WinFind, A Tool to Measure the Bandwidth of Asymmetric Links on Internet

GRADUATE PROJECT REPORT

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

by

Krishna Reddy Daggula
Fall 2003

Committee Members

Dr. Dulal C. Kar
Committee Chairperson

Dr. Mario Garcia
Committee Member

Dr. David Thomas
Committee Member
WinFind, A Tool to Measure the Bandwidth of Asymmetric Links on Internet

GRADUATE PROJECT REPORT

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

by

Krishna Reddy Daggula
Fall 2003

Committee Members

Dr. Dulal C. Kar
Committee Chairperson

Dr. Mario Garcia
Committee Member

Dr. David Thomas
Committee Member
ABSTRACT

This Project is the design, development, and implementation of WinFind, a network utility to measure the upstream and downstream bandwidths of an asymmetric link on the Internet. WinFind is a network path characterization tool developed for MS-Windows environment. This tool is developed in Visual Basic.

WinFind is developed assuming network links are asymmetric. It can find all the links present along the path between the sending host and the destination host. With WinFind both upstream and downstream bandwidths of all the links present along the path between the sending host and the destination host can be found.

WinFind finds all the links present along the sending host to destination host by sending Internet Control Message Protocol (ICMP) messages to destination host by incrementing Time To Live (TTL) field from 1 until it reaches the destination host. To get the upstream and downstream bandwidths, ICMP EchoError and ICMP EchoReply messages are sent to hosts to get Round Trip Times (RTTs) with different size packets.

The results obtained from WinFind are very much useful in finding bottlenecks on the Internet.
# TABLE OF CONTENTS

Abstract............................................................................................................................ ii

Table of Contents.............................................................................................................. iii

List of Figures .................................................................................................................. v

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

2. NARRATIVE.................................................................................................................... 3
   2.1 Links, Bandwidth, Bottlenecks, Port and IP Datagram.............................................. 3
   2.2 Basic Mechanism of WinFind.................................................................................... 5
   2.3 Users of WinFind....................................................................................................... 6
   2.4 User Interface........................................................................................................... 7

3. SYSTEM DESIGN........................................................................................................... 13
   3.1 Internet Control Message Protocol (ICMP)............................................................ 13
   3.2 User Datagram Protocol (UDP).............................................................................. 14
   3.3 Algorithms Used to Estimate Bandwidth................................................................. 15
   3.4 Measurement Algorithm......................................................................................... 18
   3.5 The “Least Square Linear Fit” Algorithm............................................................... 19
   3.6 Socket Programming in WinFind............................................................................. 20
   3.7 Implementation of WinFind..................................................................................... 29
   3.8 Implementation of WinFind in Visual Basic........................................................... 32
   3.9 Minimum Hardware Requirements........................................................................... 34
3.10 Minimum Software Requirements .................................................. 34

4. EVALUATION AND RESULTS ...................................................... 35
   4.1 Testing and Evaluation of User Interface .................................. 35
   4.2 Testing of Results ................................................................. 35

5. FUTURE WORK AND CONCLUSION ............................................. 39

6. ACKNOWLEDGEMENTS .............................................................. 41

BIBLIOGRAPHY AND REFERENCES ................................................. 42
Appendix A. Socket API Used in WinFind ........................................ 43
Appendix B. WinFind File List ...................................................... 44
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Nodes and links</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>Bottleneck links</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>TCP/IP layer model</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>WinFind interface</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>WinFind status</td>
<td>9</td>
</tr>
<tr>
<td>2.6</td>
<td>WinFind results</td>
<td>10</td>
</tr>
<tr>
<td>2.7</td>
<td>WinFind help index</td>
<td>11</td>
</tr>
<tr>
<td>2.8</td>
<td>About WinFind</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>User datagram header format</td>
<td>14</td>
</tr>
<tr>
<td>3.2</td>
<td>RTT for two consecutive loops</td>
<td>15</td>
</tr>
<tr>
<td>3.3</td>
<td>WinFind results</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Table 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Comparison of WinFind results with UNIX based tools</td>
<td>38</td>
</tr>
</tbody>
</table>
1. INTRODUCTION AND BACKGROUND

1.1 Background

The Internet has become one of the most important communication media for people. It has made the world a global village. It provides an extensive source of information for students, scientists, and business people. It has become vital to many fields like communication, entertainment, and education. This rapid increase in use of the Internet leads to vast Internet traffic. Hence fast, efficient and reliable data communication has become very important to provide the best quality of service over the Internet. Even a single bottleneck along the network (path) can cause enormous damage in data transfer between any two hosts. Measuring bandwidth can easily detect such bottlenecks easily.

Existing bandwidth measurement tools like pathchar, Bing, and Bprobe assume that the links in a path are symmetric [Wen 2001]. But as ADSL lines, cable modems, satellite links, and 56K modems are getting popular, now it is important to develop bandwidth measurement tools which consider asymmetric links as well.

1.2 Rationale

Even though many people are using the Internet, there is no central control or administration on it. This is certainly the strength in terms of growth and scalability of the Internet, but this may become a weakness if some problem arises. It has become necessary to implement automated tools, which can detect such problems quickly and accurately. There are some tools available for UNIX environment, which can detect such
bottlenecks, but as users are increasingly using Microsoft Windows, it has become necessary to implement such tools in Microsoft Windows environment.

These are some good reasons for choosing this project:

- It requires extensive research on similar tools developed for UNIX environment.
- It requires complete understanding of related network protocols such as UDP (User Datagram Protocol), ICMP (Internet Control Message Protocol), and so on.
- It requires a good understanding of network performance analysis algorithms.
- It requires windows socket programming (WinSock) skills.
- It requires Visual Basic programming skills.
- It requires extensive research on MS-Windows APIs.

1.3 Previous Work

In 1997, Van Jacobson of the LBNL’s (Lawrence Berkley National Laboratory) Network Research Group developed a tool, called pathchar to estimate the performance characteristics of each link along an internet path from a source to a destination host. This tool was developed for UNIX environment [Path 2001]. Similar tools such as Clink, pchar are also available for UNIX environment. These tools can only measure the upstream bandwidths.

There exists one more tool called Bing. Bing is a point-to-point bandwidth measurement tool. Bing determines the real throughput on a link by measuring roundtrip times of ICMP echo request messages for different packet sizes for each end of the link. This tool is also developed for UNIX. It does not take asymmetric links into consideration [Bing 1995].
2. NARRATIVE

2.1 Links, Bandwidth, Bottlenecks, Port, and IP datagram

Network connectivity occurs at many different levels. At the lowest level, a network can consist of two or more computers directly connected by some physical medium, such as a coaxial cable or an optical fiber. Such a physical medium is called a link and computers it connects are often referred as nodes [Larry 2000]. In Figure 2.1, Node A and Node B are connected by Link L.

![Diagram](Figure 2.1 Nodes and links)

Network performance is measured in two fundamental ways: Bandwidth (also called throughput) and latency (also called delay). Bandwidth of a network is given by the number of bits that can be transmitted over the network in a certain period of time, and it is measured in bits/sec. Latency corresponds to how long it takes a message to travel from one end of network to other. Latency is measured in terms of time.

Internet users who communicate with a machine in another country often experience poor performance. This poor performance is caused by bottlenecks due to limited bandwidths of links. Bottlenecks due to links in an Internet depend on the link properties [Ahsan 1999]. 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 2.1.2.

As can be seen in Figure 2.2, except for the two middle links, the capacity of all 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 [Rui 2001].

A port number represents an endpoint or "channel" for network communications. Port numbers allow different applications on the same computer to utilize network resources without interfering with each other. In IP (Internet Protocol) networking, port numbers can theoretically range from 0 to 65,535. Most popular network applications, though, use numbers at the low end of the range (such as 80 for HTTP). The port number is included as a field within the header of each IP packet.
The Internet datagram (IP datagram) is the base transfer packet in the Internet protocol suite. It has a header containing information for IP, and data that is relevant only to the higher level protocols. The IP datagram is encapsulated in the underlying network’s frame, which usually has a maximum length or frame limitation, depending on the hardware used. For Ethernet, this maximum length is about 1500 bytes. Instead of limiting the IP datagram length to some maximum size, IP can deal with fragmentation and re-assembly of its datagrams. However, the actual limit of an IP packet is 64 Kbytes.

2.2 Basic Mechanism of WinFind

The implementation of WinFind is based on TCP/IP protocol (Transmission Control Protocol / Internetworking Protocol). The TCP/IP network model consists of five layers.

- Application Layer
- Transport Layer
- Internet Layer
- Network Layer
- Physical Layer

![TCP/IP network model](image)

Figure 2.3 TCP/IP network model
The Internet layer of the TCP/IP network model is responsible for IP datagrams. This layer is primarily concerned with the handling of information about the routers that are available on the Internet and the paths that can be taken from one router to another. This layer specifies the format of the packets sent across the Internet.

WinFind sends several ICMP packets (probes) to a remote host to measure round-trip time. In the IP header of an ICMP packet, there is a TTL (Time To Live) field. The TTL field specifies the number of links (hops) that a packet can travel. An ICMP message with the TTL field set to successive values starting with 1, is used to find the roundtrip times (RTT) to different hosts on a route to a remote host. Based on the RTT found, the bandwidth of the corresponding link can be estimated by using some estimation algorithms that are described later in the document.

2.3 Users of WinFind

WinFind has many important features with many functionality. It can be used to find whether some host is alive or not. It can be used to find identity of the routers along the path. Most importantly it can be used to find the bandwidth of each asymmetric link along the path to a host. Users of WinFind are:

- Internet researchers
- Internet service providers
- Advanced Internet users
- Home computer users.
- Any common user interested in finding characteristics of the Internet.
2.4 User Interface

The user Interface of WinFind is like any typical windows application’s interface. It contains menus, command buttons, text boxes, list boxes, etc., for using different functionality of WinFind. WinFind user interface looks like Figure 2.4.

![WinFind interface](image)

**Figure 2.4 WinFind interface**

In the text box adjacent to *HostName* a user can enter the Domain Name System (DNS) name of the destination host. When the *Find BW* command button is pressed, the *IP Address* textbox displays the corresponding IP address of that host. If the DNS name is not available, the user can enter the IP address in the *IP Address* textbox. This address
can be either in string format or byte order format. When the Find BW command button is pressed, the corresponding DNS name of that host is displayed in the HostName text box. If the user inputs a wrong DNS name or an IP address, it displays ‘host not alive message’. The No of Packets field is the number of probes or trials a user wants to try. Generally as the number of packets increases, accuracy of measurement increases. The NoAttempts field is the number of probes that a user wants to send for each packet size. A click on the FindBW button starts probing the Internet path and it displays the status of the WinFind in the progress bar and also it displays the runtime messages and error messages in the list box as shown in Figure 2.5. In the end it displays the results i.e, the upstream and downstream bandwidths of all intermediate hosts with their DNS names displayed with measured bandwidths as shown in the Figure 2.6.
Figure 2.5 WinFind status
The user interface of WinFind also contains the File, Edit and Help menu options. The File menu has Save, Print and Exit options. A user by clicking on the Save option or pressing Ctrl + S can save the output of the program to ‘WinFind_results.txt’ file. Similarly, a user by clicking on the Print option or Ctrl + P can print results to a default printer. A user can close the application by clicking on the Exit option.

The Edit menu has Clear and Set Default options. The Clear option clears all the input textboxes. The Set Default option sets all input textboxes to default values. The
Help menu contains Contents and About WinFind options. Selecting Contents displays ‘WinFind Help Index’ as shown in the Figure 2.7, which shows information about how to use WinFind. Selecting About WinFind displays ‘About WinFind’ dialogue box as shown in Figure 2.8, which shows information about version, release, and copyright details of WinFind.

Figure 2.7 WinFind help index
Figure 2.8 About WinFind
3. SYSTEM DESIGN

3.1 Internet Control Message Protocol (ICMP)

Internet Control Message Protocol (ICMP) is a required protocol tightly integrated with IP. Functions of ICMP are, announcing network errors (such as a host being unreachable), announcing network congestion, assisting troubleshooting (ICMP supports an echo function, which sends a packet on a round-trip between two hosts), and announcing timeouts (when IP packet’s TTL field drops to zero) in the communication environment [ICMP 2002].

Each ICMP message is encapsulated in an IP packet. Since ICMP uses IP, ICMP packet delivery is unreliable. To avoid the infinite regress, no ICMP messages are sent about ICMP error messages. So the host cannot count on receiving ICMP packets for every network problem. There are about a dozen types of ICMP messages. The ECHO REQUEST and ECHO REPLY messages are used to inform the sender that a given destination is reachable and alive.

Different types of ICMP messages that WinFind uses are:

- Type 0 Code 0 for echo reply message,
- Type 8 Code 0 for echo message,
- Type 11 Code 0 for time to live exceeded in transit,
- Type 13 Code 0 for timestamp message, and
- Type 14 Code 0 for timestamp reply message.
3.2 User Datagram Protocol (UDP)

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) is used as the underlying protocol [Larry 2000]. UDP sits directly on top of the base Internet Protocol (IP).

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented and delivery, duplicate protection, and order of delivery are not guaranteed. UDP's main purpose is to abstract network traffic in the form of datagrams.

A datagram comprises one single "unit" of binary data; the first eight (8) bytes of a datagram contain the header information and the remaining bytes contain the data itself. The UDP header consists of four fields of two bytes each as shown in Figure 3.1

- Source port number
- Destination port number
- Datagram size
- Checksum

<table>
<thead>
<tr>
<th>0</th>
<th>15</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td>Checksum</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.1 User datagram header format
UDP port numbers allow different applications to maintain their own "channels" for data. The sending application (that could be a client or a server) sends UDP datagrams through the source port, and the recipient of the packet accepts this datagram through the destination port. Some applications use static port numbers that are reserved for or registered to the application. Other applications use dynamic (unregistered) port numbers. Because the UDP port headers are two bytes long, valid port numbers range from 0 to 65535 and by convention, values above 49151 represent dynamic ports.

The datagram size is a simple count of the number of bytes contained in the header and data sections. Because the header length is fixed, this field essentially refers to the length of the variable-sized data portion also called payload. The maximum size of a datagram varies depending on the operating environment.

UDP checksums work as a safety feature. The algorithm is not foolproof, but it is effective in many cases. In UDP, checksumming is optional. Turning it off squeezes a little extra performance from the system.

3.3 Algorithms Used to Estimate Bandwidth

Let \( T_i \) denote the RTT of an ICMP echo/reply message of packet \( B \) bytes, \( v_i \) denote the link upstream speed and \( v'_i \) denote the link downstream speed between hop \( i - 1 \) and hop \( i \), where \( i \) is the number of hops between sender (hop 0) and receiver (hop \( i \)). Figure 3.2 shows a network with hops 0, 1, and 2.

![Figure 3.2 RTT for two consecutive hops](image-url)
From Figure 3.2, RTT $T_2$, can be obtained as:

$$T_2 = \frac{B}{v_1} + \frac{B}{v_1} + \frac{B}{v_2} + \frac{B}{v_2} + \text{a const} \quad [\text{Wen 2001}]$$

(1)

In general, for any $i$

$$T_i = B \sum_{j=1}^{i} \left( \frac{1}{v_j} + \frac{1}{v_j} \right) + \text{a const}$$

(2)

Where $\text{a const}$ is a constant that depends on $i$. It is the sum of the propagation delay and the processing delay at each hop. Without any loss of generality, it is assumed that queuing delay is 0.

Let $K_i$ denotes the slope of linear function $T_i(B)$,

Then $K_i = \sum_{j=1}^{i} \left( \frac{1}{v_j} + \frac{1}{v_j} \right)$

(3)

$$\Rightarrow K_i - K_{i-1} = \frac{1}{v_i} + \frac{1}{v_i}$$

(4)

The above formula does not give value of $v_i$ or $v_i'$, but if any of these two values is known, other value can be found easily. RTT is the sum of transit times in both directions ($T_{to} + T_{from}$). RTT depends on link speeds in both the directions and packet size $B$. If the reply message is forced to have a constant size $D$, then $T_{from}$ no longer depends on $B$ but only $v_i'$ for any $i$. Since $v_i$ is also constant for any $i$, $T_{from}$ is also a constant. Therefore $v_i$ can be solved.

If we send an echo packet to the $(i+1)$ th router with TTL = $i$; at $i$ th router, it will bounce back as an ICMP time-exceeded message. Let this RTT be $T_i'$. Then for $i = 2$, 

16
\[ T_2' = \frac{B}{v_1} + \frac{D}{v_1} + \frac{B}{v_2} + \frac{D}{v_2} + \text{a const} \]

\[ = \frac{B}{v_1} + \frac{B}{v_2} + \text{a const} \tag{5} \]

\[ T_i' = \sum_{j=1}^{i} \left( \frac{B}{v_j} + \frac{D}{v_j} \right) + \text{a const} \]

\[ = B \sum_{j=1}^{i} \left( \frac{1}{v_j} \right) + \text{a const} \tag{6} \]

Let \( K_i' \) denotes the slope of the linear function \( T_i'(B) \),

Then \( K_i' = \sum_{j=1}^{i} \left( \frac{1}{v_j} \right) \) \tag{7}

\[ \Rightarrow K_i' - K_{i-1}' = \frac{1}{v_i} \]

\[ v_i = \frac{1}{(K_i' - K_{i-1}')} \tag{8} \]

But from equation (4), we know

\[ K_i - K_{i-1} = \frac{1}{v_i} + \frac{1}{v_j} \]

\[ = (K_i' - K_{i-1}') + \frac{1}{v_i} \tag{9} \]

Thus,

\[ \frac{1}{v_i} = (K_i - K_{i-1}) - (K_i' - K_{i-1}') \]

\[ v_i' = \frac{1}{(K_i - K_{i-1} - (K_i' - K_{i-1}'))} \tag{10} \]
Therefore, from (8) and (10), both the upstream and the downstream speeds can be estimated.

3.4 Measurement Algorithm

These following steps are used to measure the upstream and the downstream speeds.

**Step 1.** For the first hop on the path from source to destination, send an ICMP echo/reply message of size $B$ to measure the round trip time (RTT). Repeat the probing process for some number of times (minimum enough to find RTT) to obtain minimum RTT from the measurements. Thus it minimizes the effect of the queuing delay in the measurement. Record the minimum RTT for the packet size.

**Step 2.** Repeat step 1 for different packet sizes from some allowable minimum to some allowable maximum size. Record the minimum RTTs for all different packet sizes.

**Step 3.** Using some good estimation algorithm fit a straight line on the minimum RTT versus packet size curve. From the equation 4, the slope of the fitted straight line gives the sum of the inverse of upstream and downstream bandwidths of the hop.

**Step 4.** Send an ICMP echo message of size $B$ to immediate next hop to first hop and make it bounce back at first hop. This makes the downstream trip time constant. Repeat the probing process for some number of times and record minimum RTT from the measurements.

**Step 5.** Repeat step 4 for different packet sizes from some allowable minimum to some allowable maximum size. Record the minimum RTTs for all different packet sizes.
**Step 6.** As in step 3, fit a straight line on the minimum RTT versus packet size curve. From equation 8, the slope of the fitted straight line gives the inverse of the upstream bandwidth of the 1st hop.

**Step 7.** From the results obtained in step 3 and step 6, the downstream bandwidth for 1st hop can be calculated (equation 10).

**Step 8.** For the other hops, the same steps can be followed, but in each case, time measurements on two consecutive hops are needed in order to find the bandwidth of the immediate next hop in the pair.

### 3.5 The “Least Square Linear Fit” algorithm

WinFind collects RTTs for different size packets and for each packet it sends some number of probes to eliminate queuing delays. The roundtrip time for each packet size is the minimum RTT obtained. From the Figure 3.3, the line drawn on the graph is the line joining the minimum RTTs for different packet sizes. Using “Least Square Linear Fit” algorithm a linear regression line is drawn using those points and slope of the line is obtained.

The “Least Square Linear Fit” algorithm assumes that the best fit curve of a given type is the curve that has sum of the deviations squared from a given data set as minimum. This algorithm fits all the data obtained in to a straight line as shown in the Figure 3.4. Bandwidths can be measured by using the slope of the fitted line.
Figure 3.3 Packet size Vs RTT for a single link

Figure 3.4 Packet Size Vs RTT for multiple links
3.6 Socket Programming in WinFind

In 1982, Berkeley Software Distribution’s 4.1c BSD system for VAX introduced sockets as an interface for communication between local processes. In 1986, the Berkeley UNIX version 4.3 BSD extended the sockets interface for use over the TCP/IP and XNS protocols for IPC (Inter Process Communication). This extension allowed two processes to communicate with each other, whether they were present on the same computer or two different computers on the Internet. Soon, most UNIX vendors supported the socket interface, and such a wide acceptance made the socket interface a standard of network computing in the UNIX environment. With the popularity of Windows, a consortium of companies including Microsoft, proposed an industry-wide socket standard for the Windows 3.x and Windows NT environments. This standard is called Windows Socket Interface or Winsock [Sinha 1996].

Windows Sockets (Winsock) provides a general-purpose networking application programming interface (API) based on BSD socket interface. Winsock is designed to run efficiently on Windows OSs while maintaining compatibility with BSD. Winsock is an interface, not a protocol. It provides a protocol-independent interface fully capable of supporting emerging networking capabilities, such as real-time multimedia communications. As an interface, it is used to discover and utilize the communication 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 communication link.

Winsock provides access to multiple transport protocols allowing a user to create applications that support multiple socket types. Winsock's support of multiple protocols
also includes IPv6. Following the Windows Open System Architecture (WOSA) model, Winsock defines a standard service provider interface (SPI) between the API, with its functions exported from the Winsock DLL, Ws2.dll, and the protocol stacks. Users can use the Winsock SPI to create his own transport service provider or extend an existing transport service provider by implementing a Layered Service Provider (LSP).

3.6.1 Basic socket structures in Windows sockets

There are three basic structures in Windows Socket API: SOCKADDR_IN, HOSTENT and WSADATA which are used in WinFind. Their declarations in Visual Basic are described below:

**SOCKADDR_IN**

In the Internet address family, this structure is used by Windows Sockets to specify a local or remote endpoint address to which to connect a socket and the structure is:

```vbat
Public Type sockaddr_in
    sin_family As Integer
    sin_port As Integer
    sin_addr As Long
    sin_zero(1 To 8) As Byte
End Type
```

**HOSTENT**

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

```vbat
Type HOSTENT
    hName As Long
    hAliases As Long
    hAddrType As Integer
    hLength As Integer
End Type
```
hAddrList As Long
End Type

**WSADATA**

The **WSADATA** structure stores information about the Windows Sockets implementation being used by WinFind. Most of the structure's contents refer to the version of Winsock the program is interfacing with. The structure is

Private Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription(0 To 255) As Byte
    szSystemStatus(0 To 128) As Byte
    iMaxSockets As Integer
    iMaxUdpDg As Integer
    lpVendorInfo As Long
End Type

### 3.6.2 Visual Basic declaration of Winsock API functions

Winsock API functions can be used in Visual Basic by declaring functions using “ws2_32.dll”.

**gethostbyname()**

In order to get all the IP addresses of a host we can use the gethostbyname Winsock API function. The function declaration is:

---

`Declare Function gethostbyname _
Lib "ws2_32.dll" (ByVal host_name As String) As Long`

---

This function receives a host name as an argument and return a pointer to the hostent structure. If the returned value is 0, the function failed to get the IP address of the host.
gethostbyaddr()

The **gethostbyaddr** Winsock API function is used to retrieve a host name given its IP address. The function declaration is:

\[
\text{Declare Function gethostbyaddr Lib "ws2_32.dll" (addr As Long, _}
\text{ ByVal addr_len As Long, _}
\text{ ByVal addr_type As Long) As Long}
\]

The function returns a pointer to the HOSTENT structure.

WSAStartup()

The WSAStartup function initiates use of the Windows Sockets DLL by a process. The function declaration is:

\[
\text{Declare Function WSAStartup Lib "WSOCK32" (ByVal wVersionRequested As Long,}
\text{ lpWSADATA As WSAData) As Long}
\]

The parameter \(wVersionRequested\), is the latest version of Windows Sockets support that the caller can use and \(lpWSADATA\), is a pointer to the WSADATA data structure to receive details of the Windows Sockets implementation.

WSAStartup returns zero if successful. Otherwise, it returns error code.

WSACleanup()

The WSACleanup function terminates use of the Windows Sockets DLL. The function declaration is:

\[
\text{Declare Function WSACleanup Lib "WSOCK32" () As Long}
\]

The return value is zero if the operation was successful. Otherwise, the value SOCKET_ERROR is returned.
**WSAGetLastError()**

The WSAGetLastError function gets the error status for the last operation that failed. The function declaration is:

*Declare Function WSAGetLastError Lib "WSOCK32" () As Long*

The return value indicates the error code for this thread’s last Windows Sockets operation that failed.

**IcmpCreateFile()**

This function creates a handle on which Internet Control Message Protocol (ICMP) requests can be issued. The function declaration is:

*Declare Function IcmpCreateFile Lib "icmp.dll" () As Long*

An ICMP handle indicates success. INVALID_HANDLE_VALUE indicates failure.

**IcmpSendEcho()**

This function sends an Internet Control Message Protocol (ICMP) echo request, and returns one or more replies. The function declaration is:

*Private Declare Function IcmpSendEcho Lib "ICMP" (ByVal IcmpHandle As Long, ByVal DestAddress As Long, ByVal RequestData As String, ByVal RequestSize As Integer, RequestOptns As IP_OPTION_INFORMATION, ReplyBuffer As IP_ECHO_REPLY, ByVal ReplySize As Long, ByVal TimeOut As Long) As Boolean*

The number of replies received and stored in the reply buffer indicates success. A return value of zero indicates failure of the function.

**IcmpCloseHandle()**

This function closes an Internet Control Message Protocol (ICMP) handle opened by IcmpCreateFile. The function declaration is:
Declare Function IcmpCloseHandle Lib "icmp.dll" (ByVal HANDLE As Long) As Boolean

A return value TRUE indicates success. A FALSE Value indicates failure.

inet_addr()

The Winsock API inet_addr() subroutine is used to convert an IP address in the string format to the Long value. The Visual Basic declaration of the function is:

Declare Function inet_addr Lib "ws2_32.dll" (ByVal cp As String) As Long

The function receives the IP address in dotted decimal notation and returns the address as a Long value that can be passed to other Winsock API functions.

WSASocket()

The WSASocket function creates an endpoint for communication and returns a descriptor, which is a small integer used to reference the socket. The function declaration is:

Declare Function WSASocket Lib "ws2_32.DLL" Alias "WSASocketA" (ByVal af As Long, ByVal s_type As Long, ByVal protocol As Long, lpProtocolInfo As Any, ByVal g As Long, ByVal dwFlags As Long) As Long

closesocket()

The closesocket function closes an existing socket. The function declaration is:

Declare Function closesocket Lib "ws2_32.dll" (ByVal s As Long) As Long

inet_addr()

The inet_addr interprets a null-terminated character string, pointed to by cp, that represents numbers in the Internet standard dotted decimal notation and returns a corresponding Internet address. The dotted decimal string can contain up to four components. The function declaration is:
The `inet_ntoa` function takes an Internet address, `inn`, and returns a pointer to a null-terminated string representing the address in dotted decimal notation. The dotted decimal string has four components. The host address is specified in network byte order because it is contained in a `struct in_addr`. The function declaration is:

```vba
Public Declare Function inet_ntoa Lib "ws2_32.dll" (ByVal inn As Long) As Long
```

The `htons` macro converts an unsigned short integer `hostshort` from the host byte order to the network byte order. The function declaration is:

```vba
Public Declare Function htons Lib "ws2_32.dll" (ByVal hostshort As Integer) As Integer
```

The `htonl` macro converts an unsigned long integer `hostlong` from the host byte order to the network byte order. The function declaration is:

```vba
Public Declare Function htonl Lib "ws2_32.dll" (ByVal hostlong As Long) As Long
```

The `ntohl` macro converts an unsigned long integer `netlong` from the network byte order to the host byte order. The function declaration is:

```vba
Public Declare Function ntohl Lib "ws2_32.dll" (ByVal netlong As Long) As Long
```

The `ntohs` macro converts an unsigned short integer `netshort` from the network byte order to the host byte order.
Public Declare Function ntohs Lib "ws2_32.dll" (ByVal netshort As Integer) As Integer

3.6.3 Other WINDOWS System APIs

System functions provided by MS-WINDOWS that are used in implementation of WinFind are:

GetTickCount

The GetTickCount function retrieves the number of milliseconds that have elapsed since Windows was started. The function declaration is:

Declare Function GetTickCount Lib "kernel32" Alias "GetTickCount" () As Long

If the function succeeds, the return value is the number of milliseconds that have elapsed since Windows was started.

QueryPerformanceCounter

The QueryPerformanceCounter function retrieves the current value of the high-resolution performance counter, if one exists. The function declaration is:

Declare Function QueryPerformanceCounter Lib "kernel32" (lpPerformanceCount As LARGE_INTEGER) As Long

If the installed hardware supports a high-resolution performance counter, the return value is nonzero. If the installed hardware does not support a high-resolution performance counter, the return value is zero.

QueryPerformanceFrequency

The QueryPerformanceFrequency function retrieves the frequency of the high-resolution performance counter, if one exists. The function declaration is:

Declare Function QueryPerformanceFrequency Lib "kernel32" (lpFrequency As LARGE_INTEGER) As Long
If the installed hardware supports a high-resolution performance counter, the return value is nonzero. If the installed hardware does not support a high-resolution performance counter, the return value is zero.

### 3.7 Implementation of WinFind

Implementation of WinFind consists of three major parts. Those are validation of input data, collection of data, and measurement of bandwidths.

#### 3.7.1 Validation of Input data

Data entered by user through text boxes are checked for input validation. The destination host name entered in the text box *HostName* is converted from its DNS name to the IP address using `gethostbyname()` WinSock API and displayed in *HostIPAddress* the textbox.

#### 3.7.2 Collection of Data

Collection of data consists of mainly three parts. Those are finding intermediate links, collecting upstream RTTs and collecting total RTTs.

**Finding Intermediate Links**

To get the bandwidth of all asymmetric links on the internet first we need to know all the links present in between source and destination host. Using `IcmpCreateFile()` function, a handle on which Internet Control Message Protocol (ICMP) requests can be issued is created. With TTL field set from 1 to 30 `ICMPEchoRequest` messages are sent to the destination host. From the `EchoReply` messages generated from the hosts, all the
intermediate hosts’ addresses are found and are stored in HostArray array and number of hosts in TimeToLive variable.

EchoReply.Address values are stored in HostArray as shown in the code.

\[ \text{HostArray(TimeToLive)} = \text{EchoReply.Address} \]

**Collecting Upstream RTTs (Data1)**

Using GetEchoErrorData() function ICMP SendEcho() messages are sent to all the addresses present in HostArray. To each host \text{NoPackets} packets and each packet \text{NoAttempts} times are sent to the host. RTT for each probe is measured using MicroTimer() function. This function works the same way as in finding Intermediate Links except that here all RTT’s are measured.

**Collecting Total RTTs (Data2)**

Using CollectDataAll() function ICMP packets are sent to each host with \text{NoPackets} packet sizes and each packet \text{NoAttempts} of times and RTTs are measured.

Some of the headers used by this function are:

**IP Header**

\[
\begin{align*}
\text{Type} & \text{ IpHeader} \\
& \text{h_len As Byte} & \text{length of the header} & \text{Version of IP} \\
& \text{tos As Byte} & \text{Type of service} \\
& \text{total_len As Integer} & \text{total length of the packet} \\
& \text{ident As Integer} & \text{unique identifier} \\
& \text{frag_and_flags As Integer} & \text{flags} \\
& \text{ttl As Byte} & \\
& \text{proto As Byte} & \text{protocol (TCP, UDP etc)} \\
& \text{CheckSum As Integer} & \text{IP checksum} \\
& \text{sourceIP As Long} & \\
& \text{destIP As Long} & \\
\end{align*}
\]

\text{End Type}
ICMP Packet Header

Type IcmpHeader
   i_type As Byte
   i_code As Byte
   i_cksum As Integer
   i_id As Integer
   i_seq As Integer
   timestamp As Double
End Type

The rawsockets are initialized with WSASocket(). It creates a descriptor, which is a small integer used to reference the socket as shown in the code bellow.

sockRaw = WSASocket(AF_INET, SOCK_RAW, IPPROTO_ICMP, ByVal 0, 0, WSA_FLAG_OVERLAPPED)

The Windows Sockets setsockopt() function sets a socket option as in the following code.

bread = setsockopt(sockRaw, SOL_SOCKET, SO_RCVTIMEO, timeout, LenB(timeout))

Using checksum function, the checksum of the ICMP packet to be sent is calculated using the following statement:

Checksum icmp_data, datasize

The Windows Sockets sendto() function sends data on a connected socket to the destination host as shown below:

bwrote = sendto(sockUdp, udp_data(0), datasize, 0, dest, LenB(dest))

The Windows Sockets recvfrom() function receives data on a socket using the following statement:

bread = recvfrom(sockUdp, recvbuf(0), MAX_PACKET, 0, from, fromlen)

The closesocket function closes socket created by WSASocket() function using the following statement.

closesocket sockUdp
3.7.3 Measurement of Bandwidths

After measuring all the RTTs, the slope of the RTT vs. Packet Size graph is measured for Data1 and Data2 using FindSlope() function which uses regression principle in determining the slope. The main objective of using this technique is to eliminate or minimize queuing delays.

FindBW() function calculates the bandwidths of all links using the equations mentioned in-section 3.4. Finally results calculated are displayed to the list box.

3.8 Implementation of WinFind in Visual Basic

3.8.1 Creating a project

We begin creating this application by choosing New Project from the File menu, then selecting the Standard EXE option in the New Project dialog box. Visual Basic creates a new project and displays a new form.

There are three main steps to creating an application in Visual Basic:

1. Create the interface.
2. Set properties.
3. Write code.

3.8.2 Creating the Interface

The first step in creating an application in Visual Basic is to create the interface, the visual part of the application with which the user will interact.

Forms, Controls, and Menus

Forms and controls are the basic building blocks used to create the interface. They are the objects that we work with to build our application. We can use forms to add
windows and dialog boxes to our application. We can also use them as containers for items that are not a visible part of the application's interface [VB 2002].

Forms are objects that expose properties, which define their appearance, methods which define their behavior, and events which define their interaction with the user. By setting the properties of the form and writing Visual Basic code to respond to its events, we customize the object to meet the requirements of our application. Controls are objects that are contained within form objects. Each type of control has its own set of properties, methods, and events that make it suitable for a particular purpose. Some of the controls we can use in our applications are best suited for entering or displaying text.

The menu bar appears immediately below the title bar on the form and contains one or more menu titles. When a user clicks a menu title (such as File), a menu containing a list of menu items drops down. Menu items can include commands (such as New), separator bars, and submenu titles.

3.8.3 Setting properties

The next step is to set properties for the objects we have created. The Properties window provides an easy way to set properties for all objects on a form. To open the Properties window, a user chooses the Properties Window command from the View menu, clicks the Properties Window button on the toolbar, or uses the context menu for the control.

3.8.4 Writing Code

The Code Editor window is where a user writes Visual Basic code for an application. Code consists of language statements, constants, and declarations. Using the Code Editor window, we can quickly view and edit any of the code in our application. To
open the Code window, a user needs to do double-click the form or control for which he or she chooses to write code or from the Project Explorer window, select the name of a form or module, and chooses the View Code button.

### 3.8.5 Running the Application

To run the application, a user chooses Start from the Run menu, or clicks the Start button on the toolbar, or presses F5.

### 3.9 Minimum Hardware Requirements

WinFind requires a Pentium class or equivalent computer with a minimum of 16 MB of random access memory (RAM) and 500 MB of free hard disk space. A Pentium III with 64 MB RAM or higher configuration is recommended. A stable Internet connection of minimum 28 kbps speed is required.

### 3.10 Minimum Software Requirements

WinFind is able to run under MS-WINDOWS 95/98/Me/2000/XP. Internet Explorer 4.0 or above is required to see the help files.
4. EVALUATION AND RESULTS

4.1 Testing and Evaluation of User Interface

After developing the application, it was tested for User Interface Design. It was given to fellow graduate students, friends and users in academic institutions as well as in industry for usability evaluation.

The application is very simple to install on a computer and it is very easy to use. The interface of this tool is like any typical windows application. Since it is provided with text boxes, list boxes and command buttons etc., any home user can use this tool without much effort. As the graphical user interface of this tool is superior to the UNIX based tools whose interface is mainly command line, the usability of this tool is better than UNIX based tools. The results of the test were satisfactory.

4.2 Testing of Results

After developing this application, it is tested for Windows Application Behavior, Application Functionality validation and for Functions Accuracy numerous times.

4.2.1 Windows Application Behavior

- Input values from the text boxes are checked for validity and also confirmed that application is using only valid results.

- The “Save (Ctrl + s)” option in File menu is tested. It is saving the results to a text file.

- The function of “FindBW” command button is tested. It is disabled while application is running.
• The progress bar is tested for displaying the progress of the application. It is accurately showing the progress of the application.

• The application interface is tested for keyboard response.

• The application window position on the screen is tested. It is verified that the window is always at the center to the screen.

• WinFind is tested on MS-WINDOWS 95/98/2000/XP and works fine on Pentium class machines.

4.2.2 Application Functionality Validation

• WinFind is tested for invalid IP addresses and invalid DNS names. When invalid values are entered it prompts an error message and stops the application.

• Menu options provided in the menu bar are working fine. When the “Save” button is pressed, it saves the output to a text file and when the “Print” button is pressed, it prints the output.

• It is confirmed that all the options set in the options field are successfully passed to the application.

• The “Contents” option in the “Help” menu is tested. It provides the help files that contain all information about the usage of WinFind.

4.2.3 Functions Accuracy

• WinFind is tested for links along an Internet path and results are compared with Tracert program available on MS-Windows. Test shows that WinFind is able to identify all intermediate links correctly.

• Since WinFind uses micro second level timer, RTTs found are very accurate.
• UNIX-based applications, Pathchar and Clinks, are run on two sides of selected network and both upstream and downstream bandwidths are measured.

• Results of the WinFind are compared with results obtained with Pathchar and Clinks as shown in the Table 1. The screen shot of the application, with results obtained is shown in the Figure 3.3.

• WinFind is distributed to colleagues and friends to test for the accuracy of the results. It is confirmed that results obtained are comparable with actual values.

![Figure 3.3 WinFind results](image)
Table 1 Comparison of WinFind results with UNIX based tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Hop1 (FDDI)</th>
<th>Hop2 (FDDI)</th>
<th>Hop2 (Ethernet)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Upstream (Mbps)</td>
<td>Downstream (Mbps)</td>
<td>Upstream (Mbps)</td>
</tr>
<tr>
<td>Pathchar</td>
<td>47.7</td>
<td>32.2</td>
<td>40.4</td>
</tr>
<tr>
<td>Clink</td>
<td>47.2</td>
<td>10.6</td>
<td>51.6</td>
</tr>
<tr>
<td>WinFind</td>
<td>44</td>
<td>38</td>
<td>72</td>
</tr>
</tbody>
</table>

The experiment is conducted on a known campus wide network along a path with three links. The first two links are FDDI (Fiber Distributed Data Interface), and the last one is Ethernet. The results are shown in Table 1. As can be seen from the Table 1, the results of WinFind are comparable with UNIX based tools. It can be observed that all the tools fall short in estimating the exact bandwidth for any hop. The difference between actual bandwidth and the measured bandwidth of links can be justified from the fact that a measured bandwidth by a tool is in fact the IP (Internet Protocol) layer bandwidth. Overhead of bits of the lower layers are simply ignored in the estimation of bandwidth.
5. FUTURE WORK AND CONCLUSION

WinFind is a Microsoft Windows based networking application. It is used to find the bandwidths of all asymmetric links present along the destination host. This tool is tested for behavior, function validation, and accuracy. It meets all the proposed requirements satisfactorily. According to the algorithm and APIs used in the implementation of WinFind, the accuracy of the results depends on the network traffic. Under normal traffic it performs well as other tools existed in UNIX environment. Under heavy network traffic conditions, it may not get RTT with minimum queuing delays.

There is no existing tool in a Windows environment to measure bandwidths of asymmetric links on the Internet. WinFind is the first known MS-Windows application for measuring the bandwidths of the links in both upstream and downstream directions from one end of the path. Using WinFind one can easily find bottlenecks present on the Internet. This tool is useful to researchers, service providers and home users who are interested to find bottlenecks present along the path.

There are some limitations in using this tool in measuring bandwidths. Measuring the bandwidths of multiple links is not straightforward as in the case of single link. There may be some inaccuracy in results due to various sources of error present along the path, such as:

- A slow intermediate link in the path can cause problem in measuring the bandwidth of a fast link down the path.
- The path Maximum Transmission Unit (MTU) can limit the accuracy in measuring the bandwidths of a link.
• Presence of hidden links or ATM networks in the path, change of route during the experiment, and presence of multiple queues in a router can limit the accuracy of the results.

• If the target host is not online or protected by a firewall system that prevents replying to ICMP echo reply messages, this tool can not measure the bandwidth of that link.

   To make this tool more useful to network engineers and researchers, it would require further improvements on accuracy and reliability of results under heavy network traffic conditions.
6. ACKNOWLEDGEMENTS

I wish to express my profound gratitude to Dr. Dulal C. Kar, who has been a great inspiration for me during my graduate studies, for his invaluable support, encouragement and technical expertise rendered throughout this Project. It is of immense pleasure and privilege to work under his guidance. I express my sincere thanks to Dr. Mario Garcia, for his precious time, valuable advice and suggestions. I am grateful to Dr. David Thomas, for his counsel and patient reviewing of my Project Proposal. I express my special heartfelt thanks to Gail Sutton, Brien Nicolau, B.J. Sparks, and Ibanez Herminio for their cooperation, encouragement, sharing of the resources and valuable time. I would like to thank my beloved parents for their unending love, affection and support throughout my career.
BIBLIOGRAPHY AND REFERENCES


[Bing 1995] Pierre Beyssac, Bing. Available from


[ICMP 2002] ICMP messages. Available from


APPENDIX A: Socket API Used in WinFind

The basic Windows Sockets API that will be used to develop WinFind 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: WinFind File List

**WinFind.vbp**: Visual Basic Project file for WinFind. This is the main file to start the application.

**TraceClass.cls**: Visual Basic Class module which controls all network functionality of the application.

**WinSock.bas**: Visual Basic module for WinSock control.

**WinFind.frm**: Visual Basic form file for WinFind interface. It controls the functionality of the application.

**frmAbout.frm**: Visual Basic form file for About WinFind form.

**WINFIND.CHM**: Compiled HTML file for WinFind help topics.