Using Qtcp to Measure Network Service Quality

Introduction

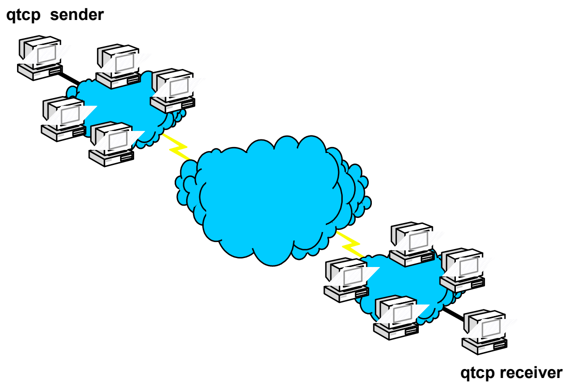
Qtcp is a tool, derived from Ttcp, that can be used to measure end-to-end network service quality. Qtcp sends a sequence of test packets through a network, then reports on the queuing delay experienced by each packet. Packets that do not arrive at the destination are recorded as dropped packets. Qtcp relies on a kernel-mode timestamping module that can be run only on Microsoft® Windows® 2000.

Features of Qtcp:

- Qtcp is able to report very precise delay variations, on the order of microseconds.
- By default, Qtcp invokes network Quality of Service (QoS) and is useful for the purpose of evaluating QoS mechanisms.
- Qtcp can simulate traffic flows for a range of user selectable packet sizes.
- Qtcp can simulate traffic flows shaped to a range of token bucket parameters.
- Qtcp can be used on an isolated, controlled network or a production network.
- Qtcp generates detailed result logs.
- Qtcp can collate a group of .sta files and produce a statistics summary across variables.

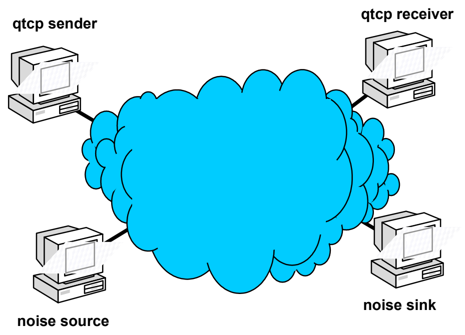
Network Under Test

Qtcp may be used on either a production network or on a controlled network that may be artificially congested. Devices in the network under test may provide quality of service through any number of QoS mechanisms or may not provide quality of service at all. The following diagram illustrates the usage of Qtcp in a production network:



In this diagram, a Qtcp sender and a Qtcp receiver are located at opposite ends of the network. The network under test is a production network. Other senders and receivers compete with the Qtcp session for network resources. In this scenario, the Qtcp user has no control over the current network load.

The following diagram illustrates the use of Qtcp on a controlled network:



In this diagram, the Qtcp sender and receiver are again located at the opposite ends of the network under test. In these scenarios, the network under test tends to be smaller and simpler to facilitate the evaluation of specific network elements in isolation. Note that, unlike the production network, there are only four hosts attached to this network. Two hosts are used to run the Qtcp test session. The other two hosts act as a noise source and noise sink. In these type of tests, the user controls the amount of noise generated across the network during any specific test run. Generally, noise sources are selected to mimic the distribution of packet sizes and the traffic level that would be encountered on a production network.

Theory of Operation

A Qtcp session is invoked on both a sending and a receiving host. After invocation, Qtcp sets up a TCP control channel between sender and receiver to properly negotiate the test. At the same time, it uses the GQoS API (part of Winsock2) to invoke QoS from local traffic control and from the network. The Qtcp sender will cause an RSVP PATH message to be sent towards the receiver and will wait until a response is received. The Qtcp receiver will wait for an RSVP PATH message from the sender and will respond by transmitting an RSVP RESV message.

Receipt of the RESV message at the sender triggers the measurement phase. At this time, the sender begins submitting buffers to the kernel for transmission. (Note that kernel traffic control must be installed). The kernel paces the transmission of traffic according to the token bucket parameters and service type selected by the user via the Qtcp command line. As packets are transmitted, the timestmp module stamps a sequence number and the local time (to a precision of 100ns) in each packet. If the timestmp module is not installed, Qtcp uses a user mode timestamp that is less accurate than the kernel mode module.

When packets are received at the receiver's traffic control, the timestmp module stamps in each packet the local time of the receiving host, then passes the packet up to the receiving Qtcp peer. The receiving Qtcp process maintains a list of all received packets, including the packet sequence number, the time sent and the time received.

The test terminates on the sending side when the transmitter has sent the required number of packets (default is 2048 packets, may be overriden via the Qtcp command line). Following transmission of the last packet, the sender sends a terminating sequence of ten 'termination' packets. The test terminates on the receiving side upon receipt of a termination packet or upon receipt of the required number of packets (default 2048, may be overriden via the receiver Qtcp command line). Note that, on particularly congested links, the receiver may never receive the required number of packets and the termination packets may be dropped. The TCP control channel should still get through its 'DONE' message, in which case the receiver will terminate normally. If both of these methods fail, the receiver may be terminated manually, by typing 'q' <return>.

Upon termination, the receiver Qtcp parses and processes the log of received packets. Three logs are generated; filename.sta, filename.raw and filename.log. The first of these contains summary statistics. It reports the total number of packets received and specifies the sequence number of each dropped packet. The second file (.raw) contains a detailed log showing normalized send time and receive time for each packet, the latency (difference between sent and received time), packet size and sequence number. Finally, the third file (.log) is a result of normalizing the results of the second file, to account for clock drift between the two hosts (see Appendix A).

Usage

The simplest invocation of Qtcp is as follows:

On sender: **qtcp -l64 -t 2.3.4.5**
On receiver: **qtcp -f"filename" -r**

The **-l** option on the sender specifies the buffer size to be used in the test. The **-t** option specifies that this is the Qtcp sender. This option must be followed by the IP address of the receiving host. On the receiver, the **-f** option is used to specify the prefix of the report files that will be generated. The **-r** option indicates that this is the Qtcp receiver.

Initially, the sender prints the message: *Initiated QoS connection. Waiting for receiver.* The receiver prints the message: *Waiting for QoS sender to initiate QoS connection*. At this time, the receiver and the sender are awaiting the required exchange of RSVP messages before beginning the data transfer. Once the data transfer begins, the sender will start sending 64 byte buffers (as specified in the **-l** option on the sender), to kernel traffic control, for transmission to the receiver at address **2.3.4.5**. By default, kernel traffic control will pace transmitted packets to a rate of 100 KBps (kilobytes per second).

Qtcp will print a series of dots to the console, both on the receiver and on the sender. Each dot corresponds to 100 packets sent or received. Note that the first dot is printed on the receiver prior to the actual receipt of the first 100 packets. The dots should be used as an indication that Qtcp is 'alive'.

Upon transmission of the specified number of packets (2048 by default), the sender terminates with a message regarding the transmission rate. Note that the transmission rate and other statistics printed by the sender are from its perspective alone. That is, the rate it prints is the rate at which it sends packets and does not necessarily correspond to the rate at which packets are received. Upon receipt of the required number of packets (or the termination packets) at the receiver, it terminates with the message: *Received 2048 buffers*. This is followed by statistics, which are not reliable. At this time, the receiving Qtcp will generate the files: **filename.sta**, **filename.raw** and **filename.log**.

Usage Variations

In this section, we discuss the usage variations and optional parameters that may be used. In the table below, the third column indicates whether each option is useful on the Qtcp sender (T), Qtcp receiver (R) or both (B). Note that options are case-sensitive.

Optional Parameters

Option	**Description**	**Sender (T) Receiver (R) Both (B)**	**Default and Range**
-B	This is the Token Bucket size. The token bucket size represents the largest burst that traffic control will transmit to the network. It should generally be set to be equal to the buffer size. See notes below.	T	Default is 64 bytes. Must be no smaller than buffer size and no smaller than MTU size.
-m	Minimum policed size advertised by RSVP signaling.	T	Default is buffer size.
-R	This is the Token Rate. The token rate is the average rate at which data will be transmitted in Kbytes per second. This parameter may be used to control the inter-packet gap on the sending host. See notes below.	T	Default is 100 Kbytes. Should be lower than the media rate.
-R##B	Token Rate as above expressed in bytes per second.	T	See above.
-S	This is the intserv service type that will be signaled to the network and that will be used for local traffic control. See notes below.	T	**GR** or **CL**.
-e	This option forces shaping to the token rate	T	Default is off (GR shapes, CL does not)
-W	This will suppress waiting for an RSVP reservation. See notes below.	B	No parameters.
-v	This option is used to set up an RSVP reservation only and then wait for the user to exit. No data is sent.	T	Data is sent by default.
-f	This option is used to specify the prefix name for the	R	By default no files

	logging and statistics files on the receiver.		will be generated.
-F	This option is used to convert a raw file to a log file.		See notes below.
-n	Number of buffers transmitted.	B	See notes below.
-c	Number of calibration buffers to be sent.	B	See notes below.
-k#	0: Do not calculate clock skew 1: Calculate clock skew based on a linear regression with goodness of fit measured by sum of squared error. 2: Calculate clock skew based on a bracketing and bisection root finding method with goodness of fit measured by sum of absolute error. 3: Same as 2, except also try to compensate for clock jumps (see appendix A).	R	Default is 2
-y	This option skips waiting for user confirmation to continue after the calibration phase has been completed.	T	Default is to wait for confirmation.
-p	Destination IP port number on receiving host. In order to run multiple simultaneous Qtcp tests between the same pair of hosts, it is necessary to use separate ports for each test. This option may be used to force a specific port to be used. Otherwise all streams will attempt to send to the default port on the receiver. This option must be used on both sender and receiver.	B	Default is port 5003.
-l	Length of user level buffers generated by Qtcp. See notes below on relationship of buffer size to token bucket size and MTU size.	T	Default is 1472. Must be greater than 48 bytes.
-d	By default, dropped packets omitted from the log. If this option is specified, they are represented by dummy entries in the log files showing a sent and received timestamp of zero and the maximum latency experienced by any packet in the test run. This option can be used to facilitate accommodate different types of log file post processing.	R	Default is no dummy entries.
-N	Causes raw file to be dumped after normalization.	R	Default is pre-normalization. See notes below.
-M	MaxSDUSize to be used in signaling messages	T	Default is buffer size.
-P	Suppresses console reporting of dropped packets.	R	Default is to report.
-u	Report user-mode timestamps in logfiles.	R	Default is kernel mode.
-i	Use more compressible data in the packets.	T	Default is less compressible.
-q	Log only every nth packet. For instance, -q2 will log every other packet (0,2,4,…).	R	Default is –q1 (log every packet)
-A	Used to tell Qtcp to collate all the .sta files in a directory and produce a statistics summary output. For instance, to collect the results in c:\results, you would enter qtcp –A"c:\results"	B	Not involved in default operation.

Notes Regarding Parameter Usage

Fragmentation Avoidance - Relationship of Token Bucket Size, Buffer Size and Packet Size

Under certain conditions, the protocol stack on the sending host would have to fragment user level buffers into multiple packets in order to transmit them over the network. Fragmentation is undesirable for a number of reasons, as described below:

1. Certain network QoS mechanisms are unable to handle fragmented packets.
2. The loss of a single packet results in loss of the entire corresponding buffer at the receiving host, even though all packets may have successfully traversed the network.
3. The user level Qtcp process on the receiving host recognizes buffers, not individual packets. As a result timestamps and sequence numbers would be recovered only from the first packet comprising each buffer. Information stamped by the sending timestmp module in the remaining packets of the buffer, would not be recoverable.

The size of user level buffers generated by the sending Qtcp is determined by the **-l** option (default is 1472 bytes since 1472(buf size)+20(IP Header)+8(UDP Header) = 1500(MTU size)). If the buffer is larger than the token bucket size (selected by the **-B** option) used by traffic control, or larger then the MTU size defined for the network interface, then the sending protocol stack will have to fragment each buffer into multiple packets. Since fragmentation is undesirable, this condition is to be avoided. Qtcp will prevent the user from defining a buffer size that is larger than the token bucket size. However, the user is responsible for determining the MTU size for the interface and assuring that the buffer size is no larger than the MTU size.

On the other hand, if the buffer size is smaller than the token bucket size, then multiple user level buffers may be transmitted by sending traffic control in a single burst. As a result, the transmit timestamps in these packets will be closely related. This tends to distort the results of the measurement. Optimal results are obtained when packets are sent at a steady rate with equal inter-packet gaps between successive packets.

In conclusion, for best results, it is recommended that the buffer size be equal to the token bucket size and that both are equal or smaller to the MTU size.

Token Rate
As discussed previously, optimal results are obtained when the sending host sends packets at a constant rate. If the token bucket size is chosen to be equal to the buffer size, then the token rate (determined by the **-R** option) will determine the constant packet rate. So, for example, for a token bucket and buffer size of 64 bytes and a token rate of 16, packets will be sent at the uniform rate of 250 packets per second.

Each packet can be considered to be taking a snapshot of the network conditions at the time it is sent. Thus, the token rate can be used to select a sampling interval for the network under test. Too low a sampling interval may cause transient network conditions to be missed. On the other hand, the token rate should be selected so that the packet rate is relatively low compared to the measuring processes in the sending and receiving hosts. Measurement error increases with higher packet rates.

The token rate selected should be lower than the media rate. When using Qtcp to evaluate the effects of QoS on telephony traffic (for example), we usually select token rates on the order of 3 - 10 Kbytes per second.

Service Type
The service type (selected by the **-S** option) can be used to select the intserv service type requested by the RSVP signaling messages and to control the mode of the sending host's traffic control. The two types of service are **GR** (guaranteed service) and **CL** (controlled load) service. See RFC 2210 for a description of these services. Guaranteed service is selected by default.

If there are RSVP/Intserv aware devices in the network under test, then the choice of service type will affect the handling of the test traffic by these devices (as described in RFC 2210). Regardless of the existence of RSVP/Intserv aware network devices, the choice of service will affect traffic control on the sending host by determining the mode in which the packet scheduler operates. Unless configured otherwise, the packet scheduler on the sending host will operate in *shape* mode for guaranteed service and in *borrow* mode for controlled load service. In shape mode, the packet scheduler will shape transmitted traffic to the token bucket parameters. In borrow mode, the packet scheduler will not shape traffic to the token bucket parameters. Instead, it will transmit traffic up to the media rate, demoting in priority, those packets that are transmitted in excess of the media rate.

Operating Qtcp in borrow mode tends to result in bursts of traffic and is not recommended for measurement purposes. However, it may be desirable to select controlled load service to compare the effect of various service types when there are RSVP/Intserv aware devices in the network under test. In this case, we recommend configuring the packet scheduler to operate in shape mode for controlled load flows. This can be accomplished by specifying the –e flag when running Qtcp.

No Wait Flag
By default, Qtcp will not begin data transmission until an RSVP reservation is in place. The **-W** option allows data transmission to proceed even when there is no reservation in place. This option can be invoked in order to enable testing when the network under test prevents a reservation from being installed. This could happen for the following reasons:

1. A firewall in the path between sender and receiver is configured to block RSVP messages.
2. An RSVP aware device in the network is rejecting the RSVP request due to lack of resources, policy or other reasons.

Note that by using the **-W** option, the synchronization inherent in RSVP is lost. This means that the sender will not wait for the receiver to be started. Thus, it is necessary to start the receiver *before* the sender.

Number of Buffers
The **-**n option can be used to select the number of buffers which the transmitter sends or which the receiver expects to receive during a test run.

Too small a number of buffers will result in the error message "Time interval too short for valid measurement." Too large a number of buffers on the receiver will result in the error message "Could not allocate X bytes for log buffer.", indicating that the receiver was unable to allocate sufficient memory to record timing data for the number of buffers it would have to receive.

Note that both sender and receiver use 2048 as the default number of buffers. If the **-n** option is used on the sender to restrict the number of buffers sent to less than the default, then no action is required on the receiver. However, if the sender is configured to transmit more than the default, then the corresponding option must also be selected on the receiver to prevent it from terminating after the default number of buffers have been received.

If you would like to send buffers for an approximate amount of time, you can use the 's' suffix to the n parameter. If, for example, you want to send buffers for 3600 seconds, you would specify –n3600s.

Calibration
The **-c** option can be selected to enforce a calibration phase. This option takes a number of calibration buffers as an argument. On the sender, the specified number of calibration buffers is sent in addition to the number of buffers specified by the **-n** parameter (or the default of 2048). On the receiver, the best-fit curve calculations used to normalize for clock skew (see Appendix A) are based on the calibration buffers received only (unless none are specified, in which case they are based on all buffers). (Other statistics are based on all buffers including both calibration buffers and none-calibration buffers).

When calibration buffers are specified on the sender, it sends the number of calibration buffers specified then pauses and prompts the user with the message "Calibration complete. Type 'c' to continue." When the network under test is isolated and under user control, this mechanism facilitates the evaluation of the network. In this case, the user should begin the test run on a quiescent network. Data obtained during the calibration phase of the run is used by the receiving Qtcp to improve the integrity of the latency reports generated and of clock skew normalizing (see Appendix A). Once the calibration phase has completed, the user should start any noise generating tools being used to congest the network under test. After noise generation has been started, the user should type 'c' to continue with the measurement phase of the test.

The following sample invocation may be used:

On sender: **qtcp -c1000 -n1000 -l64 -t 2.3.4.5**
On receiver: **qtcp -c1000 -f"filename" -r**

This will cause the sender to transmit 1000 calibration buffers, followed by 1000 non-calibration buffers. The receiver will use the first 1000 buffers received to normalize for clock skew.

Converting a raw file
The –F option will convert a given raw file into a log file, sending no network traffic of any kind. This can be used in case there are improvements to the normalizing algorithm, for instance, on an old raw file. If you used calibration packets in the run that created the raw file, you should specify their number when invoking this conversion by adding the –c## option; i.e., for 1000 calibration packets, -c1000.

Example:

To convert test.raw into a log file, use **qtcp –F"test"**
The output will be in **test.log**

Diffserv Codepoint, TOS/Precedence and 802.1p

Because Qtcp invokes QoS on the sending host, it will cause traffic control to mark transmitted packets for certain QoS service levels. Specifically, traffic control will mark the diffserv codepoint (DSCP, formerly known as TOS and precedence bits) in the IP header. It will also mark 802.1p tags in the MAC header of packets sent on an 802.1p capable network (such as 802.1p enabled Ethernet). The user should consider the effects of these packet markings when evaluating the results of Qtcp tests.

The user may control the markings applied by traffic control in order to study the effects of particular markings on network service quality. By default, packet markings are determined by the service level (guaranteed or controlled load) selected for test traffic. Default mappings are as follow:

Service Level	**DSCP**	**802.1p**
Guaranteed	IP precedence 5	5
Controlled Load	IP precedence 3	3
Best effort	IP precedence 0	0
Non-conforming traffic	IP precedence 0	1

The user may alter the markings by creating a marking table in the registry under the Psched/Parameter key for the appropriate interface.

Usage Hints and Troubleshooting

Waiting for RSVP Reservations

Before Qtcp begins sending test data, it waits for an RSVP reservation to be established between the sender and receiver. Reservations may take up to 30 seconds to be established and may not be established at all under certain conditions. If an RSVP reservation is failing to be established it may be because there is a network device in the path between sender and receiver, which is rejecting or blocking the reservation. Firewalls may do so by simply blocking all RSVP messages. RSVP enabled routers may do so if they are not provisioned to allow the requested reservation. Check for the presence of such devices. If it is not possible to complete the RSVP reservation, it may be necessary to run using the **-W** flag, which allows Qtcp to transmit test data without a reservation in place.

TCMON

TCMon enables the user to observe traffic control behavior. Install tcmon on the Qtcp sender, then select the interface over which Qtcp will be running and enable auto-refresh. As soon as Qtcp is invoked on the sender, tcmon should indicate two flows - one for the RSVP messages themselves (identifiable by the service type *Network Control*). The other flow is for the Qtcp data. This flow is identifiable initially by the service type Best *Effort*. However, upon completion of the RSVP reservation, the service type of the data flow should change to either *Guaranteed* or *Controlled Load*. Note that the rate indicated for the data flow will actually be higher than the token rate specified. This is because the QoS service provider prorates the requested data rate to account for network layer packet headers. As data is being sent, the *Bytes Sent* counter in tcmon should increase in value.

Invalid Log File Data

If the data in the .log file appears invalid, it may be as a result of a number of conditions. The receive and send timestamps should each be monotonically increasing. If either the receive timestamps or the send timestamps are all equal, it is likely that the timestmp module did not install correctly on the corresponding host. Remove and re-install the timestmp module.

If timestamps are not equal, but appear to vary widely, it may be the result of an error in the processing step that normalizes for clock skew. This can be confirmed by comparing the contents of the .log file to the contents of the .raw file. The .raw file is generated any normalizing processing is applied. The results in the .raw file may be used if clock skew is considered negligible.

If the .log file is completely empty, check the .sta file. It may indicate that all packets have been dropped.

Receiver Termination

If the network under test is extremely congested, the Qtcp receiver may not receive its termination sequence and may not receive sufficient packets to terminate automatically. It should still get the message to terminate over its control channel. If it does not, however, type ‘q’<return> at the receiver console some time *after* the sender has indicated that its transmission is complete. This will terminate the receiver session causing the log files to be generated based on whatever data was successfully received at that time.

Error Messages

Network transmission rejected - This error indicates that the sender attempted to send to a closed socket. This can occur when the RSVP session is torn down by an RSVP aware network device on the path from sender to receiver. In this case, the offending device should be corrected and the test should be re-run.

This message may also be received under normal conditions at the end of a test. In this case, it indicates that one of the first termination sequences caused the receiving Qtcp session to shut down the Qtcp receiving socket, sending an RSVP teardown message to the sender. This may cause the sending socket to close before the last termination sequence has been submitted to the network, resulting in the behaviour described. In this case, the logged data can be considered valid.

Appendix A

Upon receiver termination and before the Qtcp receiving application exits, it parses the list of sent and received times in its received packet log. Assume for the purpose of this example that receiving timestamps are always later than sender timestamps (logic is implemented to allow for the case in which the receiver's timestamps are actually earlier than the sender's timestamps). Qtcp looks for the lowest sent time stamp (should be the first) and for the lowest difference between sent time and received time across all pairs of timestamps (lowest latency). It records these two values as 'LowestSendTime' and 'LowestLatency'. These are used in the subsequent normalizing process.

Once Qtcp has completed parsing the list, it subtracts the LowestSendTime from all sent timestamps. This has the effect of normalizing the first packet's send timestamp to zero and each following packet's timestamp to the difference between the time the first packet was sent and the time the following packet was sent. Next Qtcp subtracts the LowestSendTime from all received timestamps. Finally, Qtcp subtracts LowestLatency, from each packet's received timestamp. This has the effect of normalizing all received timestamps such that they represent the latency *in excess of the minimum latency seen.* For example, consider the following set of timestamps:

TimeSent	**TimeReceived**	**Latency**
10	13	3
11	19	8
12	18	6

Upon parsing the records, Qtcp determines that LowestSendTime is 10 and LowestLatency is 3.

After the first normalizing step, the table looks as follows:

TimeSent	**TimeReceived**	**Latency**
0	13	3
1	19	8
2	18	6

During the next normalizing step, LowestSendTime is subtracted from all received time stamps, yielding the following results:

TimeSent	**TimeReceived**	**Latency**
0	3	3
1	9	8
2	8	6

In the final normalizing step, LowestLatency is subtracted from all received time stamps and the latency is updated to reflect the difference between the normalized send and receive time stamp pairs. This yields the following results:

TimeSent	**TimeReceived**	**Latency**
0	0	0
1	6	5
2	5	3

Note that the first packet shows a latency of zero. This does not mean that the transmission delay is zero. Rather, it means that this packet's delay represents the best case or fixed delay that occurs between sender and receiver. Normalized latencies that are greater than zero indicate the amount of delay *beyond* the fixed delay or beyond the minimum delay. As such, these latencies represent the variable delay component that results from queuing and congestion. This is considered more interesting than the fixed delay component from the perspective of Qtcp.

Validity of Results

No attempts are made to determine the actual latency between sender and receiver. Instead, Qtcp attempts to determine the variable delay component that is considered to be indicative of queuing delays and congestion effects. This approach will not always yield valid results. It is based on the assumption that one of the following two requirements are met:

1. The test includes a calibration phase - if the network under test is an isolated network, controlled exclusively by the tester, then it is recommended that the calibration option (**-c**) be used. During the

calibration phase, no noise should be generated on the network. This assures that queues in network devices will be empty and that the minimal latency logged by Qtcp will indeed be indicative of the fixed delay component of the network.

2. Sufficient packets are sent - if a large enough number of packets is sent during a test run, it is statistically very likely that at least one of the packets will end up in a very short (or zero length) queue and will not be subjected to congestion delay. The minimum delay used by Qtcp will be the delay experienced by this packet. This assumption fails on a heavily congested network in which queues never drop to zero length.

Normalizing for Clock Skew - Difference Between .raw and .log Files

PC clocks are based on a crystal oscillator timebase. Oscillators are subject to deviation from their nominal frequency, which is on the order of several parts per million. As a result, send time stamps and receiving timestamps are generated based on a slightly different speed clock. When measuring queuing delays on the order of milliseconds over a period of seconds or minutes, the skew between the sending oscillator's rate and the receiving oscillator's rate is negligible. However, when measuring queuing delays that are much lower (such as on a high speed LAN) or when measuring delays over a long period of time, the clock skew may become significant.

To compensate for the clock skew, Qtcp normalizes the output of the raw file into the log file. This normalizing step is based on the assumption that the clock skew is constant, while queuing delay is variable. In order to normalize for the effects of clock skew, Qtcp attempts to fit the latency reports to a constant slope line. There are several options for this clock skew normalization. The –k0 option turns off the normalization. –k1 fits to a straight line using the sum of squared error as the goodness of fit measure (this makes an implicit assumption that latency is normally distributed). The default, -k2, option uses absolute deviation as the goodness of fit measure (the assumption here is that latency is more like a double exponential about the mean).

The –k3 option is only to be used in special cases. It is designed to fix a known problem. On certain machines, those with a piix4 timer chip (Qtcp should warn you if you have a clock chip whose frequency matches that of the piix4), the clock sets itself forward backward a specific amount every so often. The –k3 option tells Qtcp to try to detect and compensate for this. It is not certain to detect the clock jumps and only the clock jumps, but it has succeeded in all tests so far.

As a consequence of all this, any constant variation in latency will be removed from the log file. In the rare case that the user is interested in measuring constant changes in latency over time, the user should work from the raw file (not the log file) and should account for the fact that part of the latency is attributable to clock skew. (Clock skew measured on current technology PCs between two PCs is on the order of 10 microseconds per second).

Appendix B - Known Bugs

Disclaimer - Qtcp has not been subjected to extensive testing and is provided 'as is', with no guarantees. Comments, suggestions and questions may be sent to *rkinput@microsoft.com.* Qtcp has been used over a limited range of parameter values and has been found to operate correctly over these values. The following parameter values have been used and as such, can be considered to have been partially tested:

Option	**Values Tested**
-B	Primarily 64 bytes. Occasional testing up to 1500 bytes.
-m	Never used.
-R	Primarily 3 to 8 Kbytes. Occasional use up to 100 Kbytes.
-S	Rarely used. Most often defaulted to guaranteed service.
-W	Used occasionally.
-f	Always used.

-n	Primarily 1000 - 2000. Occasionally up to 10000.
-c	Primarily 1000, occasionally less.
-p	Never used.
-l	Primarily 64 bytes. Occasional testing up to 1500 bytes.
-d	Rarely used.
-a	Rarely used. Occasionally increased from default of 3, to 20.

The following are known bugs:

1. None

Appendix C - Post Processing

The .log file and the .raw file data should be interpreted as follows:

Column 1: Send time stamp in units of 100 nsec.
Column 2: Receive time stamp in units of 100 nsec.
Column 3: Latency in units of 100 nsec.
Column 4: Packet size.
Column 5: Sequence number.

It is helpful to plot the data captured in the .log file in order to interpret the results of the test runs. You may use a Microsoft® Excel macro contained in the file Qtcpmacros.xls to read the log file and plot the results. Instructions are contained in the .xls file itself. If you do not have this macro file, simply use the *File Open* menu to load the data file. Specify that a colon will be used as the delimiter. Once Excel has loaded the data file, highlight the third column (latency) . Next, chart it using the *Line* chart. You may also select the *Data Analysis* tools to obtain statistics and histograms regarding the distributions of latencies. When applying the data analysis tools, be sure not to include log entries generated during the calibration phase of the test (if calibration was used), as this will skew the results.

Appendix D - Sample Results

Here we present sample results. Two test trials were run. Both trials were run across an isolated network consisting of two RSVP capable routers connected by a 128 Kbps serial line. Each router was also equipped with an ethernet interface. The Qtcp sender was connected to one of the router's ethernet interfaces, the Qtcp receiver was connected to the other.

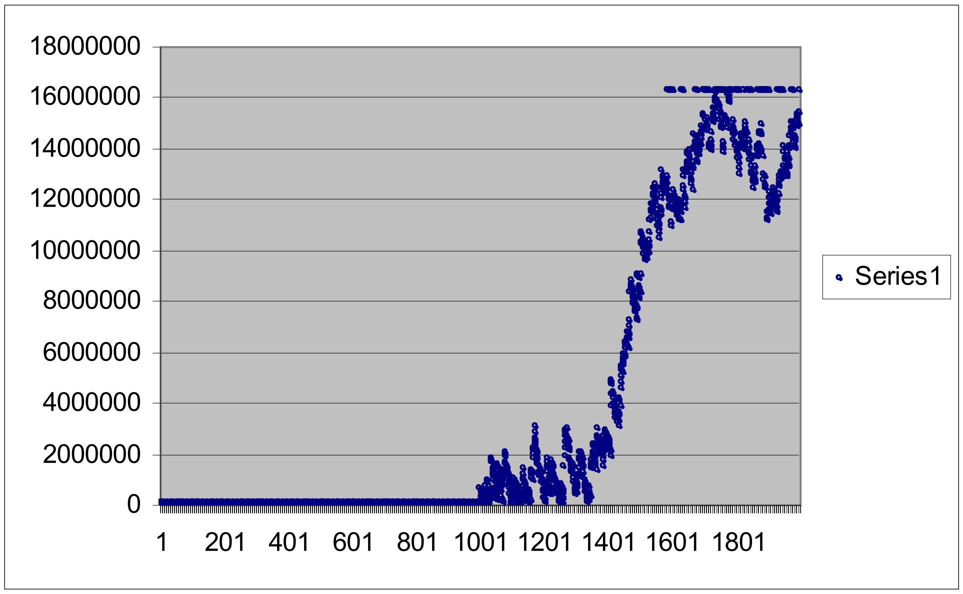
In addition, the sending port of a 'Smartbits' noise generator was connected to the same ethernet network as the Qtcp sender. The receiving port of the 'Smartbits' was connected to the same ethernet network as the Qtcp receiver. The 'Smartbits' was programmed to send 100 Kbps of noise traffic from sending port to receiving port such that the traffic generated by the 'Smartbits' would compete with the traffic generated by the Qtcp session for resources on the 128 Kbps link. The 'Smartbits' was programmed to generate a mix of packet sizes that simulates the typical load on a real corporate WAN link.

Qtcp was invoked in a manner intended to simulate a telephony traffic flow. The following parameters were used:

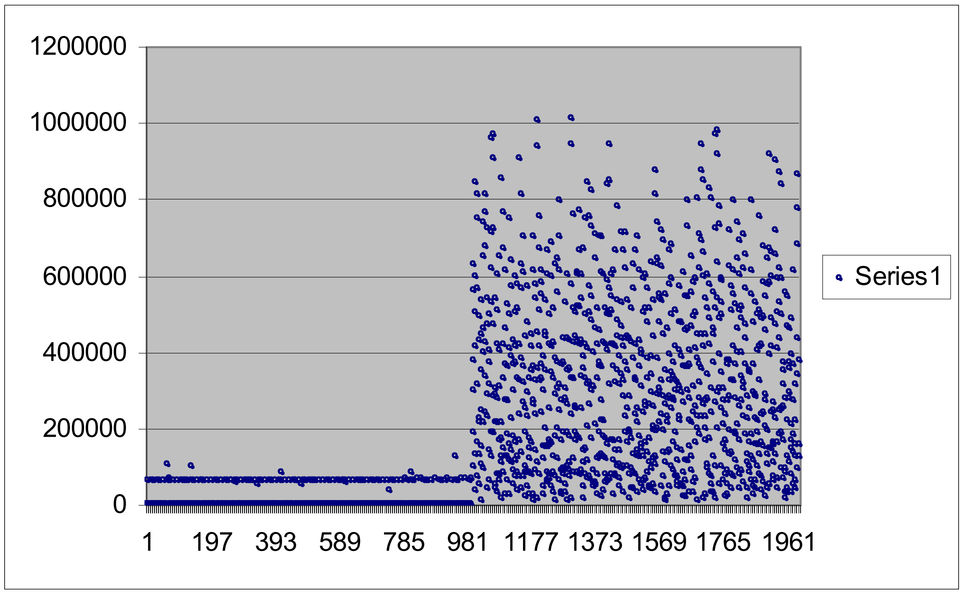
Sender: **qtcp -l64 -R3 -c1000 -n1000 -t 2.2.2.2**
Receiver: **qtcp -f"test" -c1000 -r**

This results in a test traffic flow of 64 byte packets sent at a rate of 24 Kbps.

The purpose of the test was to examine the utility of RSVP in protecting the signal flow on a WAN link driven to near saturation. To this end, the first trial was run without RSVP enabled on the routers. The second trial was run with RSVP enabled on the routers. In both cases, RSVP signaling was used between the sender and receiver. The results are illustrated below. Plots were generated using the technique described in Appendix C. The X-axis represents the packet sequence number. The Y-axis represents the normalized latency in units of 100 nsec.



This plot illustrates the latency experienced by the signal packets without RSVP protection. Note that the first 1000 packets (sent during the no-noise calibration phase) show negligible latency. The second 1000 packets however, sent while background noise was generated) show a steadily increasing latency, up to over 1.6 seconds. Also, note the set of points aligned horizontally near the top of the plot. These correspond to the entries generated for dropped packets (which are represented by the maximum latency experienced by any of the received packets).



This plot illustrates the latency experienced by the signal packets with RSVP protection. Again, the first 1000 calibration packets show very low latency (note that the Y-axis scale is different between the two plots). The second 1000 packets show a distribution of latencies. However, the maximum latency is limited to 100 msec. Furthermore, no packets have been dropped.

For questions or feedback concerning this tool, please contact rkinput@microsoft.com.