How to Use NTttcp to Test Network Performance

March 20, 2008

Abstract

NTttcp is a multithreaded, asynchronous application that sends and receives data between two or more endpoints and reports the network performance for the duration of the transfer. This paper is a summary of how to use NTttcp.

NTttcp is essentially a Winsock-based port of the [ttcp](http://en.wikipedia.org/wiki/Ttcp) tool that measures networking performance in terms of bytes transferred per second and CPU cycles per byte. Because it can be difficult to diagnose a system’s overall performance without dividing the system into smaller subsystems, NTttcp allows users to narrow the focus of their testing and investigation to just the networking subsystem.

This information applies for the following operating systems:  
 Windows Server® 2008  
 Windows Vista®  
 Windows Server 2003  
 Windows XP

References and resources discussed here are listed at the end of this paper.

For the latest information, see:   
 <http://www.microsoft.com/whdc/>

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Win32, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Document History

|  |  |
| --- | --- |
| Date | Change |
| March 20, 2008 | First publication |

Contents

[Introduction 4](#_Toc193618698)

[How to Use NTttcp 4](#_Toc193618699)

[NTttcp Examples 5](#_Toc193618700)

[Single-Threaded Operation 5](#_Toc193618701)

[Multithreaded Operation 6](#_Toc193618702)

[NTttcp Best Practices 7](#_Toc193618703)

[NTttcp Reference 8](#_Toc193618704)

[-l 8](#_Toc193618705)

[-n 9](#_Toc193618706)

[-p 9](#_Toc193618707)

[-a 9](#_Toc193618708)

[-x 9](#_Toc193618709)

[-rb 9](#_Toc193618710)

[-sb 10](#_Toc193618711)

[-i 10](#_Toc193618712)

[-f 10](#_Toc193618713)

[-u 11](#_Toc193618714)

[-w 11](#_Toc193618715)

[-d 11](#_Toc193618716)

[-t 11](#_Toc193618717)

[-v 11](#_Toc193618718)

[-6 12](#_Toc193618719)

[-fr 12](#_Toc193618720)

[-mb 12](#_Toc193618721)

[-m 12](#_Toc193618722)

[Resources 13](#_Toc193618723)

# Introduction

NTttcp is a multithreaded, asynchronous application that sends and receives data between two or more endpoints and reports the network performance for the duration of the transfer. It is essentially a Winsock-based port of the [ttcp](http://en.wikipedia.org/wiki/Ttcp) tool that measures networking performance in terms of bytes transferred per second and CPU cycles per byte. Because it can be difficult to diagnose a system’s overall performance without dividing the system into smaller subsystems, NTttcp allows users to narrow the focus of their testing and investigation to just the networking subsystem.

NTttcp measures a system’s networking performance for both Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) traffic. The application can be configured in many ways, including:

* Setting software affinity for threads to a specified processor index.
* Specifying asynchronous or synchronous data transfers.
* Specifying data verification at the application level for a predetermined pattern in the application buffers.
* Sending and receiving traffic from multiple Internet Protocol (IP) addresses with single command.
* Supporting IPv6 performance testing.
* Supporting UDP performance testing.
* Supporting time-driven testing.

This paper provides a summary of how to use NTttcp to test network performance. For more discussion of network performance testing in general, and ttcp and NTttcp in particular, see “Resources” at the end of this paper.

# How to Use NTttcp

The application that is included in the download package is named Ntttcp.exe. To use NTttcp.exe as a receiver or sender, simply rename a copy of the executable, as follows:

* To use NTttcp as a receiver, change the name to NTttcp**r**.exe.
* To use NTttcp as a sender, change the name to NTttcp**s**.exe.

If you use NTttcp without changing the name, it functions as a receiver by default.

To use NTttcp to measure networking performance:

1. Run NTttcpr.exe on the receiving computers first. It should open a listening socket with a default port number of 5001.

2. Run NTttcps.exe on the sending computer.

You can use multiple senders and receivers, and they can run on separate computers. For a good starting point for a one-to-one test on a low-latency configuration, see “NTttcp Examples” later in this paper.

The following are some usage guidelines.

#### Firewall Settings

Make sure that the NTttcp network traffic can pass the firewall on your test computers by specifying a firewall exception for the program name. If you don’t do this, the program will fail with a connect failure.

#### Port Numbers

Each specified thread has an assigned port, which means that as the thread number is incremented, so is the associated port number. For example, if the sender uses 6 threads to transfer data to two physical receivers with 3 threads each, the first client receives data on ports 5001 through 5003. The second receiver should start with a port offset of 5004 and receive data on ports 5004 through 5006.

#### Data Transfer Mode

By default, NTttcp uses synchronous data transfer because that is how most applications are written. However, you usually obtain the best network performance by transferring data asynchronously.

Data transfer is controlled by the **-a** option, as follows:

* For asynchronous data transfer, you must explicitly set the **‑a** option for both the sender and receiver.

The value you assign to **‑a** specifies the number of posted overlapped buffers to be used. If you set the **‑a** option but do not assign a value, the application uses two posted overlapped buffers per thread for sending and receiving data.

* If you do not set the **‑a** option, then the data transfer is synchronous.

**Tip:** The default value of two posted overlapped buffers should be sufficient for most asynchronous data transfer scenarios. We recommend that you increase the number of posted overlapped buffers for the receiver to ensure that the application has enough posted buffers for incoming data. This eliminates the possibility of suboptimal performance that can occur when the transport layer must buffer on behalf of the application.

# NTttcp Examples

This section contains two examples of how to run NTttcp. For details on the various options, see “NTttcp Reference” later in this paper.

## Single-Threaded Operation

This example shows examples of sender and receiver command lines for a typical single-threaded operation with one sender and one receiver.

#### Sender Syntax for Single-Threaded Operation

The following command line:

NTttcps -m 1,0,10.1.2.3 -a 2

runs NTttcps with the following explicit settings:

* “-m 1,0,10.1.2.3” specifies a single thread bound to CPU 0 that connects to the computer with IP address 10.1.2.3.
* “-a 2” specifies asynchronous data transfer that posts two send overlapped buffers.

This example uses the following default settings:

* A buffer size of 64 KB.
* The number of buffers to send of 20,000.
* A sender that starts sending on port 5001.

#### Receiver Syntax for Single-Threaded Operation

The following command line:

NTttcpr -m 1,0,10.1.2.3 -a 6 -fr

runs NTttcpr with the following explicit settings:

* “-m 1,0,10.1.2.3” specifies a single thread bound to CPU 0 that is bound on the receiving computer to IP address 10.1.2.3.
* “-a 6” specifies asynchronous data transfer that posts six receive overlapped buffers.
* “-fr” directs NTttcp to always post full-length (64‑K) receive buffers.

This example uses the following default settings:

* A buffer size of 64 KB.
* The expected number of buffers to be received of 20,000.
* A receiver that starts listening on port 5001.

## Multithreaded Operation

This example shows examples of sender and receiver command lines for a typical multithreaded operation with one sender and two receivers.

#### Sender Syntax for Multithreaded Operation

The following command line:

NTttcps -m 2,0,10.1.2.3 2,1,11.4.5.6 -a 2

runs NTttcps with the following explicit settings:

* “-m 2,0,10.1.2.3 2,1,11.4.5.6” specifies two threads per IP address.

The first set of threads is bound to CPU 0 and the second set of threads is bound to CPU 1. The threads connect to the receiving computer with IP addresses of 10.1.2.3 and 11.4.5.6, respectively.

In this example, each IP address can correspond to a separate physical computer.

* “-a 2” specifies asynchronous data transfer that posts two send [overlapped](http://support.microsoft.com/kb/q181611/) buffers for each thread.

This example uses the following default settings:

* A buffer size of 64 KB.
* The number of buffers to send of 20,000.
* A sender that starts sending on ports 5001 through 5002 to IP 10.1.2.3 and starts sending on ports 5003 through 5004 to IP 11.4.5.6.

#### Receiver Syntax for Multithreaded Operation with Two Receivers

The following command line on Receiver 1:

NTttcpr -m 2,0,10.1.2.3 -a 6 -fr

runs NTttcpr with the following explicit settings:

* -m 2,0,10.1.2.3” specifies two threads to receive data, both of which are bound to CPU 0.

The threads are bound on the local computer to IP address 10.1.2.3.

* “-a 6” specifies asynchronous data transfer that posts six overlapped receive buffers.
* “-fr” directs NTttcp to always post full-length (64‑K) receive buffers.

This example uses the following default settings:

* A buffer size of 64 KB.
* The expected number of buffers to be received of 20,000.
* A receiver that starts listening on port 5001.

The following command line:

NTttcpr -m 2,0,10.4.5.6 -a 6 -fr *-p 5003*

runs NTttcpr for Receiver 2 with the following explicit settings:

* “-m 2,0,10.4.5.6” specifies two threads to receive data, both of which are bound to CPU 0.

The threads are bound on the local computer to IP address IP 10.4.5.6.

* “-a 6” specifies asynchronous data transfer that posts six overlapped receive buffers.
* “-fr” directs NTttcp to always post full-length (64‑K) receive buffers.
* “-p 5003” directs the receiver to start listening on port 5003.

This example uses the following default settings:

* A buffer size of 64 KB.
* The expected number of buffers to be received of 20,000.

# NTttcp Best Practices

This section contains some best practices for using NTttcp.

#### Receive Buffers

Use the **-a** option to post sufficient receive buffers. This alleviates receiver-side TCP copying on the application’s behalf and avoids sender retransmits, which degrade networking performance. However, posting a large number of receive buffers is unnecessary because the first ones posted typically return quickly for reuse. Posting too many receive buffers can hurt performance because the scheduler must track a large number of outstanding transfers.

**Tip:** The following are good practices for the appropriate number of receive buffers:

* For 1‑gigabit links, use 6 receive outstanding buffers.
* For 10‑gigabit links, use 16 receive outstanding buffers.

#### Multiple threads

If you use multiple threads on a multiprocessor machine, distribute the threads across multiple processors by using the **-m** option’s second delimiter value. A single thread is usually sufficient to achieve line rate on a 1‑gigabit link, but multiple threads might be required to achieve line rate on a 10‑gigabit link.

#### The Full-Receive Option

To avoid tracking short receives, use the **‑fr** (full receive) option on the receiver. If, for example, the initially posted receive buffer has a length of 64 KB and you received only 60 KB, setting **-fr** forces the application to post another 64 KB to retrieve the remaining 4 KB. Most applications follow this model and do not post short buffers for small blocks of remaining data.

#### TCP Receive Window Size

For setups with high latency, use the **-rb** option to specify an appropriate TCP receive window size on the receiver, which describes the maximum allowed amount of inflight data. The value you assign to **-rb** specifies the latency in seconds multiplied by the bandwidth in bytes.

**Note:** Windows Server® 2008 and Windows Vista® dynamically tune the TCP window size, so you usually should not set the **-rb** option on those systems. NTttcp supports the **-rb** option for experimentation and for use with Windows® versions earlier than Windows Server 2008 and Windows Vista.

#### Using Continuous Loops

When running the NTttcp in a continuous loop to test different buffer lengths, we strongly recommend setting the **‑t** option. This option provides a warm-up and cool-down period between two instances of the test, which ensures that you get measurements at the stable point of operation. For consistency, the **-t** option also allows you to specify a constant time duration for all I/O sizes. If you do not use **‑t**, you might experience system socket errors caused by port reuse.

# NTttcp Reference

This section contains a detailed reference for the NTttcp options.

### -l

Description

Specifies the length of each buffer that NTttcp uses to transfer data. Each buffer contains random characters unless you also set the **‑d** option. The data transfer is from sender memory to receiver memory because the data is not read from a source and the data is discarded at the remote end.

Default Value

The default value is 64 KB. If you do not set **-l**, NTttcp uses the default value.

### -n

Description

Specifies the number of buffers that NTttcp transfers before it stops and reports the results. The length of each buffer is specified by the **‑l** option.

Default Value

The default value is 20,000. If you do not set **-n**, NTttcp uses the default value.

### -p

Description

Specifies the port number that the first thread uses to start transferring data. The port number is incremented for each additional thread.

Default Value

The default value 5001. If you do not set **-p**, NTttcp uses the default value.

### -a

Description

Enables asynchronous data transfer. The value assigned to **-a** specifies the number of outstanding I/O buffers to be used for sending or receiving data. If the sender experiences a large number of retransmits, increasing the value that is assigned to **-a** on the receiver might help to mitigate the problem.

Default Value

The default value is 2 overlapped I/O buffers.

If you do not assign a value to **-a**, NTttcp uses the default value.

If you do not set **-a** at all, the application transfers data synchronously.

### -x

Description

Directs the application to use the Win32® **TransmitFile** function. This option simulates scenarios like File Transfer Protocol (FTP), where large buffers are transferred.

Default Value

The default value is 1 buffer for each WSA call.

If you do not assign a value to **-x**, NTttcp uses the default value.

If you do not set **-x** at all, NTttcp uses the default **WriteFile** function instead of **TransmitFile**.

### -rb

Description

Specifies the total per-socket buffer space reserved for receives by setting the SO\_RCVBUF socket option value. This value, in turn, controls the receiver’s TCP window size. This option is used only by receivers and is set on a per-connection basis, as follows:

Windows versions earlier than Windows Vista set the TCP window statically. On these systems, the **-rb** option provides a way to adjust the window size depending on factors such as round trip time (RTT) and bandwidth. For high-latency links, it can be useful to increase this value above the default value of 64 KB.

In Windows Vista and later, the TCP window size is dynamically adjusted, so the **-rb** option should be used only if you understand the implications of statically setting TCP window size.

Default Value

In Windows Vista, the value that this option controls is set dynamically by default. In earlier versions of Windows, the default value is 64 KB. If you do not set **-rb**, NTttcp uses the default value.

### -sb

Description

Specifies the total per-socket buffer space reserved for sends by setting the SO\_SNDBUF socket option value. This option has a negligible impact on receivers. Senders set **-sb** on a per-connection basis, as follows:

In synchronous mode, if this option is not set, SO\_SNDBUF is set to the current WinSock default value of 8 KB. If you set **-rb** to a nonzero value, the send buffers complete faster, which speeds up the overall transfer rate. NTttcp posts a single buffer, which is similar to specifying **‑a 1**.

In asynchronous mode, if this option is not set, SO\_SNDBUF is set to zero. The buffers complete asynchronously and—with enough posted buffers—the application always has available buffers.

Default Value

NTttcp uses following default values for all Windows versions:

8 KB for synchronous transfer mode.

0 KB for asynchronous transfer mode.

If you do not set the **-rb** option or if you set the option but do not specify a value, NTttcp uses the default value.

### -i

Description

Used, for example, to send and receive UDP traffic indefinitely, while monitoring it through the UDP performance counters. It works only with UDP mode and cannot currently be set for TCP traffic.

Default Value

If you do not specify **-i** with UDP, NTttcp aggregates the results and reports them at the end of the test run.

### -f

Description

Redirects the results to a text file.

Default Value

If you set this option but do not provide a file name, the output is sent to output.txt in the current directory. If you do not set the option, NTttcp displays the output in the command window.

### -u

Description

Enable UDP mode.

Default Value

If you do not specify this option, NTttcp transfers TCP data.

### -w

Description

Directs NTttcp to use the Win32 **WSASend** and **WSARecv** functions to transfer data.

Default Value

If you do not specify this option, NTttcp uses the Win32 **WriteFile** and **ReadFile** functions to transfer data.

### -d

Description

Enables application-level data verification on the receiver computer. NTttcp expects a defined set of characters in its buffers and fails if the characters have been altered. The set of characters that is used for this option is defined by NTttcp.

**Important:** To enable this option, you must set it on *both* sender and receiver computers.

Default Value

If you do not specify this option, NTttcp does not verify data.

### -t

Description

Specifies the length of the test, in seconds. If you set this option, NTttcp automatically adds a warm-up and cool-down period to the beginning and end of the test, respectively. The warm-up and cool-down periods are each one-half the length of the specified duration. Adding warm-up and cool-down periods helps ensure that, when doing multiple runs, you measure performance when the machine is in a stable state.

Default Value

If you set **-t**, you must specify a value. If you do not set **-t**, NTttcp transfers the number of buffers specified by the **‑n** option and reports the results. If you specify a value for **-t**, this value takes precedence over **-n**.

### -v

Description

Enables verbose mode.

Default Value

By default, NTttcp does not display verbose information.

### -6

Description

Enables the IPv6 data transfer mode.

Default Value

By default, NTttcp transfers data in IPv4 mode.

### -fr

Description

Specifies full receive buffer mode. When **-fr** is set, NTttcp always posts buffers of the length that the **‑l** option specifies.

Default Value

If you do not set **-fr**, NTttcp posts a buffer with the length of the outstanding data. For example, if a 64‑KB receive buffer was posted and only 60 KB of data was filled, then the next receive buffer posted will be 4 KB in length. If you set **-fr**, the next receive buffer posted will be 64 KB.

### -mb

Description

Specifies the multiple receive buffer mode. A normal application uses distinct memory locations to store the content of separate buffers. Setting **-mb** simulates this scenario and allocates memory separately for each buffer specified by the **‑a** option. In synchronous mode, NTttcp uses only one memory location and only one buffer is posted at a time.

Default Value

By default, all receive buffers point to the same memory location. Overwriting memory is permissible for this test because the transferred data has no consumers. The default value measures the maximum attainable networking performance by stressing the stack and the network driver.

### -m

Description

Specifies a mapping for each IP address that is used. The mapping describes the number of threads, the processor ID, and the IP address. You can use multiple mappings on the sender or the receiver to transfer data between multiple computers, each with their own set of parameters. To specify a mapping, the number of threads, CPU, and IP address follow **-m**, separated by commas. A second mapping follows the first, separated by a space, and so on.

For example, the first mapping in the multithreaded sender example, discussed earlier, uses 2 threads bound to CPU 0 and sends to IP address 10.1.2.3. The second mapping uses 2 threads bound to CPU 1 and sends to IP address 11.4.5.6.

Default Value

You must set this option and specify at least one mapping.

# Resources

The following links provide additional information on network performance testing.

Testing Network Paths for Common Types of Traffic

<http://technet.microsoft.com/en-us/library/bb877965.aspx>

High Performance Network Adapters and Drivers

<http://www.microsoft.com/whdc/device/network/NetAdapters-Drvs.mspx>

Performance Tuning Guidelines for Windows Server 2008

<http://www.microsoft.com/whdc/system/sysperf/Perf_tun_srv.mspx>