Network Working Group
Request for Comments: 998
Obsoletes: RFC 969

David D. Clark
Mark L. Lambert
Lixia Zhang
MIT

March 1987

NETBLT: A Bulk Data Transfer Protocol

1. Status

This document is a description of, and a specification for, the NETBLT protocol. It is a revision of the specification published in NIC RFC-969. The protocol has been revised after extensive research into NETBLT's performance over long-delay, high-bandwidth satellite channels. Most of the changes in the protocol specification have to do with the computation and use of data timers in a multiple buffering data transfer model.

This document is published for discussion and comment, and does not constitute a standard. The proposal may change and certain parts of the protocol have not yet been specified; implementation of this document is therefore not advised.

2. Introduction

NETBLT (NETwork BLock Transfer) is a transport level protocol intended for the rapid transfer of a large quantity of data between computers. It provides a transfer that is reliable and flow controlled, and is designed to provide maximum throughput over a wide variety of networks. Although NETBLT currently runs on top of the Internet Protocol (IP), it should be able to operate on top of any datagram protocol similar in function to IP.

NETBLT's motivation is to achieve higher throughput than other protocols might offer. The protocol achieves this goal by trying to minimize the effect of several network-related problems: network congestion, delays over satellite links, and packet loss.

Its transmission rate-control algorithms deal well with network congestion; its multiple-buffering capability allows high throughput over long-delay satellite channels, and its various timeout/retransmit algorithms minimize the effect of packet loss during a transfer. Most importantly, NETBLT's features give it good performance over long-delay channels without impairing performance over high-speed LANs.

The protocol works by opening a connection between two "clients" (the "sender" and the "receiver"), transferring the data in a series of large data aggregates called "buffers", and then closing the connection. Because the amount of data to be transferred can be very large, the client is not required to provide at once all the data to the protocol module. Instead, the data is provided by the client in buffers. The NETBLT layer transfers each buffer as a sequence of packets; since each buffer is composed of a large number of packets, the per-buffer interaction between NETBLT and its client is far more efficient than a per-packet interaction would be.

In its simplest form, a NETBLT transfer works as follows: the sending client loads a buffer of data and calls down to the NETBLT layer to transfer it. The NETBLT layer breaks the buffer up into packets and sends these packets across the network in Internet datagrams. The receiving NETBLT layer loads these packets into a matching buffer provided by the receiving client. When the last packet in the buffer has arrived, the receiving NETBLT checks to see that all packets in that buffer have been correctly received. If some packets are missing, the receiving NETBLT requests that they be resent. When the buffer has been completely transmitted, the receiving client is notified by its NETBLT layer. The receiving client disposes of the buffer and provides a new buffer to receive more data. The receiving NETBLT notifies the sender that the new buffer is ready, and the sender prepares and sends the next buffer in the same manner. This continues until all the data has been sent; at that time the sender notifies the receiver that the transmission has been completed. The connection is then closed.

As described above, the NETBLT protocol is "lock-step". Action halts after a buffer is transmitted, and begins again after confirmation is received from the receiver of data. NETBLT provides for multiple buffering, a transfer model in which the sending NETBLT can transmit new buffers while earlier buffers are waiting for confirmation from the receiving NETBLT. Multiple buffering makes packet flow essentially continuous and markedly improves performance.

The remainder of this document describes NETBLT in detail. The next sections describe the philosophy behind a number of protocol features: packetization, flow control, transfer reliability, and connection management. The final sections describe NETBLT's packet formats.

3. Buffers and Packets

NETBLT is designed to permit transfer of a very large amounts of data between two clients. During connection setup the sending NETBLT can inform the receiving NETBLT of the transfer size; the maximum transfer length is 2**32 bytes. This limit should permit any practical application. The transfer size parameter is for the use of the receiving client; the receiving NETBLT makes no use of it. A

NETBLT receiver accepts data until told by the sender that the transfer is complete.

The data to be sent must be broken up into buffers by the client. Each buffer must be the same size, save for the last buffer. During connection setup, the sending and receiving NETBLTs negotiate the buffer size, based on limits provided by the clients. Buffer sizes are in bytes only; the client is responsible for placing data in buffers on byte boundaries.

NETBLT has been designed and should be implemented to work with buffers of any size. The only fundamental limitation on buffer size should be the amount of memory available to the client. Buffers should be as large as possible since this minimizes the number of buffer transmissions and therefore improves performance.

NETBLT is designed to require a minimum amount of memory, allowing the client to allocate as much memory as possible for buffer storage. In particular, NETBLT does not keep buffer copies for retransmission purposes. Instead, data to be retransmitted is recopied directly from the client buffer. This means that the client cannot release buffer storage piece by piece as the buffer is sent, but this has not been a problem in preliminary NETBLT implementations.

Buffers are broken down by the NETBLT layer into sequences of DATA packets. As with the buffer size, the DATA packet size is negotiated between the sending and receiving NETBLTs during connection setup. Unlike buffer size, DATA packet size is visible only to the NETBLT layer.

All DATA packets save the last packet in a buffer must be the same size. Packets should be as large as possible, since NETBLT's performance is directly related to packet size. At the same time, the packets should not be so large as to cause internetwork fragmentation, since this normally causes performance degradation.

All buffers save the last buffer must be the same size; the last buffer can be any size required to complete the transfer. Since the receiving NETBLT does not know the transfer size in advance, it needs some way of identifying the last packet in each buffer. For this reason, the last packet of every buffer is not a DATA packet but rather an LDATA packet. DATA and LDATA packets are identical save for the packet type.

4. Flow Control

NETBLT uses two strategies for flow control, one internal and one at the client level.

The sending and receiving NETBLTs transmit data in buffers; client flow control is therefore at a buffer level. Before a buffer can be transmitted, NETBLT confirms that both clients have set up matching buffers, that one is ready to send data, and that the other is ready to receive data. Either client can therefore control the flow of data by not providing a new buffer. Clients cannot stop a buffer transfer once it is in progress.

Since buffers can be quite large, there has to be another method for flow control that is used during a buffer transfer. The NETBLT layer provides this form of flow control.

There are several flow control problems that could arise while a buffer is being transmitted. If the sending NETBLT is transferring data faster than the receiving NETBLT can process it, the receiver's ability to buffer unprocessed packets could be overflowed, causing packet loss. Similarly, a slow gateway or intermediate network could cause packets to collect and overflow network packet buffer space. Packets will then be lost within the network. This problem is particularly acute for NETBLT because NETBLT buffers will generally be quite large, and therefore composed of many packets.

A traditional solution to packet flow control is a window system, in which the sending end is permitted to send only a certain number of packets at a time. Unfortunately, flow control using windows tends to result in low throughput. Windows must be kept small in order to avoid overflowing hosts and gateways, and cannot easily be updated, since an end-to-end exchange is required for each window change.

To permit high throughput over a variety of networks and gateways, NETBLT uses a novel flow control method: rate control. The transmission rate is negotiated by the sending and receiving NETBLTs during connection setup and after each buffer transmission. The sender uses timers, rather than messages from the receiver, to maintain the negotiated rate.

In its simplest form, rate control specifies a minimum time period per packet transmission. This can cause performance problems for several reasons. First, the transmission time for a single packet is very small, frequently smaller than the granularity of the timing mechanism. Also, the overhead required to maintain timing mechanisms on a per packet basis is relatively high and lowers performance.

The solution is to control the transmission rate of groups of packets, rather than single packets. The sender transmits a burst of packets over a negotiated time interval, then sends another burst. In this way, the overhead decreases by a factor of the burst size, and the per-burst transmission time is long enough that timing mechanisms will work properly. NETBLT's rate control therefore has two parts, a burst size and a burst rate, with (burst size)/(burst rate) equal to the average transmission time per packet.

The burst size and burst rate should be based not only on the packet transmission and processing speed which each end can handle, but also on the capacities of any intermediate gateways or networks. Following are some intuitive values for packet size, buffer size, burst size, and burst rate.

Packet sizes can be as small as 128 bytes. Performance with packets this small is almost always bad, because of the high per-packet processing overhead. Even the default Internet Protocol packet size of 576 bytes is barely big enough for adequate performance. Most networks do not support packet sizes much larger than one or two thousand bytes, and packets of this size can also get fragmented when traveling over intermediate networks, lowering performance.

The size of a NETBLT buffer is limited only by the amount of memory available to a client. Theoretically, buffers of 100 Kbytes or more are possible. This would mean the transmission of 50 to 100 packets per buffer.

The burst size and burst rate are obviously very machine dependent. There is a certain amount of transmission overhead in the sending and receiving machines associated with maintaining timers and scheduling processes. This overhead can be minimized by sending packets in large bursts. There are also limitations imposed on the burst size by the number of available packet buffers in the operating system kernel. On most modern operating systems, a burst size of between five and ten packets should reduce the overhead to an acceptable level. A preliminary NETBLT implementation for the IBM PC/AT sends packets in bursts of five. It could send more, but is limited by the available memory.

The burst rate is in part determined by the granularity of the sender's timing mechanism, and in part by the processing speed of the receiver and any intermediate gateways. It is also directly related to the burst size. Burst rates from 20 to 45 milliseconds per 5-packet burst have been tried on the IBM PC/AT and Symbolics 3600 NETBLT implementations with good results within a single local-area network. This value clearly depends on the network bandwidth and packet buffering available.

All NETBLT flow control parameters (packet size, buffer size, burst size, and burst rate) are negotiated during connection setup. The negotiation process is the same for all parameters. The client initiating the connection (the active end) proposes and sends a set of values for each parameter in its connection request. The other client (the passive end) compares these values with the highest-performance values it can support. The passive end can then modify any of the parameters, but only by making them more restrictive. The modified parameters are then sent back to the active end in its response message.

The burst size and burst rate can also be re-negotiated after each buffer transmission to adjust the transfer rate according to the performance observed from transferring the previous buffer. The receiving end sends burst size and burst rate values in its OK messages (described later). The sender compares these values with the values it can support. Again, it may then modify any of the parameters, but only by making them more restrictive. The modified parameters are then communicated to the receiver in a NULL-ACK packet, described later.

Obviously each of the parameters depend on many factors -- gateway and host processing speeds, available memory, timer granularity -- some of which cannot be checked by either client. Each client must therefore try to make as best a guess as it can, tuning for performance on subsequent transfers.

5. The NETBLT Transfer Model

Each NETBLT transfer has three stages, connection setup, data transfer, and connection close. The stages are described in detail below, along with methods for insuring that each stage completes reliably.

5.1. Connection Setup

A NETBLT connection is set up by an exchange of two packets between the active NETBLT and the passive NETBLT. Note that either NETBLT can send or receive data; the words "active" and "passive" are only used to differentiate the end making the connection request from the end responding to the connection request. The active end sends an OPEN packet; the passive end acknowledges the OPEN packet in one of two ways. It can either send a REFUSED packet, indicating that the connection cannot be completed for some reason, or it can complete the connection setup by sending a RESPONSE packet. At this point the transfer can begin.

As discussed in the previous section, the OPEN and RESPONSE packets are used to negotiate flow control parameters. Other parameters used in the data transfer are also negotiated. These parameters are (1) the maximum number of buffers that can be sending at any one time, and (2) whether or not DATA packet data will be checksummed. NETBLT automatically checksums all non-DATA/LDATA packets. If the negotiated checksum flag is set to TRUE (1), both the header and the data of a DATA/LDATA packet are checksummed; if set to FALSE (0), only the header is checksummed. The checksum value is the bitwise negation of the ones-complement sum of the 16-bit words being checksummed.

Finally, each end transmits its death-timeout value in seconds in either the OPEN or the RESPONSE packet. The death-timeout value will be used to determine the frequency with which to send KEEPALIVE packets during idle periods of an opened connection (death timers and KEEPALIVE packets are described in the following section).

The active end specifies a passive client through a client-specific "well-known" 16 bit port number on which the passive end listens. The active end identifies itself through a 32 bit Internet address and a unique 16 bit port number.

In order to allow the active and passive ends to communicate miscellaneous useful information, an unstructured, variable-length field is provided in OPEN and RESPONSE packets for any client-specific information that may be required. In addition, a "reason for refusal" field is provided in REFUSED packets.

Recovery for lost OPEN and RESPONSE packets is provided by the use of timers. The active end sets a timer when it sends an OPEN packet. When the timer expires, another OPEN packet is sent, until some predetermined maximum number of OPEN packets have been sent. The timer is cleared upon receipt of a RESPONSE packet.

To prevent duplication of OPEN and RESPONSE packets, the OPEN packet contains a 32 bit connection unique ID that must be returned in the RESPONSE packet. This prevents the initiator from confusing the response to the current request with the response to an earlier connection request (there can only be one connection between any two ports). Any OPEN or RESPONSE packet with a destination port matching that of an open connection has its unique ID checked. If the unique ID of the packet matches the unique ID of the connection, then the packet type is checked. If it is a RESPONSE packet, it is treated as a duplicate and ignored. If it is an OPEN packet, the passive NETBLT sends another RESPONSE (assuming that a previous RESPONSE packet was sent and lost, causing the initiating NETBLT to retransmit its OPEN packet). A non-matching unique ID must be treated as an attempt to open a second connection between the same port pair and is rejected by sending an ABORT message.

5.2. Data Transfer

The simplest model of data transfer proceeds as follows. The sending client sets up a buffer full of data. The receiving NETBLT sends a GO message inside a CONTROL packet to the sender, signifying that it too has set up a buffer and is ready to receive data. Once the GO message is received, the sender transmits the buffer as a series of DATA packets followed by an LDATA packet. When the last packet in the buffer has been received, the receiver sends a RESEND message inside a CONTROL packet containing a list of packets that were not received. The sender resends these packets. This process continues until there are no missing packets. At that time the receiver sends an OK message inside a CONTROL packet, sets up another buffer to receive data, and sends another GO message. The sender, having received the OK message, sets up another buffer, waits for the GO message, and repeats the process.

The above data transfer model is effectively a lock-step protocol, and causes time to be wasted while the sending NETBLT waits for permission to send a new buffer. A more efficient transfer model uses multiple buffering to increase performance. Multiple buffering is a technique in which the sender and receiver allocate and transmit buffers in a manner that allows error recovery or successful transmission confirmation of previous buffers to be concurrent with transmission of the current buffer.

During the connection setup phase, one of the negotiated parameters is the number of concurrent buffers permitted during the transfer. If there is more than one buffer available, transfer of the next buffer may start right after the current buffer finishes. This is illustrated in the following example:

Assume two buffers A and B in a multiple-buffer transfer, with A preceding B. When A has been transferred and the sending NETBLT is waiting for either an OK or a RESEND message for it, the sending NETBLT can start sending B immediately, keeping data flowing at a stable rate. If the receiver of data sends an OK for A, all is well; if it receives a RESEND, the missing packets specified in the RESEND message are retransmitted.

In the multiple-buffer transfer model, all packets to be sent are re-ordered by buffer number (lowest number first), with the transfer rate specified by the burst size and burst rate. Since buffer numbers increase monotonically, packets from an earlier buffer will always precede packets from a later buffer.

Having several buffers transmitting concurrently is actually not that much more complicated than transmitting a single buffer at a time. The key is to visualize each buffer as a finite state machine; several buffers are merely a group of finite state machines, each in one of several states. The transfer process consists of moving buffers through various states until the entire transmission has completed.

There are several obvious flaws in the data transfer model as described above. First, what if the GO, OK, or RESEND messages are lost? The sender cannot act on a packet it has not received, so the protocol will hang. Second, if an LDATA packet is lost, how does the receiver know when the buffer has been transmitted? Solutions for each of these problems are presented below.

5.2.1. Recovering from Lost Control Messages

NETBLT solves the problem of lost OK, GO, and RESEND messages in two ways. First, it makes use of a control timer. The receiver can send one or more control messages (OK, GO, or RESEND) within a single

CONTROL packet. Whenever the receiver sends a control packet, it sets a control timer. This timer is either "reset" (set again) or "cleared" (deactivated), under the following conditions:

When the control timer expires, the receiving NETBLT resends the control packet and resets the timer. The receiving NETBLT continues to resend control packets in response to control timer's expiration until either the control timer is cleared or the receiving NETBLT's death timer (described later) expires (at which time it shuts down the connection).

Each control message includes a sequence number which starts at one and increases by one for each control message sent. The sending NETBLT checks the sequence number of every incoming control message against all other sequence numbers it has received. It stores the highest sequence number below which all other received sequence numbers are consecutive (in following paragraphs this is called the high-acknowledged-sequence-number) and returns this number in every packet flowing back to the receiver. The receiver is permitted to clear its control timer when it receives a packet from the sender with a high-acknowledged-sequence-number greater than or equal to the highest sequence number in the control packet just sent.

Ideally, a NETBLT implementation should be able to cope with out-of-sequence control messages, perhaps collecting them for later processing, or even processing them immediately. If an incoming control message "fills" a "hole" in a group of message sequence numbers, the implementation could even be clever enough to detect this and adjust its outgoing sequence value accordingly.

The sending NETBLT, upon receiving a CONTROL packet, should act on the packet as quickly as possible. It either sets up a new buffer (upon receipt of an OK message for a previous buffer), marks data for resending (upon receipt of a RESEND message), or prepares a buffer for sending (upon receipt of a GO message). If the sending NETBLT is not in a position to send data, it should send a NULL-ACK packet, which contains its high-acknowledged-sequence-number (this permits the receiving NETBLT to acknowledge any outstanding control messages), and wait until it can send more data. In all of these cases, the system overhead for a response to the incoming control message should be small and relatively constant.

The small amount of message-processing overhead allows accurate control timers to be set for all types of control messages with a single, simple algorithm -- the network round-trip transit time, plus a variance factor. This is more efficient than schemes used by other protocols, where timer value calculation has been a problem because the processing time for a particular packet can vary greatly depending on the packet type.

Control timer value estimation is extremely important in a high-performance protocol like NETBLT. A long control timer causes the receiving NETBLT to wait for long periods of time before retransmitting unacknowledged messages. A short control timer value causes the sending NETBLT to receive many duplicate control messages (which it can reject, but which takes time).

In addition to the use of control timers, NETBLT reduces lost control messages by using a single long-lived control packet; the packet is treated like a FIFO queue, with new control messages added on at the end and acknowledged control messages removed from the front. The implementation places control messages in the control packet and transmits the entire control packet, consisting of any unacknowledged control messages plus new messages just added. The entire control packet is also transmitted whenever the control timer expires. Since control packet transmissions are fairly frequent, unacknowledged messages may be transmitted several times before they are finally acknowledged. This redundant transmission of control messages provides automatic recovery for most control message losses over a noisy channel.

This scheme places some burdens on the receiver of the control messages. It must be able to quickly reject duplicate control messages, since a given message may be retransmitted several times before its acknowledgement is received and it is removed from the control packet. Typically this is fairly easy to do; the sender of data merely throws away any control messages with sequence numbers lower than its high-acknowledged-sequence-number.

Another problem with this scheme is that the control packet may become larger than the maximum allowable packet size if too many control messages are placed into it. This has not been a problem in the current NETBLT implementations: a typical control packet size is 1000 bytes; RESEND control messages average about 20 bytes in length, GO messages are 8 bytes long, and OK messages are 16 bytes long. This allows 50-80 control messages to be placed in the control packet, more than enough for reasonable transfers. Other implementations can provide for multiple control packets if a single control packet may not be sufficient.

The control timer value must be carefully estimated. It can have as its initial value an arbitrary number. Subsequent control packets should have their timer values based on the network round-trip transit time (i.e. the time between sending the control packet and receiving the acknowledgment of all messages in the control packet) plus a variance factor. The timer value should be continually updated, based on a smoothed average of collected round-trip transit times.

5.2.2. Recovering from Lost LDATA Packets

NETBLT solves the problem of LDATA packet loss by using a data timer for each buffer at the receiving end. The simplest data timer model has a data timer set when a buffer is ready to be received; if the data timer expires, the receiving NETBLT assumes a lost LDATA packet and sends a RESEND message requesting all missing DATA packets in the buffer. When all packets have been received, the timer is cleared.

Data timer values are not based on network round-trip transit time; instead they are based on the amount of time taken to transfer a buffer (as determined by the number of DATA packet bursts in the buffer times the burst rate) plus a variance factor <1>.

Obviously an accurate estimation of the data timer value is very important. A short data timer value causes the receiving NETBLT to send unnecessary RESEND packets. This causes serious performance degradation since the sending NETBLT has to stop what it is doing and resend a number of DATA packets.

Data timer setting and clearing turns out to be fairly complicated, particularly in a multiple-buffering transfer model. In understanding how and when data timers are set and cleared, it is helpful to visualize each buffer as a finite-state machine and take a look at the various states.

The state sequence for a sending buffer is simple. When a GO message for the buffer is received, the buffer is created, filled with data, and placed in a SENDING state. When an OK for that buffer has been received, it goes into a SENT state and is disposed of.

The state sequence for a receiving buffer is a little more complicated. Assume existence of a buffer A. When a control message for A is sent, the buffer moves into state ACK-WAIT (it is waiting for acknowledgement of the control message).

As soon as the control message has been acknowledged, buffer A moves from the ACK-WAIT state into the ACKED state (it is now waiting for DATA packets to arrive). At this point, A's data timer is set and the control message removed from the control packet. Estimation of the data timer value at this point is quite difficult. In a multiple-buffer transfer model, the receiving NETBLT can send several GO messages at once. A single DATA packet from the sending NETBLT could acknowledge all the GO messages, causing several buffers to start up data timers. Clearly each of the data timers must be set in a manner that takes into account each buffer's place in the order of transmission. Packets for a buffer A - 1 will always be transmitted before packets in A, so A's data timer must take into account the arrival of all of A - 1's DATA packets as well as arrival of its own DATA packets. This means that the timer values become increasingly less accurate for higher-numbered buffers. Because this data timer value can be quite inaccurate, it is called a "loose" data timer. The loose data timer value is recalculated later (using the same algorithm, but with updated information), giving a "tight" timer, as described below.

When the first DATA packet for A arrives, A moves from the ACKED state to the RECEIVING state and its data timer is set to a new "tight" value. The tight timer value is calculated in the same manner as the loose timer, but it is more accurate since we have moved forward in time and those buffers numbered lower than A have presumably been dealt with (or their packets would have arrived before A's), leaving fewer packets to arrive between the setting of the data timer and the arrival of the last DATA packet in A.

The receiving NETBLT also sets the tight data timers of any buffers numbered lower than A that are also in the ACKED state. This is done as an optimization: we know that buffers are processed in order, lowest number first. If a buffer B numbered lower than A is in the ACKED state, its DATA packets should arrive before A's. Since A's have arrived first, B's must have gotten lost. Since B's loose data timer has not expired (it would then have sent a RESEND message and be in the ACK-WAIT state), we set the tight timer, allowing the missing packets to be detected earlier. An immediate RESEND is not sent because it is possible that A's packet was re-ordered before B's by the network, and that B's packets may arrive shortly.

When all DATA packets for A have been received, it moves from the RECEIVING state to the RECEIVED state and is disposed of. Had any packets been missing, A's data timer would have expired and A would have moved into the ACK-WAIT state after sending a RESEND message. The state progression would then move as in the above example.

The control and data timer system can be summarized as follows: normally, the receiving NETBLT is working under one of two types of timers, a control timer or a data timer. There is one data timer per buffer transmission and one control timer per control packet. The data timer is active while its buffer is in either the ACKED (loose data timer value is used) or the RECEIVING (tight data timer value is used) states; a control timer is active whenever the receiving NETBLT has any unacknowledged control messages in its control packet.

5.2.3. Death Timers and Keepalive Packets

The above system still leaves a few problems. If the sending NETBLT is not ready to send, it sends a single NULL-ACK packet to clear any outstanding control timers at the receiving end. After this the receiver will wait. The sending NETBLT could die and the receiver, with its control timer cleared, would hang. Also, the above system puts timers only on the receiving NETBLT. The sending NETBLT has no timers; if the receiving NETBLT dies, the sending NETBLT will hang while waiting for control messages to arrive.

The solution to the above two problems is the use of a death timer and a keepalive packet for both the sending and receiving NETBLTs. As soon as the connection is opened, each end sets a death timer; this timer is reset every time a packet is received. When a NETBLT's death timer expires, it can assume the other end has died and can close the connection.

It is possible that the sending or receiving NETBLTs will have to wait for long periods while their respective clients get buffer space and load their buffers with data. Since a NETBLT waiting for buffer space is in a perfectly valid state, the protocol must have some method for preventing the other end's death timer from expiring. The solution is to use a KEEPALIVE packet, which is sent repeatedly at fixed intervals when a NETBLT cannot send other packets. Since the death timer is reset whenever a packet is received, it will never expire as long as the other end sends packets.

The frequency with which KEEPALIVE packets are transmitted is computed as follows: At connection startup, each NETBLT chooses a death-timer value and sends it to the other end in either the OPEN or the RESPONSE packet. The other end takes the death-timeout value and uses it to compute a frequency with which to send KEEPALIVE packets. The KEEPALIVE frequency should be high enough that several KEEPALIVE packets can be lost before the other end's death timer expires (e.g. death timer value divided by four).

The death timer value is relatively easy to estimate. Since it is continually reset, it need not be based on the transfer size. Instead, it should be based at least in part on the type of application using NETBLT. User applications should have smaller death timeout values to avoid forcing humans to wait long periods of time for a death timeout to occur. Machine applications can have longer timeout values.

5.3. Closing the Connection

There are three ways to close a connection: a connection close, a "quit", or an "abort".

5.3.1. Successful Transfer

After a successful data transfer, NETBLT closes the connection. When the sender is transmitting the last buffer of data, it sets a "last-buffer" flag on every DATA packet in the buffer. This means that no NEW data will be transmitted. The receiver knows the transfer has completed successfully when all of the following are true: (1) it has received DATA packets with a "last-buffer" flag set, (2) all its control messages have been acknowledged, and (3) it has no outstanding buffers with missing packets. At that point, the receiver is permitted to close its half of the connection. The sender knows the transfer has completed when the following are true:

(1) it has transmitted DATA packets with a "last-buffer" flag set and (2) it has received OK messages for all its buffers. At that point, it "dallies" for a predetermined period of time before closing its half of the connection. If the NULL-ACK packet acknowledging the receiver's last OK message was lost, the receiver has time to retransmit the OK message, receive a new NULL-ACK, and recognize a successful transfer. The dally timer value MUST be based on the receiver's control timer value; it must be long enough to allow the receiver's control timer to expire so that the OK message can be re-sent. For this reason, all OK messages contain (in addition to new burst size and burst rate values), the receiver's current control timer value in milliseconds. The sender uses this value to compute its dally timer value.

Since the dally timer value may be quite large, the receiving NETBLT is permitted to "short-circuit" the sending NETBLT's dally timer by transmitting a DONE packet. The DONE packet is transmitted when the receiver knows the transfer has been successfully completed. When the sender receives a DONE packet, it is allowed to clear its dally timer and close its half of the connection immediately. The DONE packet is not reliably transmitted, since failure to receive it only means that the sending NETBLT will take longer time to close its half of the connection (as it waits for its dally timer to clear)

5.3.2. Client QUIT

During a NETBLT transfer, one client may send a QUIT packet to the other if it thinks that the other client is malfunctioning. Since the QUIT occurs at a client level, the QUIT transmission can only occur between buffer transmissions. The NETBLT receiving the QUIT packet can take no action other than immediately notifying its client and transmitting a QUITACK packet. The QUIT sender must time out and retransmit until a QUITACK has been received or its death timer expires. The sender of the QUITACK dallies before quitting, so that it can respond to a retransmitted QUIT.

5.3.3. NETBLT ABORT

An ABORT takes place when a NETBLT layer thinks that it or its opposite is malfunctioning. Since the ABORT originates in the NETBLT layer, it can be sent at any time. The ABORT implies that the NETBLT layer is malfunctioning, so no transmit reliability is expected, and the sender can immediately close it connection.

6. Protocol Layering Structure

NETBLT is implemented directly on top of the Internet Protocol (IP). It has been assigned an official protocol number of 30 (decimal).

7. Planned Enhancements

As currently specified, NETBLT has no algorithm for determining its rate-control parameters (burst rate, burst size, etc.). In initial performance testing, these parameters have been set by the person performing the test. We are now exploring ways to have NETBLT set and adjust its rate-control parameters automatically.

8. Packet Formats

NETBLT packets are divided into three categories, all of which share a common packet header. First, there are those packets that travel only from data sender to receiver; these contain the high-acknowledged-sequence-numbers which the receiver uses for control message transmission reliability. These packets are the NULL-ACK, DATA, and LDATA packets. Second, there is a packet that travels only from receiver to sender. This is the CONTROL packet; each CONTROL packet can contain an arbitrary number of control messages (GO, OK, or RESEND), each with its own sequence number. Finally, there are those packets which either have special ways of insuring reliability, or are not reliably transmitted. These are the OPEN, RESPONSE, REFUSED, QUIT, QUITACK, DONE, KEEPALIVE, and ABORT packets. Of these, all save the DONE packet can be sent by both sending and receiving NETBLTs.

All packets are "longword-aligned", i.e. all packets are a multiple of 4 bytes in length and all 4-byte fields start on a longword boundary. All arbitrary-length string fields are terminated with at least one null byte, with extra null bytes added at the end to create a field that is a multiple of 4 bytes long.

Packet Formats for NETBLT

OPEN (type 0) and RESPONSE (type 1):

```
                       1                   2                   3
   1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
  +---------------+---------------+---------------+---------------+
  |            Checksum           |    Version    |     Type      |
  +---------------+---------------+---------------+---------------+
  |            Length             |          Local Port           |
  +---------------+---------------+---------------+---------------+
  |         Foreign Port          | Longword Alignment Padding    |
  +---------------+---------------+---------------+---------------+
  |                     Connection Unique ID                      |
  +---------------+---------------+---------------+---------------+
  |                          Buffer Size                          |
  +---------------+---------------+---------------+---------------+
  |                        Transfer Size                          |
  +---------------+---------------+---------------+---------------+
  |        DATA packet size       |          Burst Size           |
  +---------------+---------------+---------------+---------------+
  |           Burst Rate          |       Death Timer Value       |
  +---------------+---------------+---------------+---------------+
  |       Reserved (MBZ)      |C|M| Maximum # Outstanding Buffers |
  +---------------+---------------+---------------+---------------+
  | Client String ...
  +---------------+---------------+---------------
                                   Longword Alignment Padding    |
                  ---------------+-------------------------------+
```

Checksum: packet checksum (algorithm is described in the section "Connection Setup")

Version: the NETBLT protocol version number

Type: the NETBLT packet type number (OPEN = 0, RESPONSE = 1, etc.)

Length: the total length (NETBLT header plus data, if present) of the NETBLT packet in bytes

Local Port: the local NETBLT's 16-bit port number

Foreign Port: the foreign NETBLT's 16-bit port number

Connection UID: the 32 bit connection UID specified in the section "Connection Setup".

Buffer size: the size in bytes of each NETBLT buffer (save the last)

Transfer size: (optional) the size in bytes of the transfer.

This is for client information only; the receiving NETBLT should NOT make use of it.

Data packet size: length of each DATA packet in bytes

Burst Size: Number of DATA packets in a burst

Burst Rate: Transmit time in milliseconds of a single burst

Death timer: Packet sender's death timer value in seconds

"M": the transfer mode (0 = READ, 1 = WRITE)

"C": the DATA packet data checksum flag (0 = do not checksum DATA packet data, 1 = do)

Maximum Outstanding Buffers: maximum number of buffers that can be transferred before waiting for an OK message from the receiving NETBLT.

Client string: an arbitrary, null-terminated, longword-aligned string for use by NETBLT clients.

KEEPALIVE (type 2), QUITACK (type 4), and DONE (type 11)

```
                     1                   2                   3
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+---------------+---------------+---------------+---------------+
|           Checksum            |    Version    |     Type      |
+---------------+---------------+---------------+---------------+
|           Length              |          Local Port           |
+---------------+---------------+---------------+---------------+
|        Foreign Port           | Longword Alignment Padding    |
+---------------+---------------+---------------+---------------+
```

QUIT (type 3), ABORT (type 5), and REFUSED (type 10)

```
                     1                   2                   3
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+---------------+---------------+---------------+---------------+
|           Checksum            |    Version    |     Type      |
+---------------+---------------+---------------+---------------+
|           Length              |          Local Port           |
+---------------+---------------+---------------+---------------+
|        Foreign Port           | Longword Alignment Padding    |
+---------------+---------------+---------------+---------------+
| Reason for QUIT/ABORT/REFUSE...
+---------------+---------------+---------------
                                 Longword Alignment Padding    |
                 ---------------+-------------------------------+
```

DATA (type 6) and LDATA (type 7):

```
                     1                   2                   3
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+---------------+---------------+---------------+---------------+
|           Checksum            |    Version    |     Type      |
+---------------+---------------+---------------+---------------+
|           Length              |          Local Port           |
+---------------+---------------+---------------+---------------+
|        Foreign Port           | Longword Alignment Padding    |
+---------------+---------------+---------------+---------------+
|                         Buffer Number                         |
+---------------+---------------+---------------+---------------+
| High Consecutive Seq Num Rcvd |         Packet Number         |
+---------------+---------------+---------------+---------------+
|   Data Area Checksum Value    |      Reserved (MBZ)         |L|
+---------------+---------------+---------------+---------------+
```

Buffer number: a 32 bit unique number assigned to every buffer. Numbers are monotonically increasing.

High Consecutive Sequence Number Received: Highest control message sequence number below which all sequence numbers received are consecutive.

Packet number: monotonically increasing DATA packet identifier

Data Area Checksum Value: Checksum of the DATA packet's data. Algorithm used is the same as that used to compute checksums of other NETBLT packets.

"L" is a flag set when the buffer that this DATA packet belongs to is the last buffer in the transfer.

NULL-ACK (type 8)

```
                     1                   2                   3
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+---------------+---------------+---------------+---------------+
|           Checksum            |    Version    |     Type      |
+---------------+---------------+---------------+---------------+
|           Length              |          Local Port           |
+---------------+---------------+---------------+---------------+
|        Foreign Port           | Longword Alignment Padding    |
+---------------+---------------+---------------+---------------+
| High Consecutive Seq Num Rcvd |        New Burst Size         |
+---------------+---------------+---------------+---------------+
|       New Burst Rate          |  Longword Alignment Padding   |
+---------------+---------------+---------------+---------------+
```

High Consecutive Sequence Number Received: same as in DATA/LDATA packet

New Burst Size: Burst size as negotiated from value given by receiving NETBLT in OK message

New burst rate: Burst rate as negotiated from value given by receiving NETBLT in OK message. Value is in milliseconds.

CONTROL (type 9):

```
                     1                   2                   3
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+---------------+---------------+---------------+---------------+
|           Checksum            |    Version    |     Type      |
+---------------+---------------+---------------+---------------+
|           Length              |          Local Port           |
+---------------+---------------+---------------+---------------+
|        Foreign Port           | Longword Alignment Padding    |
+---------------+---------------+---------------+---------------+
```

Followed by any number of messages, each of which is longword aligned, with the following formats:

GO message (type 0):

```
                     1                   2                   3
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+---------------+---------------+---------------+---------------+
|    Type       | Word Padding  |        Sequence Number        |
+---------------+---------------+---------------+---------------+
|                        Buffer Number                          |
+---------------+---------------+---------------+---------------+
```

Type: message type (GO = 0, OK = 1, RESEND = 2)

   Sequence number: A 16 bit unique message number. Sequence
   numbers must be monotonically increasing, starting from 1.

   Buffer number: as in DATA/LDATA packet

   OK message (type 1):

```
                           1                   2                   3
    1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
   +---------------+---------------+---------------+---------------+
   |     Type      | Word Padding  |        Sequence Number        |
   +---------------+---------------+---------------+---------------+
   |                         Buffer Number                         |
   +---------------+---------------+---------------+---------------+
   |    New Offered Burst Size     |   New Offered Burst Rate      |
   +---------------+---------------+---------------+---------------+
   | Current control timer value   | Longword Alignment Padding    |
   +---------------+---------------+---------------+---------------+
```

   New offered burst size: burst size for subsequent buffer
   transfers, possibly based on performance information for previous
   buffer transfers.

   New offered burst rate: burst rate for subsequent buffer
   transfers, possibly based on performance information for previous
   buffer transfers. Rate is in milliseconds.

   Current control timer value: Receiving NETBLT's control timer
   value in milliseconds.

   RESEND Message (type 2):

```
                           1                   2                   3
    1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
   +---------------+---------------+---------------+---------------+
   |     Type      | Word Padding  |        Sequence Number        |
   +---------------+---------------+---------------+---------------+
   |                         Buffer Number                         |
   +---------------+---------------+---------------+---------------+
   |  Number of Missing Packets    | Longword Alignment Padding    |
   +---------------+---------------+---------------+---------------+
   |       Packet Number (2 bytes) ...
   +---------------+---------------+----------
                                   |    Padding (if necessary)     |
                        -----------+---------------+---------------+
```

   Packet number: the 16 bit data packet identifier found in each
   DATA packet.

NOTES:

<1> When the buffer size is large, the variances in the round trip delays of many packets may cancel each other out; this means the variance value need not be very big. This expectation will be explored in further testing.