdafx.cpp:** A source file that includes the standard include file WinTrace.pch.

**OptDlg.h:** A header file for the options dialog.

**OptDlg.cpp:** An implementation file for the options dialog in the application menu.

**Trace.h:** An interface file for the CTrace class.

**Trace.cpp:** An implementation file of the CTrace class containing all the subroutines used to do ping, trace routes and find bandwidth.

**TraceThread.h:** An interface file for the CTraceThread class.

**TraceThread.cpp:** An implementation of the CTraceThread class. Construct and start the trace thread.

**WinTrace.h:** A main header file for the WinTrace application.

**WinTrace.cpp:** A file that defines the class behaviors for the application.

**Resource.h:** A Microsoft Developer Studio generated include file which is used in the file WinTrace.rc.

**WinTrace.rc:** A Microsoft Developer Studio generated file which contains all the dialogs menus and string tables etc.
**WinTraceDlg.h**: A header file for the WinTrace dialog interface.

**WinTraceDlg.cpp**: An implementation file for the dialog interface which contains the application menu, input text box, status bar, list control object, etc.

**WinTrace.hpj**: A configuration file used by HTML Work Shop to create and compile the help topic file.