# Profiling the Performance of TCP/IP on Windows NT

P.Xie, B. Wu, M. Liu, Jim Harris, Chris Scheiman

## Abstract

*This paper presents detailed network performance measurements of a prototype implementation of the TCP/IP network software on Windows NT. The measurements include latency, throughput, and CPU utilization of the protocol stack and for some key operations within the stack.*

## 1 Introduction

Performance measurements of TCP/IP network software on Unix platforms have long existed [1, 2, 3, 4]. Similar studies on the Windows NT platform, however, have not been widely available. The main reason is the limited access to the NT kernel and network software source code [5]. In 1998, the Microsoft Research group released the implementation and the source code of MSR IPv6, a prototype IPv6 protocol stack for Windows NT [6], " for testing, research, and educational purposes" , adding that " (the) implementation should prove useful to people wishing to use Windows NT as a platform for networking research or education." This study makes use of the source code thus made available to perform measurements and analyses analogous to those described in [4] on a test bed comprised of NT workstations.

## 2 The MSR IPv6 Protocol Stack

The MSR IPv6 software was the result of a project at Microsoft Research initiated as a learning experience in 1996. The source code was first released in March, 1998. It employs the Windows NT networking architecture and implements the TCP/IP protocol as a dynamically loadable device driver [7]. It was based on the TCP/IP source code for Windows NT 4.0, which was incrementally modified to support IPv6. As such, the IP-layer code was rewritten, but the TCP and UDP layers retain much of the original code base [6]. The stack was a "single stack", supporting only IPv6, as opposed to a hybrid stack supporting both IPv4 and IPv6. The code was organized as three layers: the link layer, the core network layer, and upper-layer protocols. The core network layer modules support IPv6, including modules for the basic operations send, receive, fragmentation, reassembly, header processing, neighbor discovery, and routing. The upper-layer supports transport protocols such as TCP and UDP. The software is not a complete implementation [1]of IPv6, however, it does support the basic functionalities. Although we were aware that the implementation is not the official Microsoft release, the availability of its source code made it feasible for us to perform measurements using source-level instrumentation. Our assumption is that the behavior of the MSR IPv6 network software is indicative of that of the official release.

## 3 Experimental Setup

We performed instrumentation on the MSR IPv6 protocol stack on a test bed configured as follows:

| | |
|---|---|
| CPU: | Pentium 200 MHz |
| Memory: | SRAM 64MB |

| | |
|---|---|
| OS: | Windows NT 4.0 Workstation Build 1381, Service Pack-3 |
| Network: | 100BaseT Fast Ethernet |
| NIC: | 3COM 3C905 Etherlink-II |

Two personal computers were employed. One serves as a *target* machine on which the instrumented version of the protocol stack was compiled and installed. Another machine, called the *instrumentation* host, runs a free-build version of the MSR IPv6 stack and *WinDbg*, the Windows NT debugger. In addition to the Ethernet connection, the two machines are linked using a serial cable. Test data at the application layer are sent from the target host to the instrumentation host through the Ethernet connection. At the same time, the serial cable connection allows the target host to pass instrumentation data to the instrumentation host directly in kernel mode. Figure 1 illustrates the experimental system.

## 4 Measurements

We were interested in three types of measurements: The latency or processing time within the stack, data throughput, and CPU utilization.

We define latency as the time required for a block of data to be transmitted across a network connection from the sending host to the receiving host. There are two types of latency that can be measured: *start* and *stop* latency. Start latency is defined as the interval of time between (i) when the first bit of a data stream reaches the top of the protocol stack and (ii) when the first bit of data emerges from the bottom of the stack. Stop latency, on the other hand, is the interval between (i) when the first bit of data in a stream reaches the top of the protocol stack and (ii) when the last bit of data emerges from the bottom of the stack. (The start and stop latencies, as defined here, correspond to the notion of 'wire arrival time' and 'wire exit time' described in the Internet Society's RFC 2330 [8].)

Thus defined, the start latency is an indication of the processing overhead imposed by the stack, while the stop latency measures the processing overhead as well as the data transmission delays experienced by the data block.

To obtain the measurements of latencies, we modified the source code of the MSR IPv6 to collect timestamps at strategic points. This is done by inserting instructions written in the Intel RDTSC (ReaD Time Stamp Counter) assembly language to obtain the current reading from the Pentium CPU hardware tick counter [9]. On our 200 MHz target machine, the counter provides a resolution of 5 nanoseconds. To address the perturbation introduced by the instrumentation, we measured the latency overhead of the instrumentation code, and subtracted the overheads from our measurements.

We measured throughput by dividing the byte size of the data block by the stop latency.

We were also interested in the CPU utilization within the protocol stack. CPU utilization is defined as the percentage of time that the CPU was allocated to running the functions in the stack. To obtain these measurements, we employed VTune 3.0, a Windows systems performance measurement tool provided by Intel, Inc. [10].

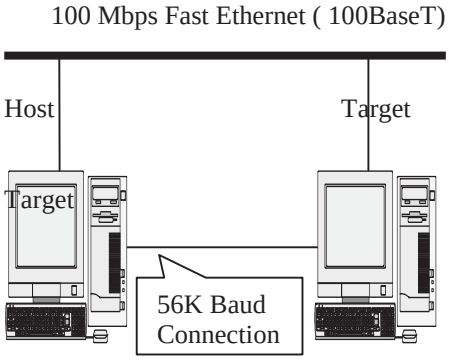



Figure 1: The experimental system

## 5 Results and Analyses

We describe below the outcomes obtained. More detailed descriptions can be found in [11].

### 5.1 Stack Latency

Figure 2 presents the measurements of the latencies as data is transmitted through the stack, obtained while varying the size of the payload data. The measurements are based on timestamps gathered at the top and bottom of the stack for individual packets as they passed through the stack on the sender's side. In Figure 2 , the lower curve represents the start latency measurements, which remain constant regardless of the buffer size, at approximately 100 *us*. The upper curve illustrates the stop latency, which exhibits the staircase pattern to be expected, due to the effect of the path MTU(Maximum Transfer Unit) on the TCP protocol. The trend-

line analysis of the measurements indicate a straight-line relation described as follows:

*stop latency*=192.57*x* + 246.98

where *x* is the size of the payload data in units of 1,000 bytes. (Note: In this paper the equations presented are based on first-order trend-line analyses. The equations are presented as a characterization of the
performance behavior and are not meant to be applied literally.)

Figure 3 shows another look of the measurements with the data size varying over a wider range.

Figure 4 shows a similar view of the stack latency when UDP is used instead of TCP. Here the start latency shows a linear trend which can be formulated as:

*start latency*=12.877*x* + 84.154

The stop latency is also linear, with a deeper slope, and can be formulated as:

*stop latency*=128.54*x* + 49.114

The larger start latency with UDP can be explained by the fragmentation performed at the IP layer.

Figure 5 juxtaposes the latency measurements under TCP and UDP respectively. Note the increasingly and slightly larger start latency with UDP and the increasing stop latency with TCP.

## 5.2 Latency Breakdown

We applied our instrumentation technique to measure the processing times in various parts within the stack, in order to obtain a breakdown of the processing overhead. Following in the path of the study described in [4], we obtained the latency measurements for three categories of operations within the stack: data checksum computation, buffer management, and data movement.

### 5.2.1 Checksum Computation

Figure 6 and Figure 7 present our measurements of the cumulative overhead for computing checksum for TCP and UDP, respectively. In both figures, the total checksum latency refers to the checksum overhead for all packets - regardless of whether the packet carries data or for protocol handshaking only, while the payload checksum latency refers to the overhead for payload data only.

From our findings, the TCP checksum overhead is no more than 12% that of the total latency. This contrasts with the nearly 50% reported with a TCP/IP running on a DEC Ultrix platform [4]. With UDP, the percentage is roughly 15%. However, the trend line observed is linear, consistent with that presented in [4].

### 5.2.2 Buffer Management

Figure 8 and Figure 9 present our measurements of the processing time overhead for allocation and deallocation of NDIS (Network Driver Interface Specification) buffers within the stack.

Under TCP, our findings reveal the followings:

- When compared to the stack latency, buffer management operations overhead is relatively small. The percentage peaks at 9.34% when the payload size approaches the adjusted path MTU of Ethernet (1,420 bytes), and stays at around 3% thereafter. It averages 3.78% of the total stop latency.

- We can expect a 4-us increase to the total buffer management overhead with every additional 1000-byte of payload data.

- Buffer allocation operations consume more processing time than the buffer deallocation operations. The former represents over 70% of the total buffer management overhead.

Under UDP, the buffer management overhead amounts to roughly 12% of the total latency.

### 5.2.3 Data Movement Operation

Data movement operations are those operations that perform the copying of packet data from one buffer to another, as needed for fragmentation in the IP layer. Our instrumentation results are described in Figure 10. The data movement overhead amounts to about 18.5% of the total UDP stop latency. No measurements were made for TCP, as no

fragmentation was observed in our experiments under TCP.

### 5.3 Throughput

We calculated the payload data throughput using the formula $T = P/L$, where $P$ is the byte size of the payload data and $L$ is the stack latency (the stop latency). The outcomes for TCP and UDP are presented Figure 11 and Figure 12 respectively.

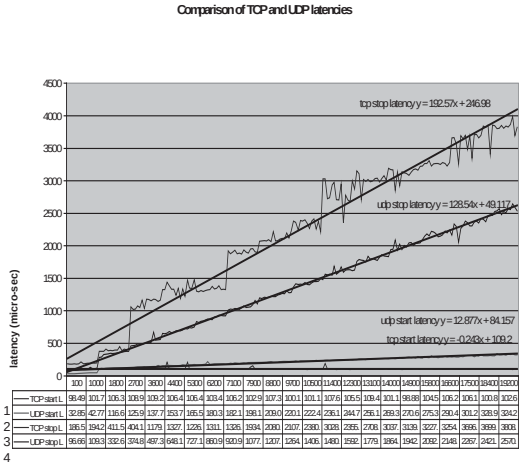
### 5.4 CPU Utilization

Using *VTune*, we obtained figures for CPU utilization for various portions of the stack.

Figure 13 shows a breakdown of the CPU utilization by functions within the *tcpip.sys* module of the stack. Our findings indicate that a total of 12.5% of CPU time was spent on the stack for TCP sending. Of that, 34% was spent on the checksum operations, and 21% on *TCPSend()*, a function which handles the TCP-protocol specific functionalities for data sending. Another 5% was consumed by the *TCPSendComplete()* function, which is called at the IP layer to complete a TCP session. The IP layer sending functions, *TunnelTransmitPacket()* and *TunnelTransmitComplete()*, took up 3.61% and 1.95% respectively.

In UDP (see Figure 14), the CPU utilization is higher, amounting to 25% on our test bed. This is consistent with the observation that UDP, a connectionless protocol, requires less protocol interaction between the two hosts and hence less wait time can be expected during the data transmission. Within the *tcpip.sys* module, the stack spends a large portion (61%) of its CPU utilization on the data-moving function *CopyFromBufferChain()*. Of the remaining CPU utilization consumed by the stack, 15% was spent on the checksum calculation function *tcpxsum()*. The fragmentation function *IPv6SendFragments()* took up another 7%. The IP layer send routine *IPv6Send()* consumed another 2.35%. The link layer routines *TunnelTransmit()* and *TunnelTransmitPacket()* made up 5.26%.

Figure 15 and Figure 16 present a breakdown of the CPU utilization by functionalities.




| | 100 | 1000 | 1800 | 2700 | 3600 | 4400 | 5300 | 6200 | 7100 | 7900 | 8800 | 9700 | 10500 | 11400 | 12300 | 13100 | 14000 | 14800 | 15800 | 16600 | 17500 | 18400 | 19300 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TCP start L | 98.49 | 101.7 | 106.3 | 108.9 | 109.2 | 106.4 | 106.4 | 103.4 | 105.2 | 102.9 | 107.3 | 100.1 | 101.1 | 107.6 | 105.5 | 109.4 | 101.1 | 98.68 | 104.5 | 105.2 | 106.1 | 100.8 | 102.6 |
| UDP start L | 32.85 | 42.77 | 116.6 | 125.9 | 137.7 | 153.7 | 165.5 | 180.3 | 182.1 | 198.1 | 209.0 | 220.1 | 222.4 | 236.1 | 244.7 | 256.1 | 269.3 | 270.6 | 275.3 | 290.4 | 301.2 | 318.9 | 324.2 |
| TCP stop L | 186.5 | 194.2 | 411.5 | 404.1 | 1179 | 1327 | 1226 | 1331 | 1336 | 1934 | 2090 | 2107 | 2380 | 3028 | 2955 | 2706 | 3097 | 3139 | 3227 | 3254 | 3696 | 3899 | 3808 |
| UDP stop L | 96.66 | 109.3 | 332.6 | 374.8 | 497.3 | 648.1 | 727.1 | 860.9 | 920.9 | 1077 | 1227 | 1264 | 1405 | 1480 | 1592 | 1779 | 1864 | 1942 | 2092 | 2148 | 2357 | 2421 | 2570 |

Figure 5: Stack latency, TCP and UDP

## 6 Idiosyncrasies of Our Results

This study was performed on a prototype of the IP Version 6 protocol stack, running on our specific processor architecture as described. The study was conducted on an isolated test bed where the test data transmission was the only application running on the personal computers. Our data gathering technique, source-code instrumentation, differed from those used in some other studies: instruction count in [1] and packet traces [4]. Our intention was to compile a profile of the performance of the stack on the NT platform, with the assumption that the outcome reported here provide a reasonable approximation of the behavior of the Microsoft TCP/IP stack.

For this paper, our measurements were gathered only on the sender host. We have plans to extend the study to cover the receiver host.

(Due to the limit on the number of pages, we are not able to include figures 2,3,4 and 6 through 16 referred to in the paper. They will be provided at the presentation, or by request via e-mail.)

## 7 Conclusion

In this paper, we presented our measurements of processing overheads (latencies), throughput, and CPU utilization within the MSR IPv6 protocol stack, as observed on the sending host. Similar to published studies conducted on Unix platforms, our work was undertaken in the hope of shedding some light on

where and how system resources are consumed and thereby providing insights on how best to maximize network throughput on NT platforms.

## 8 Acknowledgement

This work was made possible by the support of the 3Com Corporation. We would like to acknowledge in particular the guidance and support of Bill Huber of 3Com, Santa Clara. Thanks are also due to Cal Poly graduate students Jim Fisher for his contribution to the experiments and to the preparation of this manuscript.

## References

[1] David D. Clark, Van Jacobson, John Romkey, and Howard Salwen. An Analysis of TCP Processing Overhead. *IEEE Communications Magazine*, 27(6):23-29, June 1989.

[2] C. Partridge, J. Hughes, and J. Stone. Performance of checksums and CRC's over real data. In *Proceedings of the ACM SIGCOMMM Conference*, pages 68-76, July 1995.

[3] Jonathan Kay and Joseph Pasquale. Measurement analysis, and improvement of UDP/IP throughput for the DECstation 5000. In *Proceeding s of the Winter 1993 USENIX Conference*, pages 249-258, July 1993.

[4] Jonathan Kay and Joseph Pasquale. Profiling and Reducing Processing Overheads in TCP/IP. *IEEE/ACM Transactions on Networking*, 4(6):817-828, December 1996.

[5] Yasuhiro Endo and Margo I. Selzer. Measuring Windows NT – Possibilities and Limitations. In *Proceedings of the first USENIX Windows NT Symposium*, pages 61-66, August 1997.

[6] Richard P. Draves, Allison Mankin, and Brian D. Zill. Implementing Ipv6 for Windows NT. In *Proceedings of the first USENIX Windows NT Symposium*, pages 137-147, July 1998.

[7] Art Baker. *The Windows NT Device Driver Book*. Prentice Hall, 1997.

[8] V. Paxson, G. Almes, J. Mahdavai, and M. Mathis. Framework for IP Performance Metrics. Rfc2330, Network Working Group, the Internet Society, 1998.

[9] Terje Mathisen. Pentium Secrets. BYTE Magazine, *http://www.byte.com/art/9407/sec12/art2.htm*, July 1994.

[10] Intel Inc. Vtune Performance Analyzer. Webpage, *http://www.intel.com.tw/vtune/analyzer/index.htm*, 1999.

[11] Peter Xie. Network Protocol Performance Evaluation of Ipv6 for Windows NT. Master's thesis, Computer Science Department, California Polytechnic State University, San Luis Obispo, California, June 1999.