Network Working Group
Request for Comments: 969

David D. Clark
Mark L. Lambert
Lixia Zhang
M. I. T. Laboratory for Computer Science
December 1985

NETBLT: A Bulk Data Transfer Protocol

1. STATUS OF THIS MEMO

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. This is a preliminary discussion of the NETBLT protocol. It is published for discussion and comment, and does not constitute a standard. As the proposal may change, implementation of this document is not advised. Distribution of this memo is unlimited.

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 structured to provide maximum throughput over a wide variety of networks.

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 arbitrarily 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, but 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 been transmitted, the receiving NETBLT checks to see that all packets in that buffer have arrived. 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 buffer arrived, and the sender prepares and sends the next buffer in the

same manner. This continues until all buffers have been sent, at which 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 is halted after a buffer is transmitted, and begins again after confirmation is received from the receiver of data. NETBLT provides for multiple buffering, in which several buffers can be transmitted concurrently. Multiple buffering makes packet flow essentially continuous and can improve performance markedly.

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

3. BUFFERS AND PACKETS

NETBLT is designed to permit transfer of an essentially arbitrary amount of data between two clients. During connection setup the sending NETBLT can optionally inform the receiving NETBLT of the transfer size; the maximum transfer length is imposed by the field width, and 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 breaking up data into buffers on byte boundaries.

NETBLT has been designed and should be implemented to work with buffers of arbitrary 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 of its own 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 does mean that the client cannot release buffer storage piece by piece as the buffer is sent, but this has not proved 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 packet size is negotiated between the sending and receiving NETBLTs during connection setup. Unlike buffer size, 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 in most cases (including the preliminary protocol implementation) performance is directly related to packet size. At the same time, the packets should not be so large as to cause Internet fragmentation, since this normally causes performance degrada- tion.

All buffers save the last buffer must be the same size; obviously 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 while 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 packets to be lost. 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, degrading performance. 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 change.

To permit high throughput over a variety of networks and gateways of differing speeds, NETBLT uses a novel flow control ethod: 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: 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, which degrades 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 negotiated interval, then sends another burst. In this way, the overhead decreases by a factor of the burst size, and the per-burst transmission rate is large enough that timing mechanisms will work properly. The 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 rate 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 those gateways and networks intermediate to the transfer. 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, degrading performance.

The size of a NETBLT buffer is limited only by the amount of memory available to a client. Theoretically, buffers of 100K bytes 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. On most modern operating systems, a burst size of between five and ten packets should reduce the overhead to an acceptable level. In fact, a preliminary NETBLT implementation for the IBM PC/AT sends packets in bursts of five. It could send more, but is limited by 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 60 to 100 milliseconds have been tried on the preliminary NETBLT implementation 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 with its open 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 only by making them more restrictive. The modified parameters are then sent back to the active end in the response message. In addition, the burst size and burst rate can 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 a pair of burst size and burst rate values in the OK message. The sender compares these values with the values it can support. Again, it may then modify any of the parameters 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. RELIABILITY

Each NETBLT transfer has three stages, connection setup, data transfer, and connection close. Each stage must be completed reliably; methods for doing this are described below.

5.1. Connection Setup

A NETBLT connection is set up by an exchange of two packets between the active client and the passive client. Note that either client can send or receive data; the words "active" and "passive" are only used to differentiate the client initiating the connection process from the client responding to the connection request. The first packet sent is an OPEN packet; the passive end acknowledges the OPEN packet by sending a RESPONSE packet. After these two packets have been exchanged, 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 transfer of data are also negotiated. These parameters are (1) the maximum number of buffers that can be sending at any one time (this permits multiple buffering and higher throughput) and (2) whether or not DATA/LDATA 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. NETBLT uses the same checksumming algorithm as TCP uses.

Finally, each end transmits its death-timeout value 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 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 messages for an client-specific information that may be required.

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 pre-determined maximum number of OPEN packets have been sent. A similar scheme is used for the passive end when it sends a RESPONSE packet. When a RESPONSE packet is received by the active end, it clears its timer. The passive end's timer is cleared either by receipt of a GO or a DATA packet, as described in the section on data transfer.

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. A matching unique ID implies a duplicate packet, and the packet is ignored. A non-matching unique ID must be treated as an attempt to open a second connection between the same port pair and must be 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 into it. Once the GO message has been 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 which time the receiver sends an OK message inside a CONTROL packet to the sender, 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.

There are several obvious flaws with this scheme. First, if the LDATA packet is lost, how does the receiver know when the buffer has been transmitted? Second, 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. Solutions for each of these problems are presented below, and are based on two kinds of timers, a data timer and a control timer.

NETBLT solves the LDATA packet loss problem by using a data timer at the receiving end. When the first DATA packet in a buffer arrives, the receiving NETBLT sets its data timer; at the same time, it clears its control timer, described below. If the data timer expires, the receiving end assumes the buffer has been transmitted and all missing packets lost. It then sends a RESEND message containing a list of the missing packets.

NETBLT solves the second problem, that of missing OK, GO, and RESEND messages, through 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 (at the same time it clears its data timer, if one has been set).

The control timer is cleared as follows: 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, and returns this number in every packet flowing back to the receiver. The receiver is permitted to clear the control timer of every packet with a sequence number equal to or lower than the sequence number returned by the sender.

Ideally, a NETBLT implementation should be able to cope with out-of-sequence 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.

When the control timer expires, the receiving NETBLT resends the control message and resets the timer. After a predetermined number of resends, the receiving NETBLT can assume that the sending NETBLT has died, and can reset the connection.

The sending NETBLT, upon receiving a control message, should act as quickly as possible on the packet; it either sets up a new buffer (upon receipt of an OK packet for a previous buffer), resends data (upon receipt of a RESEND packet), or sends data

(upon receipt of a GO packet). If the sending NETBLT is not in a position to send data, it sends a NULL-ACK packet, which contains a high-received-sequence-number as described above (this permits the receiving NETBLT to clear the control timers of any packets which are outstanding), and waits until it can send more data. In all of these cases, the overhead for a response to the incoming control message should be small; the total time for a response to reach the receiving NETBLT should not be much more than the network round-trip transit time, plus a variance factor.

The 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 being transferred; a control timer is active while it is between buffer transfers.

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 all its control timers 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 just hang waiting for control messages.

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 at one end expires, it can assume the other end has died and can close the connection.

It is quite possible that the sending or receiving NETBLTs will have to wait for long periods of time 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 is waiting for buffer space. 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-timeout 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.

Both ends must have some way of estimating the values of the death timers, the control timers, and the data timers. The timer values obviously cannot be specified in a protocol document since they are very machine- and network-load-dependent. Instead they must be computed on a per-connection basis. The protocol has been designed to make such determination easy.

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.

The control timer must be more carefully estimated. It can have as its initial value an arbitrary number; this number can be used to send the first control packet. Subsequent control packets can 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 the corresponding sequence number) plus a variance factor. The timer value should be continually updated, based on a smoothed average of collected round-trip transit times.

The data timer is dependent not on the network round-trip transit time, but on the amount of time required to transfer a buffer of data. The time value can be computed from the burst rate and the number of bursts per buffer, plus a variance value <1>. During the RESENDing phase, the data timer value should be set according to the number of missing packets.

The timers have been designed to permit reasonable estimation. In particular, in other protocols, determination of round-trip delay has been a problem since the action performed by the other end on receipt of a particular packet can vary greatly depending on the packet type. In NETBLT, the action taken by the sender on receipt of a control message is by and large the same in all cases, making the round-trip delay relatively independent of the client.

Timer value estimation is extremely important, especially in a high-performance protocol like NETBLT. If the estimates are too low, the protocol makes many unneeded retransmissions, degrading performance. A short control timer value causes the sending NETBLT to receive duplicate control messages (which it can reject, but which takes time). A short data timer value causes the receiving NETBLT to send unnecessary RESEND packets. This causes considerably greater performance degradation since the sending NETBLT does not merely throw away a duplicate packet, but instead has to send a number of DATA packets. Because data timers are set on each buffer transfer instead of on each DATA packet transfer, we afford to use a small variance value without worrying about performance degradation.

5.3. Closing the Connection

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

The connection close occurs after a successful data transfer. When the sending NETBLT has received an OK packet for the last buffer in the transfer, it sends a DONE packet <2>. On receipt of the DONE packet, the receiving NETBLT can close its half of the connection. The sending NETBLT dallies for a predetermined amount of time after sending the DONE packet. This allows for the possibility of the DONE packet's having been lost. If the DONE packet was lost, the receiving NETBLT will continue to send the final OK packet, which will cause the sending end to resend the DONE packet. After the dally period expires, the sending NETBLT closes its half of the connection.

During the 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 to immediately notify its client and transmit a QUITACK packet. The QUIT sender must time out and retransmit until a QUITACK has been received or a predetermined number of resends have taken place. The sender of the QUITACK dallies in the manner described above.

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. Since the ABORT implies that the NETBLT layer is malfunctioning, no transmit reliability is expected, and the sender can immediately close it connection.

6. MULTIPLE BUFFERING

In order to increase performance, NETBLT has been designed in a manner that encourages a multiple buffering implementation. Multiple buffering is a technique in which the sender and receiver allocate and transmit buffers in a manner that allows error recovery of previous buffers to be concurrent with transmission of current buffer.

During the connection setup phase, one of the negotiated parameters is the number of concurrent buffers permitted during the transfer. The simplest transfer allows for a maximum of one buffer to be transmitted at a time; this is effectively a lock-step protocol and causes time to be wasted while the sending NETBLT receives permission to send a new buffer. If there are more than one buffer available, transfer of the next buffer may start right after the current buffer finishes. For example, assume buffer A and B are allowed to transfer concurrently, with A preceding B. As soon as A finishes transferring its data and is waiting for either an OK or a RESEND message, B can start sending immediately, keeping data flowing at a stable rate. If A receives an OK, it is done; if it receives a RESEND, the missing packets specified in the RESEND message are retransmitted. All packets flow out through a priority pipe, with the priority equal to the buffer number, and with the transfer rate specified by the burst size and burst rate. Since buffer numbers increase monotonically, packets from an earlier buffer in the pipe will always precede those of the later ones. One necessary change to the timing algorithm is that when the receiving NETBLT set data timer for a new buffer, the timer value should also take into consideration of the transfer time for all missing packets from the previous buffers.

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.

The state sequence of a send-receive buffer pair is as follows: the sending and receiving buffers are created independently. The receiving NETBLT sends a GO message, putting its buffer in a "receiving" state, and sets its control timer; the sending NETBLT receives the GO message, putting its buffer into a "sending" state. The sending NETBLT sends data until the buffer has been transmitted. If the receiving NETBLT's data timer goes off before it received the last (LDATA) packet, or it receives the LDATA packet in the buffer

and packets are missing, it sends a RESEND packet and moves the buffer into a "resending" state. Once all DATA packets in the buffer and the LDATA packet have been received, the receiving NETBLT enters its buffer into a "received" state and sends an OK packet. The sending NETBLT receives the OK packet and puts its buffer into a "sent" state.

7. PROTOCOL LAYERING STRUCTURE

NETBLT is implemented directly on top of the Internet Protocol (IP). It has been assigned a temporary protocol number of 255. This number will change as soon as the final protocol specification has been determined.

8. PACKET FORMATS

NETBLT packets are divided into three categories, each of which share a common packet header. First, there are those packets that travel only from sender to receiver; these contain the control message sequence numbers which the receiver uses for 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 QUIT, QUITACK, DONE, KEEPALIVE, and ABORT packets. Of these, all save the DONE packet can be sent by both sending and receiving NETBLTs.

Packet type numbers:

OPEN:	0
RESPONSE:	1
KEEPALIVE:	2
DONE:	3
QUIT:	4
QUITACK:	5
ABORT:	6
DATA:	7
LDATA:	8
NULL-ACK:	9
CONTROL:	10

Standard header:

```
local port:       2 bytes
foreign port:     2 bytes
checksum:         2 bytes
version number:   1 byte
packet type:      1 byte
packet length:    2 bytes
```

OPEN and RESPONSE packets:

```
connection unique ID:                    4 bytes
standard buffer size:                    4 bytes
transfer size:                           4 bytes
DATA packet data segment size:           2 bytes
burst size:                              2 bytes
burst rate:                              2 bytes
death timeout value in seconds:          2 bytes
transfer mode (1 = SEND, 0 = RECEIVE):   1 byte
maximum number of concurrent buffers:    1 byte
checksum entire DATA packet / checksum
DATA packet data only (1/0):          1 byte
client-specific data:                    arbitrary
```

DONE, QUITACK, KEEPALIVE:

standard header only

ABORT, QUIT:

```
reason:       arbitrary bytes
```

CONTROL packet format:

CONTROL packets consist of a standard NETBLT header of type CONTROL, followed by an arbitrary number of control messages with the following formats:

Control message numbers:

```
GO:             0
OK:             1
RESEND:         2
```

```
         OK message:

            message type (OK):  1 byte
            buffer number:      4 bytes
            sequence number:    2 bytes
            new burst size:     2 bytes
            new burst interval: 2 bytes

         GO message:

            message type (GO):  1 byte
            buffer number:      4 bytes
            sequence number:    2 bytes

         RESEND message:

            message type (RESEND):     1 byte
            buffer number:             4 bytes
            sequence number:           2 bytes
            number of missing packets: 2 bytes
            packet numbers...:         n * 2 bytes

      DATA, LDATA packet formats:

         buffer number:                                  4 bytes
         highest consecutive sequence number received: 2 bytes
         packet number within buffer:                    2 bytes
         data:                                           arbitrary bytes

      NULL-ACK packet format:

         highest consecutive sequence number received: 2 bytes
         acknowledged new burst size:                  2 bytes
         acknowledged new burst interval:              2 bytes
```

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 can be verified in further testing.

<2> Since the receiving end may not know the transfer size in advance, it is possible that it may have allocated buffer space and sent GO messages for buffers beyond the actual last buffer sent by the sending end. Care must be taken on the sending end's part to ignore these extra GO messages.