MARCH 1992

NOVELL® RESEARCH

An Introduction to Novell's Burst Mode Protocol

Dave Stevenson
Manager, Systems Engineering
Systems Engineering Department

Sandra Duncan
Software Engineering
NetWare Systems Group

This Application Note introduces Burst Mode, a new technology that employs enhancements to the NetWare Core Protocol. Burst Mode, which is built on top of IPX, was developed explicitly to improve NetWare's performance when transferring large files over wide area networks. The AppNote defines how the Burst Mode protocol works and reports on the results of some real-world performance testing. It also includes technical information on Burst Mode packet structures.

Contents

Acknowledgements

This Application Note would not exist without the hard work and technical expertise of many people both inside and outside of Novell. Thanks to Therron Powell, who was instrumental in collecting and representing performance testing data and results. Thanks also to Willie Tejada, product line manager, counselor, confidant, and collaborator, who also authored several of the original figures we modified for this AppNote.

Overview

The Burst Mode protocol, formerly called Packet Burst, is a new NetWare Core Protocol (NCP) technology for expediting large file reads and writes over wide area network links. It provides a sliding window-like dialogue for IPX/NCP, and features an adaptive, self-tuning flow control mechanism. Depending on your network and numerous other factors, you can expect to see a performance improvement ranging anywhere between 10% and 300% with Burst Mode.

This AppNote introduces the Burst Mode protocol and gives an overview of the theory behind it. It defines the implementation of Burst Mode, the workstation memory requirements and configuration parameters, and the transmission rate control algorithm. It also discusses the performance implications and reports on some performance tests conducted at various Novell customer sites. For those interested in the technical details at the packet level, we have also included a discussion of the packet structure used for Burst Mode communication.

Traditional NCP Interaction Using IPX

It is a common industry misconception that IPX is a half duplex request/response protocol. Actually, IPX has always been a full duplex protocol. The request/response nature of standard NetWare line traffic is introduced by NCP at the upper layers of the protocol stack. Figure 1 illustrates the typical one-to-one request and response relationship imposed by NCP.

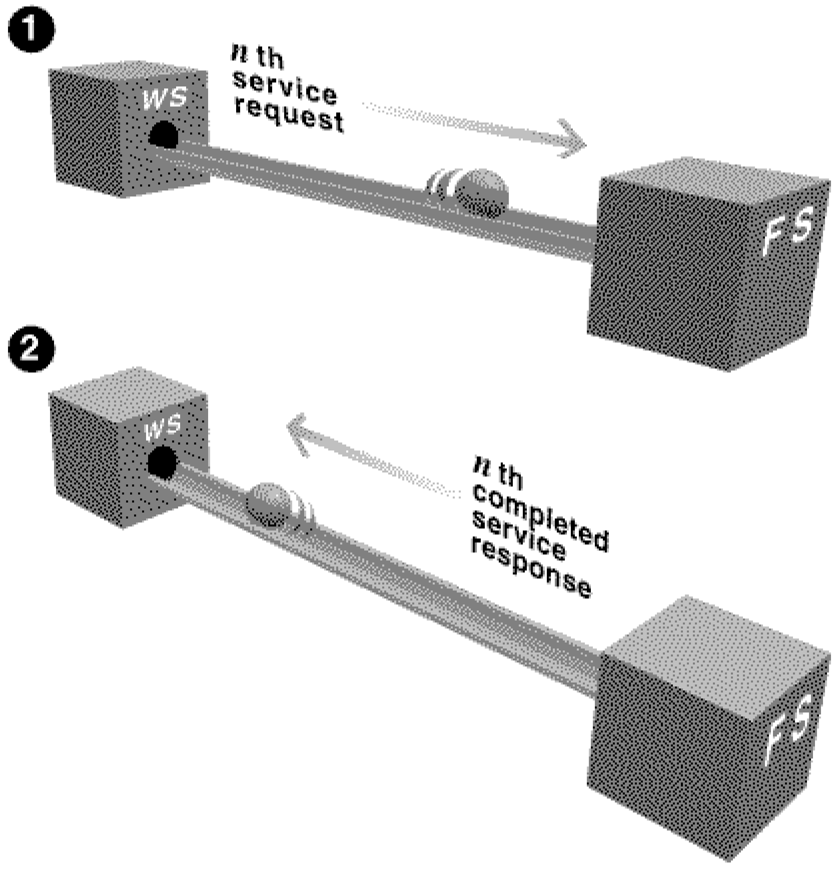


Figure 1: In a typical NCP interaction, each client request is answered with a response acknowledgment from the server.

Over wide area network (WAN) links, this one-to-one NCP request/response relationship can result in performance degradation due to the ping pong effect. As each side waits for a corresponding acknowledgment, there is a significant amount of time when no traffic is being transmitted on the line.

The delay is compounded when large files are being transmitted a packet at a time. In Figure 2, a client wants to read a 64KB file located on a remote server that is accessible through a router and across a WAN link. Since the maximum packet size for NetWare routers is 512 bytes, the client and server must exchange 128 of these 512-byte packets to fulfill the read request.

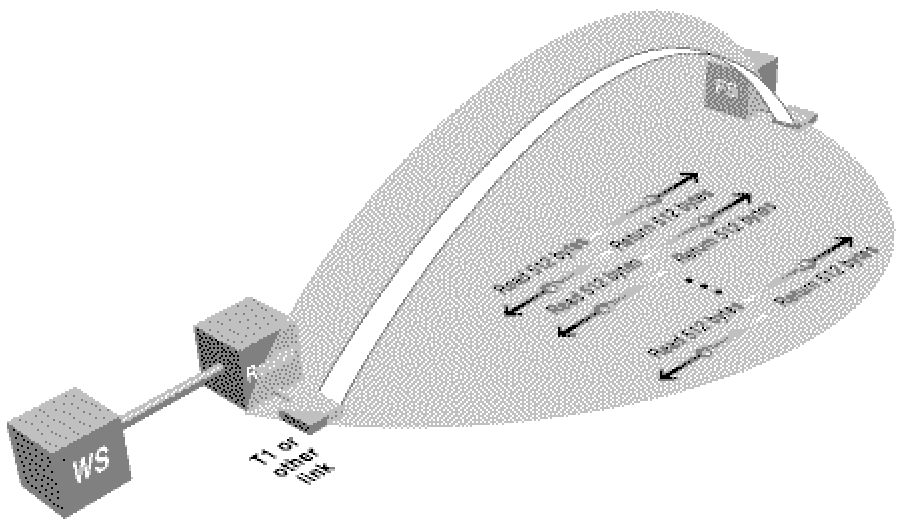


Figure 2: Over WAN links, the request/response nature of NCP interaction can cause performance degradation.

The Burst Mode Protocol

NetWare's Burst Mode protocol allows a client to issue a single read or write request for blocks of data up to 64KB in size. The data is partitioned into appropriate packets, and these packets are transmitted back-to-back on the line, as shown in Figure 3. No intervening involvement by the sender or reply packets are required until after the last of the burst packets is received.

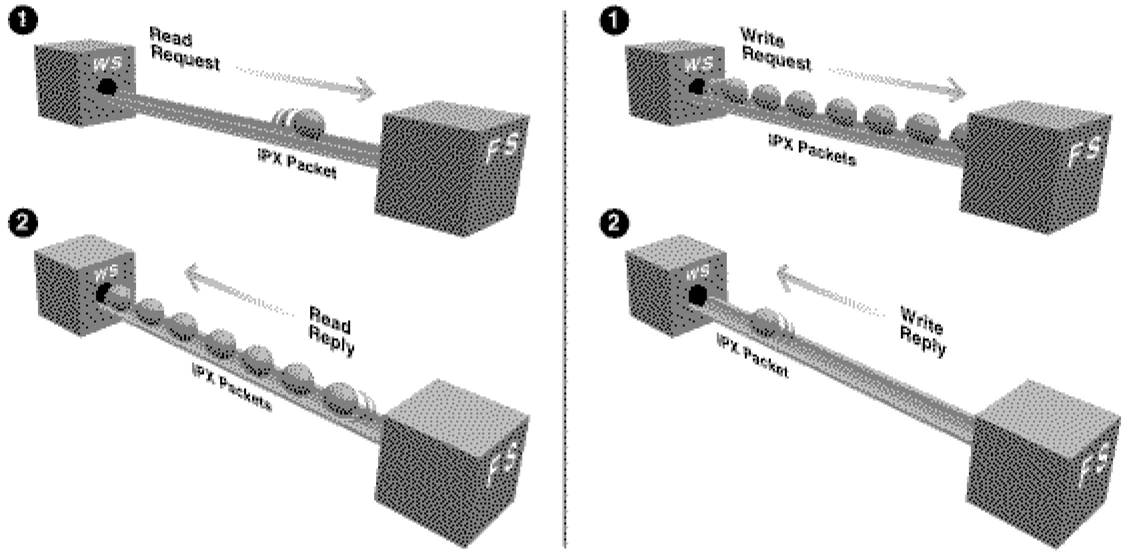


Figure 3: With Burst Mode, clients can have the server read or write large blocks of data comprising multiple packets.

The current implementation of Burst Mode requires:

- The Burst Mode NLM (PBURST.NLM) to be loaded on the file server
- The Burst Mode shell (BNETX.COM) to be loaded on the workstation
- Adequate space in conventional memory at the workstation

The amount of workstation memory needed is based on parameters specified by the user in the NET.CFG file and the capability of the network hardware media.

The Burst Mode shell itself requires more conventional memory than traditional NetWare shellspabout 4-5KB minimum, plus memory for buffers. The total amount of client memory consumed depends on (1) the number of Burst Mode buffers configured for the workstation and (2) the packet size negotiated with the file server.

See the Client Memory Requirements section later in this AppNote for more explanation of the memory requirements for Burst Mode clients.

Flow Control Mechanism

To control the flow of data, Burst Mode employs a dual volume-throttling mechanism that uses (1) an expanding and contracting window size algorithm and (2) a packet transmission metering value.

Window size refers to the number of frames or packets contained in a single burst. This is one area in which Burst Mode's window technology deviates from traditional sliding window protocols such as SDLC. In standard windowing protocols, the window size is typically a fixed value; that is, a fixed maximum number of packets (usually seven) can be transmitted before a responding acknowledge packet must be received.

With Novell's Burst Mode, the number of packets in the window is variable, up to a theoretical maximum value of 128 (64KB divided by 512-byte packets).

Burst Mode also uses a metering throttle technique in which the client requests a certain time delay before individual packets are placed back-to-back on the transmission media. This gap time is used to prevent fast servers from overrunning the client's buffers.

The actual values for these two parameters are determined individually by each client on the network. To arrive at appropriate values, the client shell executes a complex transmission rate control algorithm.

This algorithm also allows these parameters to be dynamically changeable during processing. The Burst Mode shell continually monitors line quality (bad packets) and line capacity (missed packets) and renegotiates the parameters accordingly, without the need for user intervention. When network traffic is heavy enough that packets are dropped in transmission, the Burst Mode shell decreases the window size to minimize packet loss. Under less demanding network conditions, the shell increases the window size to maximize performance. (See "How the Algorithm Works" later in this AppNote for more details.)

Bad Packet Retransmission

Another deviation from standard windowing protocols is Burst Mode's ability to request retransmission of only those packets that are missing. In standard windowing technologies, every packet after and including the bad packet must be retransmitted. The ability to request only missing data provides a significant performance benefit on poorer quality lines.

Burst Mode Initialization

The Burst Mode initialization process occurs when the client attaches to the server. (The exact procedure is explained under Setting Up a Burst Mode Connection later in this AppNote.)

If a single client is concurrently attached to multiple servers, it will use the Burst Mode NCP extensions with Burst Mode-enabled servers and use the standard NCPs with non-Burst Mode servers. This mixed server environment is illustrated in Figure 4.

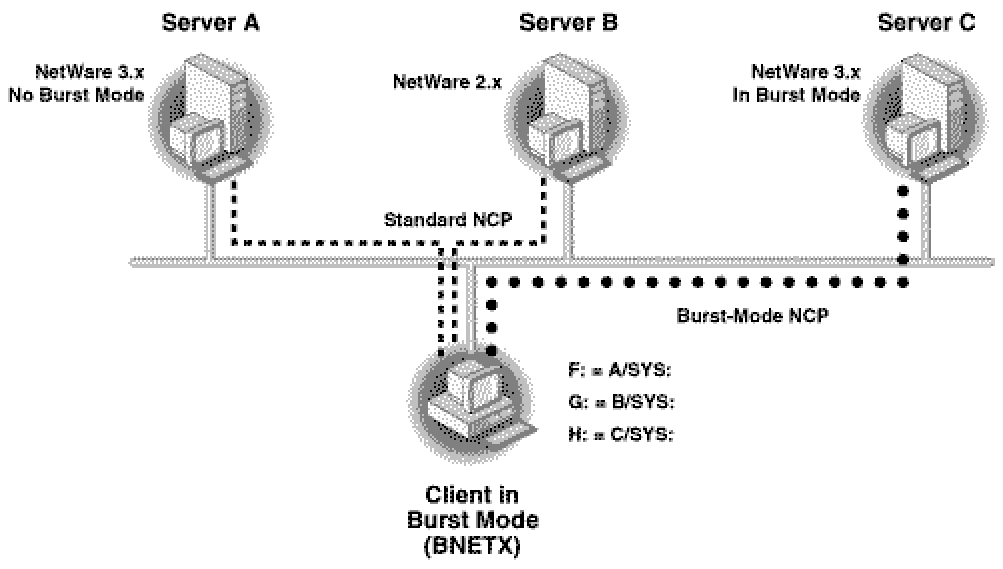


Figure 4: In a mixed server environment, a client will use Burst Mode only with Burst Mode-enabled servers.

Conversely, Burst Mode-enabled servers will use Burst Mode NCPs with enabled clients and perform traditional non-Burst Mode NCP interaction with standard workstation shells. This environment is illustrated in Figure 5.

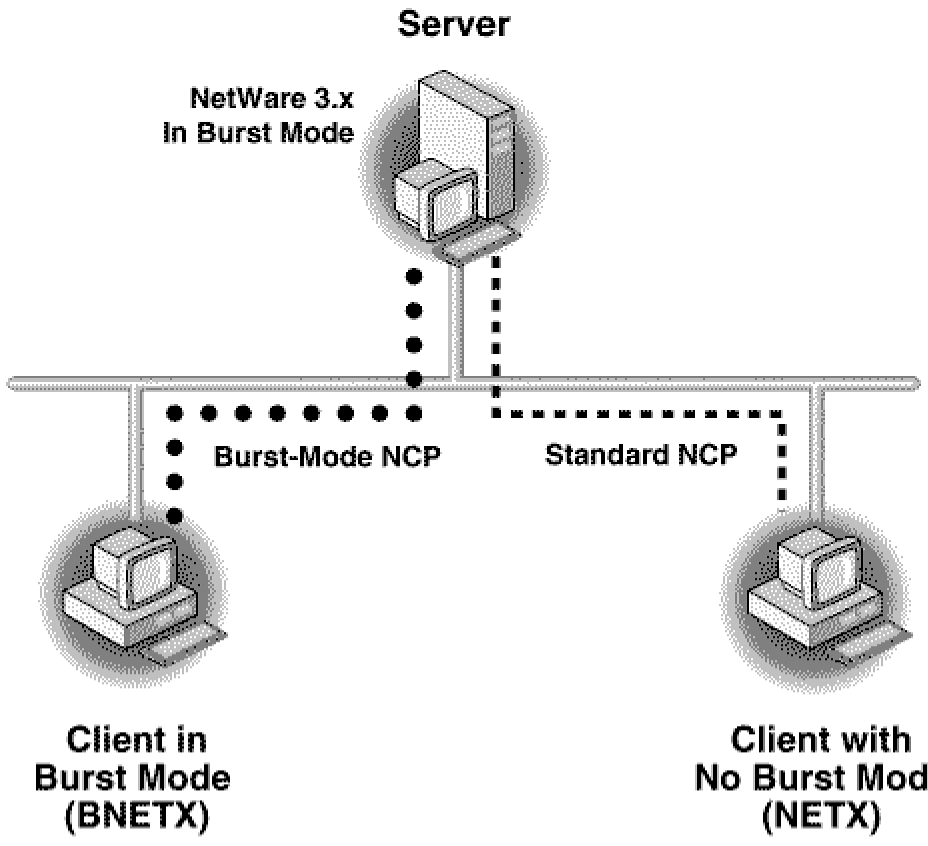


Figure 5: In a mixed client environment, a Burst Mode-enabled server replies with whatever protocol the client uses.

Burst Mode Theory

Burst Mode is a special-purpose service protocol for read and write NCPs. It is built on top of IPX; as such, it is similar to SPX in that it is a connection-oriented protocol. As a special-purpose service, it has been optimized to avoid the overhead of acknowledging the delivery of each packet, as the SPX protocol must. To fully explain the theory behind this protocol, we'll first define a few terms.

Burst Mode communicates using multipacket units called bursts. The packets that make up the burst are called fragments. A burst includes:

- The IPX headers for each packet
- The Burst Mode headers for each packet
- The request or reply, with or without data

Theoretically, one burst may be up to 64KB in length. Figure 6 illustrates fragments and bursts.

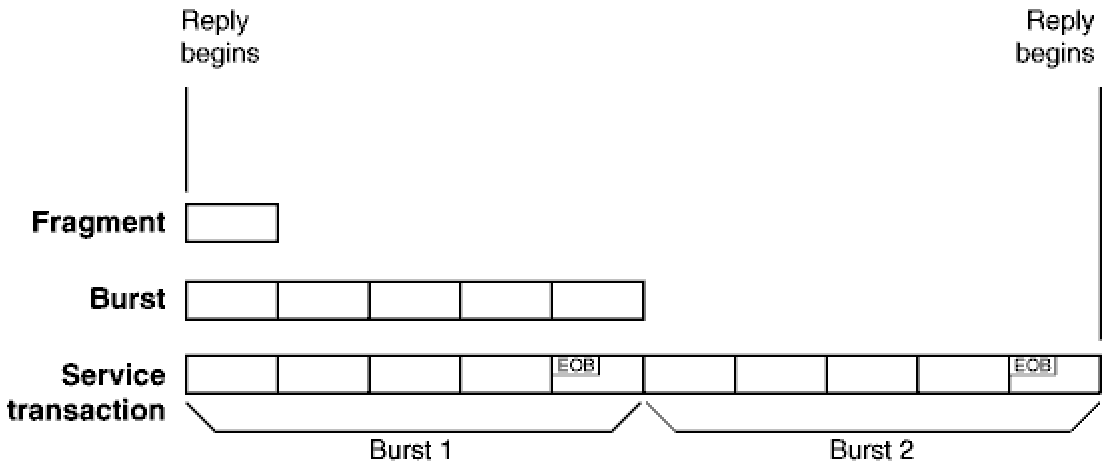


Figure 6: A burst is made up of fragments. A service transaction may require one or more bursts to complete.

A burst transaction occurs from the beginning of the burst request to the end of the reply. For example, the burst transaction begins when a client sends a request to read a file. The file server replies by sending a series of packets (a burst) to the workstation, indicating the last packet with an End Of Burst flag.

When the client receives that last packet, the burst transaction is complete, as long as there are no missing fragments. If fragments are missing, the client sends a missing fragment list to the server, which then retransmits only the missing fragments. This process repeats until no missing fragments remain or until a timeout expires.

A complete file read or write may require several burst transactions. A service transaction (see Figure 6) is the completion of the entire request, whether that entails just one burst and reply or many bursts and the associated replies. In other words, if the client requests to read a large file and the request requires many burst transaction transmittals, the completion of all transmittals is a service transaction.

Note that there is one End of Burst flag per burst, not per service transaction.

Write and Read Requests

The protocol differs slightly for write and read requests. Burst Mode is a client/server protocol, so the client initiates both write and read requests.

Write Requests. On write requests, the client does not wait for the server to acknowledge the request to write. The client simply sends the data with the write request, along with information about the starting offset and the number of bytes being written. The server replies to the client with an acknowledgement of success, or with a missing fragment list. The server requests data only if it is missing.

Read Requests. On read requests, the client must wait, by default, for the data. The server's reply to the request consists of the requested data. Therefore, the client must determine whether the read was successful or not. If the client received all the requested bytes, it makes its next request, without an acknowledgement. But if there are missing fragments, the client sends a system packet, which includes a list of missing fragments, back to the server.

System Information

In addition to exchanging data, there are situations when the server and client need to exchange information about the data and its transmission. This is called system information. It gives additional information about the communication taking place, and is contained in a specific field in the burst header structure. For example, the End Of Burst marker is a bit in the system information field.

The protocol's packet structure allows system information, acknowledgement information, and burst data all to reside in one packet. (See Appendix A for a description of the Burst Mode packet structure.)

Setting Up a Burst Mode Connection

As mentioned in the overview, a workstation sets up a Burst Mode connection with a file server at attach/login time. Once the Burst Mode connection is established, it stays up as long as PBURST.NLM remains loaded at the server and the Burst Mode shell is loaded at the workstation.

If the shell fails to make a Burst Mode connection during attach/login, it uses normal NCPs to do the work. This ensures compatibility with servers not running the Burst Mode NLM. Also, since Burst Mode is established individually for each connection, it is possible for a client to burst with one server while not bursting with another.

Upon loading, the shell first determines whether the workstation has enough memory for the requested number of Burst Mode buffers. (See How the Algorithm Works for a discussion of memory requirements and the implications of workstation speed and performance.) If there is enough memory, the client proceeds to initiate a Burst Mode connection with the server.

During this connection process, the shell negotiates maximum burst sizes, in the same fashion as it negotiates packet size, with each server connected. The size of each fragment (packet in a burst) is the same as the packet size negotiated for non-burst transmissions.

If the workstation succeeds in setting up a Burst Mode connection with the server at initialization, Burst Mode is not actually invoked until a big read or write request is made.

Once a Burst Mode connection is established between a workstation and a given file server, the NetWare workstation shell automatically uses the Burst Mode service whenever an application makes a read or write involving more than 512 bytes of data. In other words, applications do not have to be Burst Mode aware.

The Transmission Rate Control Algorithm

Various factors influence the transmission rate of data on the network, including:

- The speed of the file server
- The speed and available memory of the workstations

- The speed and buffer size of the network media
- The traffic on the network at transmission time

A protocol such as Burst Mode must be sensitive to data-transmission factors as well as to issues of interaction between host network elements. For example, a fast file server could overwhelm a congested bridge or slow workstation with a large stream of packets, resulting in dropped packets. When the network becomes congested, a bursting workstation must be able to give up a portion of the bandwidth it is using and readjust to maximum bandwidth usage when it is available.

Inasmuch as the client requests services from the server, the Burst Mode flow control is governed completely by the client. There are two basic considerations which relate directly to performance:

- How should the protocol regulate the relationship between a fast file server and a slow workstation?
- How should the protocol use the maximum amount of bandwidth available and yet not dominate the wire?

The relationship between a fast file server and a slow workstation is regulated by determining a minimum time between packet transmissions. The bandwidth issue is regulated by an expanding and contracting burst window size. The client determines both pieces of information.

Flow control also requires regulating the current maximum burst size. For example, the hardware may be capable of transmitting a 4KB burst, but traffic on the network might indicate that transmissions should be limited to 1KB bursts so as to not dominate the bandwidth.

To maximize the fair and effective transmission rate in light of these factors, the Burst Mode protocol uses a twofold transmission rate control algorithm. The algorithm has two basic control elements:

- **Burst gap time**, which governs the amount of time between packet deliveries to the wire.
- **Burst window size**, the amount of data that will be sent in the current burst.

How the Algorithm Works

Burst gap time is determined by taking the median of the time between packet arrivals. (Gap time is not meaningful when bursts must cross a bridge.)

The minimum window size is based on the maximum physical packet size negotiated for the network media and the number of Burst Mode (PB) buffers configured for the workstation. As currently implemented, the burst window size cannot fall below this number.

The maximum burst window size is based on current network traffic congestion conditions. Window size is regulated by successes and failures; a timeout is considered a failure, as is a dropped packet.

Note: The window size fluctuation is based on research conducted by Van Jacobson at UC/Berkeley.

The gap time is established early on and stays relatively stable once established. The data window size and the timeout values fluctuate much more when the network is under a load.

The following sections describe the logic of the transmission rate control algorithm in more detail.

Setting the Number of Burst Mode Buffers

Packet burst is enabled at the workstation by adding the following line to the workstation's NET.CFG file:

```
PB BUFFERS = n
```

In this line, the variable n can be any number between 0 and 10. Setting n = 0 disables Burst Mode. Setting n = 1 causes the shell to increase the value up to the minimum of 2. If n > 10, the shell simply reduces the number of buffers to 10, rather than returning an error to the user. (Applications can read the actual PB BUFFER value with a new API. Refer to Novell's API documentation for more details.)

If the workstation doesn't have enough memory for the requested number of buffers, the shell reduces the number of buffers accordingly, without notifying the user.

Increasing the number of Burst Mode buffers does not necessarily result in better performance. For example, a slow machine cannot move data from the hardware buffers to application memory fast enough, regardless of whether you increase the value for n. Also, if a network card has hardware receive buffers, the PB BUFFERS parameter could be set to a higher number than for cards without hardware buffers. In both scenarios, performance will actually degrade with the higher values.

Perhaps the safest approach is to set the number of buffers low (for example, PB BUFFERS = 2). This allows Burst Mode's flow control mechanism to adaptively use more bandwidth if it is available, without running the risk of overwhelming slow hardware.

Client Memory Requirements

In the current implementation, Burst Mode requires conventional memory space in the workstation. The amount of conventional memory, m, in bytes, required for one packet burst buffer is:

m = negotiated packet size + 102

The negotiated packet size value is the size negotiated between the NetWare client and file server. With Ethernet, for example, the negotiated packet size might be 1,024 bytes (1KB). The constant 102 is the total number of bytes needed for the IPX, NCP burst, and ECB headers.

Thus the total memory M required for all packet burst buffers, in bytes, is:

M = pb buffers x m

The value pb buffers is the number of Burst Mode buffers specified by the user in the NET.CFG file, and m is the memory required for one packet burst buffer, as calculated above.

When the client receives a burst, the data is transferred from the hardware to the workstation memory. Therefore, to maximize performance, one must consider the speed of the workstation as well as the network card being used. Performance optimization, however, will be largely a matter of experimentation.

Determining Minimum and Maximum Window Size

After figuring the maximum physical packet size and determining that there is enough memory available at the workstation for the number of Burst Mode buffers requested, the Burst Mode protocol determines a minimum and maximum size for the Burst Mode data window.

Note: All adjustments to these parameters are made on a burst-by-burst, as opposed to a packet-by-packet, basis.

As stated previously, the minimum window size is based on the maximum physical packet size negotiated for the network media and the number of Burst Mode buffers configured for the workstation. The current implementation of Burst Mode won't let the burst window size fall below this number. The Burst Mode shell allows that minimum number of packets to be transmitted regardless of network conditions. This is another reason why it may be wiser to set the number of Burst Mode buffers at 2.

The theoretical maximum window size is 64KB. Control of the current maximum allowed window size is determined by the occurrence of successful and unsuccessful burst transactions. (A successful burst transaction is one with no dropped packets; an unsuccessful burst transaction has dropped or missing packets.)

- A successful burst transaction causes the window to increase in size by 100 bytes.
- On the other hand, a burst with one or more dropped packets causes the window to decrease to 7/8 of its current size.

The window size value will not drop below the physical packet size negotiated by NetWare and the network media (such as Token Ring or Ethernet) multiplied by the number of Burst Mode buffers. In equation form:

Window Size ò <u>negotiated packet size</u> x <u>pb buffers</u>

A slow workstation will determine a maximum window size based on whether it ran out of Burst Mode receive buffers during the burst transaction.

Network Timeouts

In addition to data window size and gap time, network timeouts (how long a workstation waits before assuming that communication between the file server and the workstation has been temporarily interrupted or severed) are factored in to the rate control of Burst Mode. No response times less than two 55ms ticks are factored in.

Longer delays on the network (up to the maximum timeout set by the IPX initialization code) are processed according to a formula for calculating moving averages and moving deviations modeled by Van Jacobson. That figure is used to indicate whether the timeouts should be extended to some degree.

Performance Implications

Due to the large number of variables involved (line speed/quality, client capacity, server capacity, number of hops, number of buffers, and so on), the exact performance improvement you can expect to achieve with Burst Mode is difficult to predict. For the purposes of this AppNote, let's define performance improvement as the difference between pre-Burst Mode and post-Burst Mode response times for the completion of a single task.

Intuitively, we would expect to see Burst Mode performance increase in proportion to how much of a request/response bottleneck exists on the communication line. In other words, it would seem that the greatest benefit with Burst Mode should occur over slow lines (9600 bps and slower).

However, the actual benefit of using Burst Mode over 9600 bps (or slower) lines is not as great as we might suppose. Consider the following rationale.

Assume you are transmitting 10-bit characters (8 bits + start and stop bits) over a 9600 baud line. In this scenario, the number of seconds it takes to transmit one character is:

1/9600 x 10 = .00104 seconds per character

Further, assume that the worst-case transmission delay for AT&T land-based telephone lines is at most 1/40 (0.025) of a second. (This accuracy is guaranteed by the United States Naval Observatory master clock in Washington, D.C.)

With this much delay, the first character placed on the line will reach the other end of the line at about the same time that the 25th character is being placed on the line:

.025 sec ö .00104 chars/sec = 24.03 characters

Since data packets are usually larger than 25 bytes in size, not even one entire packet (let alone a series of full packets) can be placed on a 9600-baud line all at once. Of course, with Burst Mode, you eliminate the need for the shorter, non-data protocol control packets for which the sending side would normally have to wait for responses to come back from the receiver. This alone results in a slight (5þ15%) increase in performance.

More dramatic performance increases occur when more than one packet can be placed on the line at the same time, because that allows several packets in a burst to be in transit together. This is the case in two types of situations:

- When communicating over high-speed transmission lines (such as T1)
- When there are significant transmission delays between sender and receiver

The higher data transmission speeds typical of T1 lines makes it possible for multiple packets to be placed on the physical line at the same time. Combining T1 links with Burst Mode technology results in a performance increase of around 100%.

The next two figures illustrate network configurations in which transmission delays are significant enough to allow more than one packet to be in transit at the same time. The first configuration, shown in Figure 7, is when the data must traverse multiple routers or bridges.

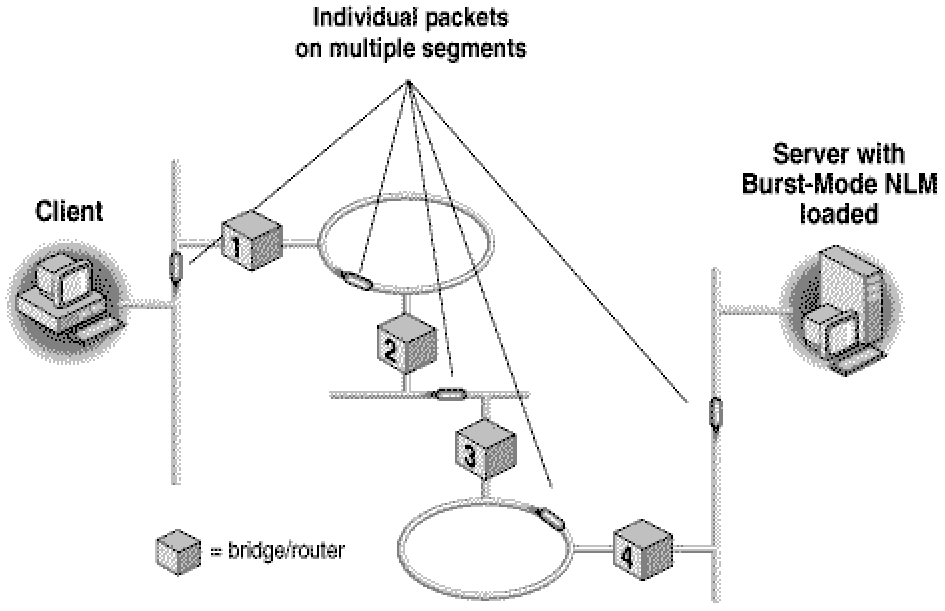

Individual packets on multiple segments
Client
Server with Burst-Mode NLM loaded
1
2
3
4
= bridge/router

Figure 7: Multiple hops through routers or bridges introduce enough delay to allow multiple packets on the line at once.

In this configuration, individual packets can occur on each separate network segment, as well as in the routing/bridging devices.

The second configuration is when communication occurs over X.25 packet-switched lines, as shown in Figure 8. Due to the nature of packet-switching networks, they also allow many packets in a burst to be on the end-to-end connection at once.

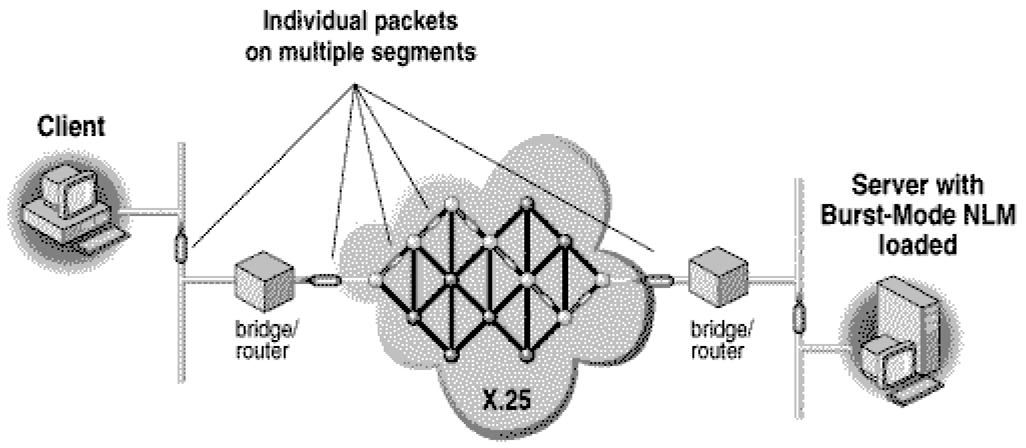

Individual packets on multiple segments
Client
Server with Burst-Mode NLM loaded
bridge/ router
bridge/ router
X.25

Figure 8: Communication over packet-switching networks is typically slow because of the multiple paths data can take.

For the multiple-hop configurations, the performance increase can be up to 300%. The greatest performance increase (near 400%) occurs when you combine these configurations with a T1 link.

In our testing, we also noted significant performance increases (up to 50%) in local Ethernet and Token Ring environments, even though both protocols (CSMA/CD and Token Passing) support only one packet on the media at a time. This performance boost is attributed to the fact that with Burst Mode, the packets in a burst are transmitted as a unit. The sender doesn't have to wait for an acknowledgment to be received for each packet, as is the case in non-Burst Mode interaction.

It is important to remember that Burst Mode is invoked only with read and write requests for large chunks of data (greater than 512 bytes). If an application on a Burst Mode-enabled system performs 80-byte data reads and writes, the Burst Mode NCP calls will not be used. In this situation, the application layer of the protocol stack implements a half duplex request/response data transfer as explained in the overview.

Performance Testing

The throughput test we used to measure Burst Mode performance is straightforward and representative of those environments that stand to benefit most from Burst Mode. The testing was performed via a standard DOS batch file named TEST.BAT (listed in Appendix B). TEST.BAT performs five sequential DOS copies from a server to a client of files that vary in size from 100 bytes to 1,000,000 bytes (1MB). Before and after each copy, the batch procedure takes time stamps and saves them to a log file named OUT.LOG.

To minimize the impact of potential disk I/O bottlenecks, the batch procedure transfers files to the NULL device on the client. It then discards the first time stamp taken for each sequence to minimize the potential effect of server disk I/O during the initial file read.

We calculated the percentage of performance increase as follows:

Percent increase in performance = (T1 - T2) x 100
ÄÄÄÄÄÄÄÄÄ
T2

where:

End Time Start Time
T1 = (1MB file - (1MB file
non-Burst Mode) non-Burst Mode

End Time Start Time
T2 = (1MB file - (1MB file
with Burst Mode) with Burst Mode

To obtain real-world performance numbers, we conducted the bulk of the performance tests at Novell customer sites. This allowed us to test Burst Mode on complicated topologies (multiple third-party bridges/routers, satellite links, and so on) that were not available in a laboratory environment. Many of these tests were conducted on actual production networks.

Although data results will invariably be affected by ancillary network traffic, our original intent was to test real-world environments. Over the long run, the impact of other network traffic on the non-Burst Mode tests should even out with the Burst Mode results.

Performance Test Results

As expected, smaller file transfers showed virtually no increase in performance with Burst Mode (see Figure 9). Therefore, the remaining figures report only the results for the 1MB file transfer to highlight performance differences. Figure 10 shows the varying performance increases for 14 test sites. Figure 11 highlights the performance differences between 5 and 10 Burst Mode receive buffers in the workstation.

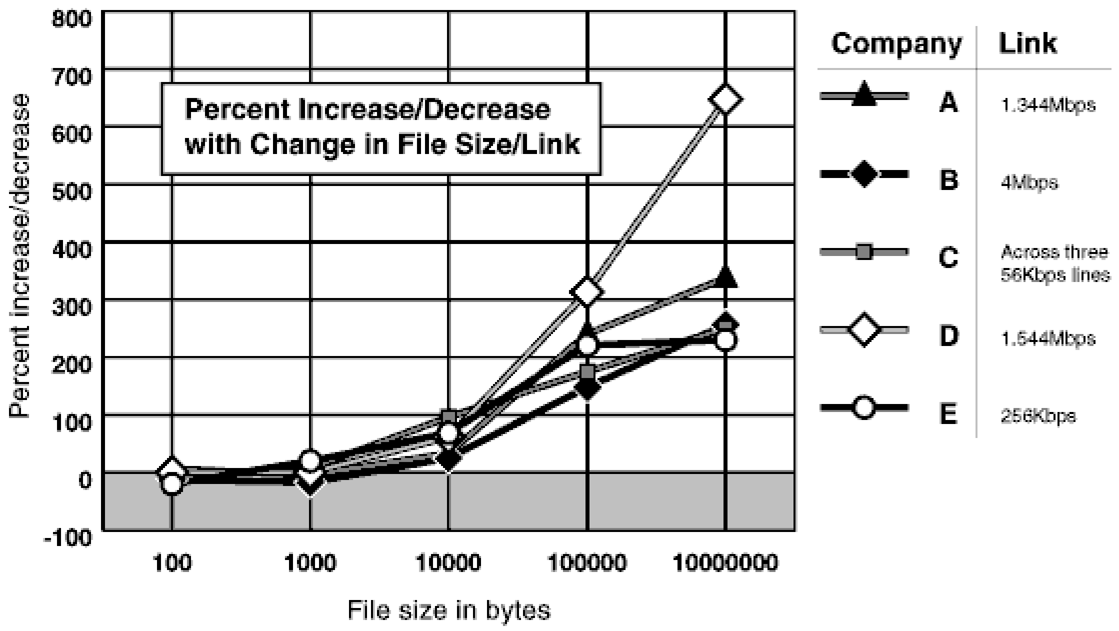


Figure 9: Burst Mode performance results for five different test sites with various link types.

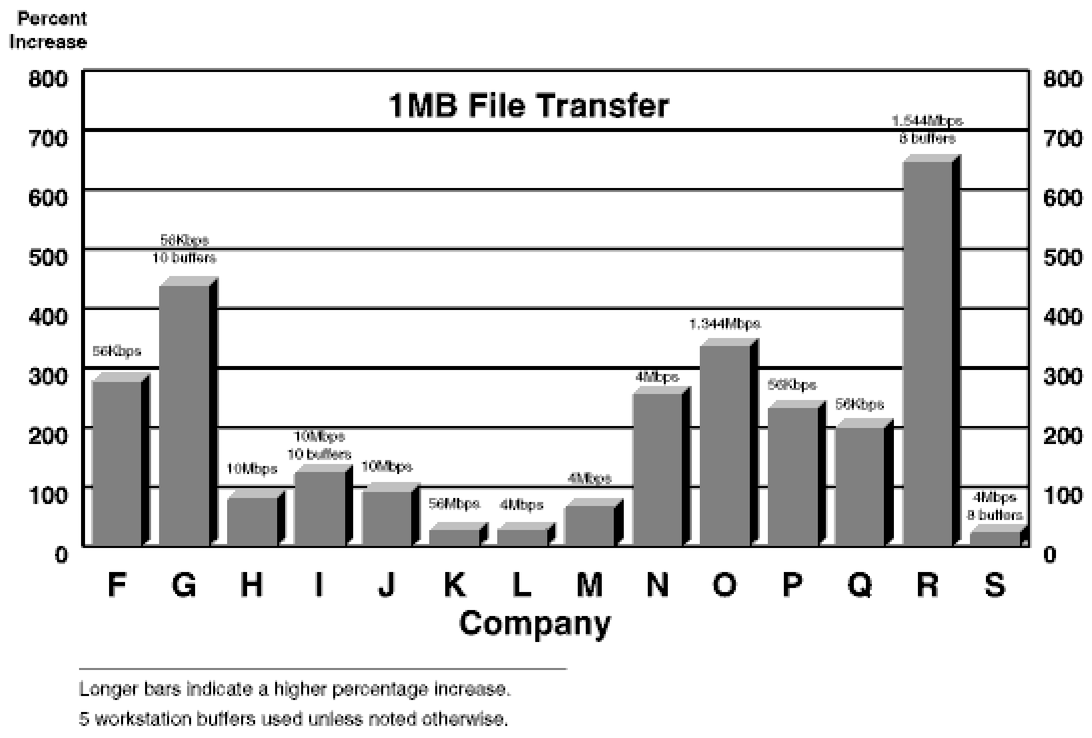
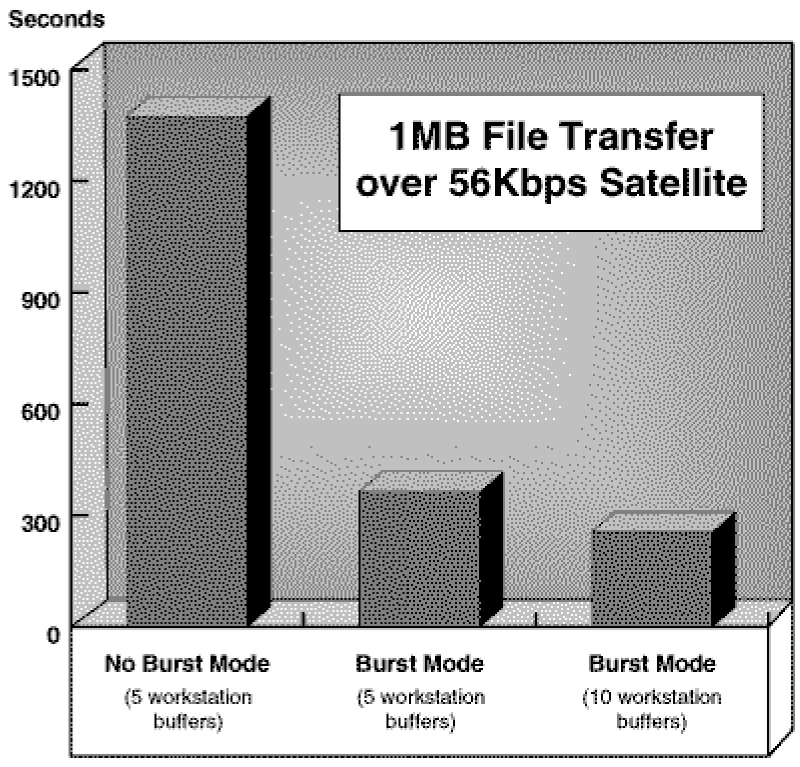


Figure 10: Burst Mode performance results for other test sites.



Shorter bars indicate better performance

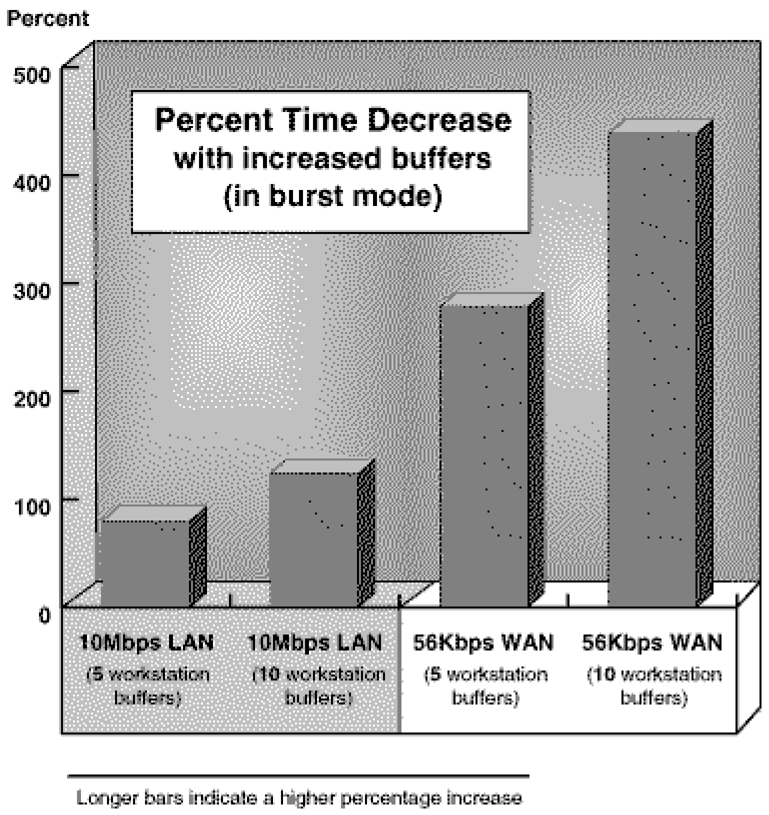


Figure 11: These two graphs show performance results with 5 and 10 Burst Mode receive buffers.

Appendix A: Technical Information

This appendix contains a technical description of the packet structure and connection request structures used with the Burst Mode protocol. It is mainly of interest to those interested in doing protocol analyzer traces of Burst Mode communications.

Burst Packet Structure

As seen on the network, a burst packet has the major fields shown in Figure A-1.

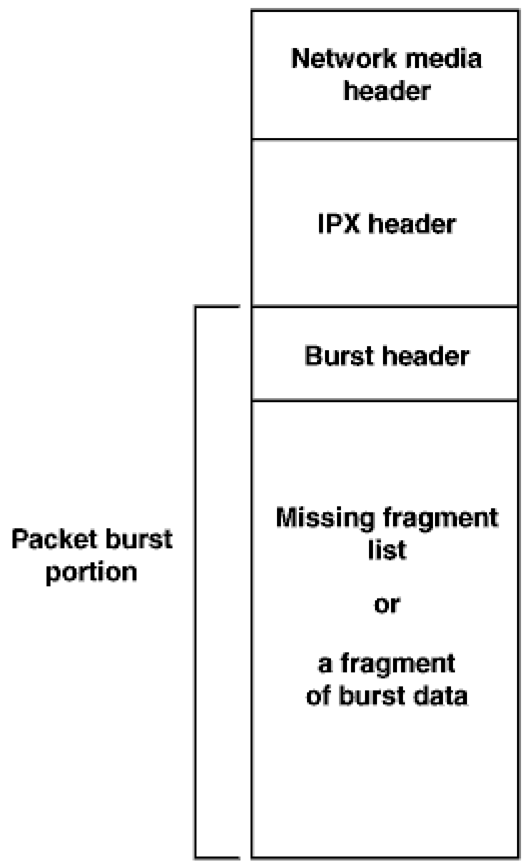
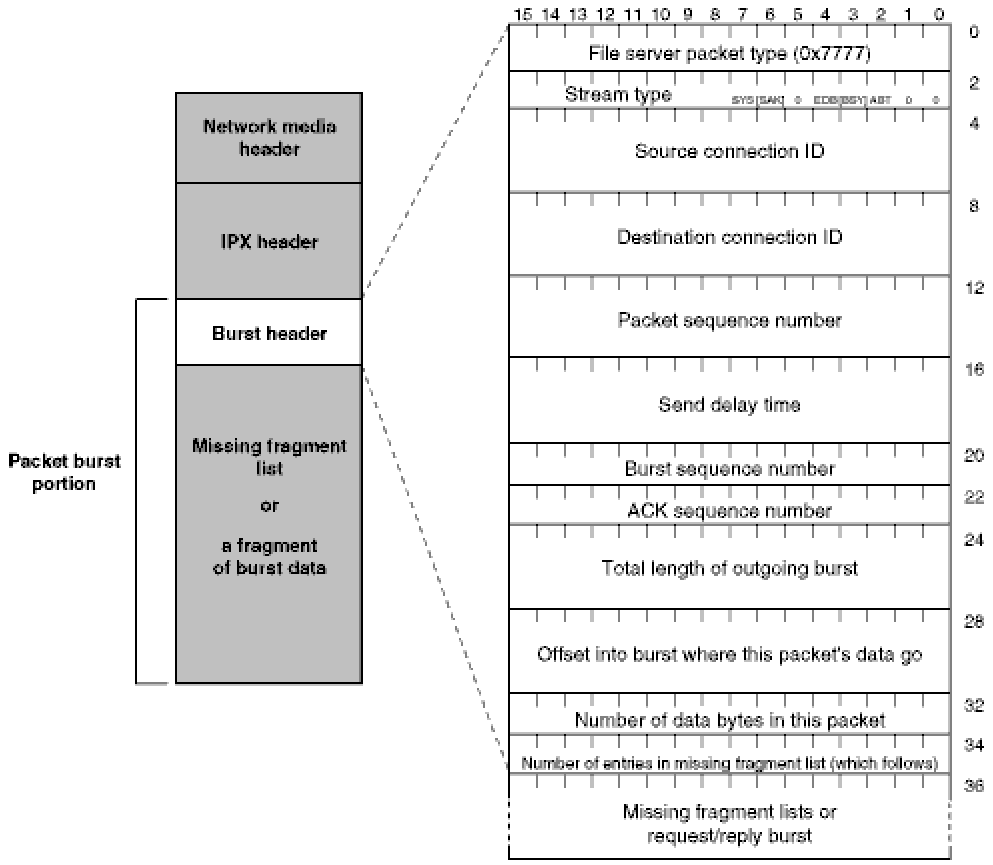


Figure A-1: On the network, a packet that includes Burst Mode information looks like this.

The following sections describe the Burst Mode portion of the packet in more detail.

Burst Header Structure

The header structure of the Burst Mode packet includes the fields shown in Figure A-2.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Network media header
IPX header
Burst header
Missing fragment list or a fragment of burst data
Packet burst portion
File server packet type (0x7777)
Stream type
SYS SAK 0 EOB BSY ABT 0 0
Source connection ID
Destination connection ID
Packet sequence number
Send delay time
Burst sequence number
ACK sequence number
Total length of outgoing burst
Offset into burst where this packet's data go
Number of data bytes in this packet
Number of entries in missing fragment list (which follows)
Missing fragment lists or request/reply burst
0
2
4
8
12
16
20
22
24
28
32
34
36

Figure A-2: The header of the Burst Mode portion of a packet.

Here are definitions of the fields in the Burst Header portion of the packet.

File Server Packet Type. This is the primary file server packet type code. For Burst Mode packets, this type code is 0x7777. The recognized File Server Packet Type codes are:

0x1111	Allocate Slot Request
0x2222	File Server Request (that is, an NCP request)
0x3333	File Server Reply (an NCP reply)
0x5555	Deallocate Slot Request
0x7777	Burst Mode Protocol Packet (big NCP request/reply)
0x9999	Positive Acknowledge

Communication Control Bits. The next six fields are communication control bits that are manipulated by the Burst Mode protocol directly, without any application or user interaction. Note that the SAK and BSY fields shown in Figure A-2 are currently reserved.

Source Connection ID. This is a random number that uniquely identifies a big connection from the sender's point of view. This number is typically formed from the current time of day. The packet burst protocol does not allow a connection ID of zero.

Since burst packets could duplicate and circulate in bridges for quite a while, especially on wide area networks, connection IDs help identify packets that are no longer relevant. Hi-lo or lo-hi ordering does not pertain to this value.

Destination Connection ID. A random, nonzero number that uniquely identifies a big connection from the receiver's point of view. See Source Connection ID above.

Packet Sequence Number. This number is incremented by one for each new packet a node transmits per service transaction. In the packet, it is kept in hi-lo format.

Send Delay Time. This field contains the amount of time delay the transmitter is using between packet transmissions. The delay is specified in units of approximately 100 microseconds. In theory, this field is for information only. This value is in hi-lo format.

Burst Sequence Number. This is the burst number currently being transmitted. In setting up a packet burst connection, two nodes begin by zeroing their burst sequence numbers. The first burst each sends is labeled zero; the nodes increment their burst sequence numbers with each successive burst they send. This field contains the current burst sequence number in hi-lo format.

ACK Sequence Number. This field indicates the burst sequence number that a node expects to receive next. When a node receives a packet, it can use this field to tell whether the last burst it transmitted arrived successfully at the destination. This value is in hi-lo format.

Total Length of Outgoing Burst. This field specifies the total length in bytes of the burst being transmitted. It is presented in hi-lo format. One use for this field is for the receiver to establish a missing fragment list as soon as it receives the first packet of a burst.

Offset Into Burst Where This Packet's Data Go. This field is self-explanatory. The value is in hi-lo format.

Number of Burst Data Bytes In This Packet. This field is also self-explanatory. It is in hi-lo format.

Number of Entries in Missing Fragment List. This field specifies how many elements (not bytes) are in the missing fragment list. The missing fragment list, if any, follows this field. Zero is a valid number and indicates that there are no missing fragments. This field is presented in hi-lo format. The system bit should be set if this field is filled in.

Missing Fragment List

The missing fragment list describes the parts of a burst that the local node is still expecting to receive from the remote node. Initially, there is basically one fragment that spans the whole burst. As parts of the burst arrive, the size of this fragment is reduced.

During reception, holes may form due to lost packets or packets arriving out of sequence. When this happens, there will be more than one fragment in the missing fragment list. The number of elements in the missing fragment list is given in the header structure. (See the Burst Header Structure section.)

Figure A-3 shows the structure of one missing fragment list element.

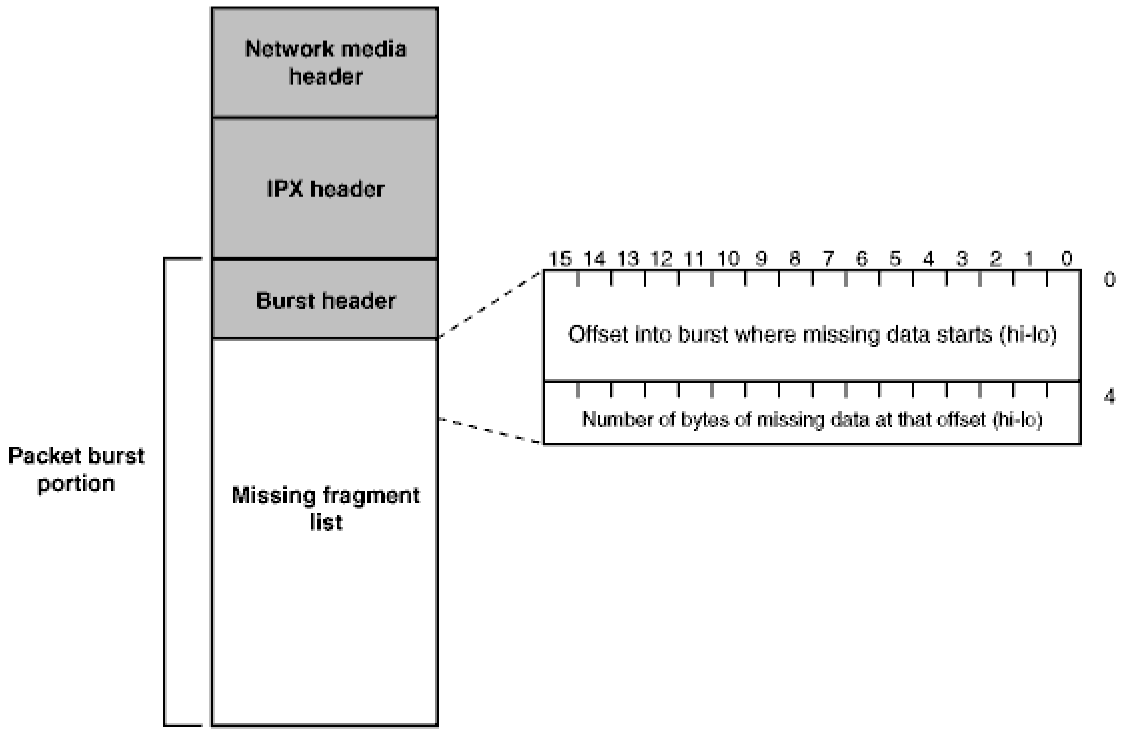


Figure A-3: An element of the missing fragments list.

Request Data

The Request Data part of a burst packet contains the part of the burst specified by the header structure. Bursts, taken as a whole, have their own structure. The Request and Reply Structures section below includes more detail.

Request and Reply Structures

As shown in the Burst Header Structure section, the BIG_SEND_BURST Stream Type in the Burst Mode header indicates support for big NCP requests and replies. The following sections describe the request and reply burst structures.

Big File Read Request

Figure A-4 shows the request burst portion of the packet burst packet for a big file read.

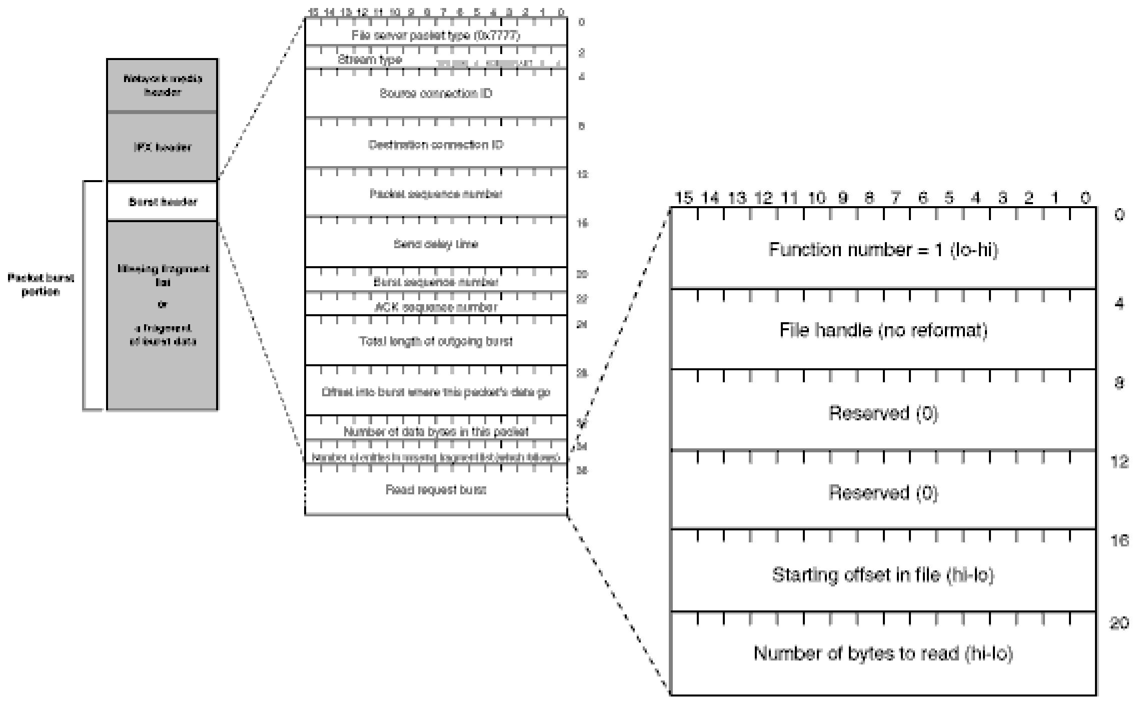

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Network media header
IPX header
Burst header
Missing fragment list or a fragment of burst data
Packet burst portion
File server packet type (0x7777)
Stream type
Source connection ID
Destination connection ID
Packet sequence number
Send delay time
Burst sequence number
ACK sequence number
Total length of outgoing burst
Offset into burst where this packet's data go
Number of data bytes in this packet
Number of entries in missing fragment list (which follows)
Read request burst
0
2
4
8
12
16
20
22
24
28
32
34
36
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Function number = 1 (lo-hi)
File handle (no reformat)
Reserved (0)
Reserved (0)
Starting offset in file (hi-lo)
Number of bytes to read (hi-lo)
0
4
8
12
16
20

Figure A-4: The request burst portion of the Burst Mode packet for a big file read.

Big File Read Reply

Figure A-5 shows the reply burst portion of the Burst Mode packet for a big file read.

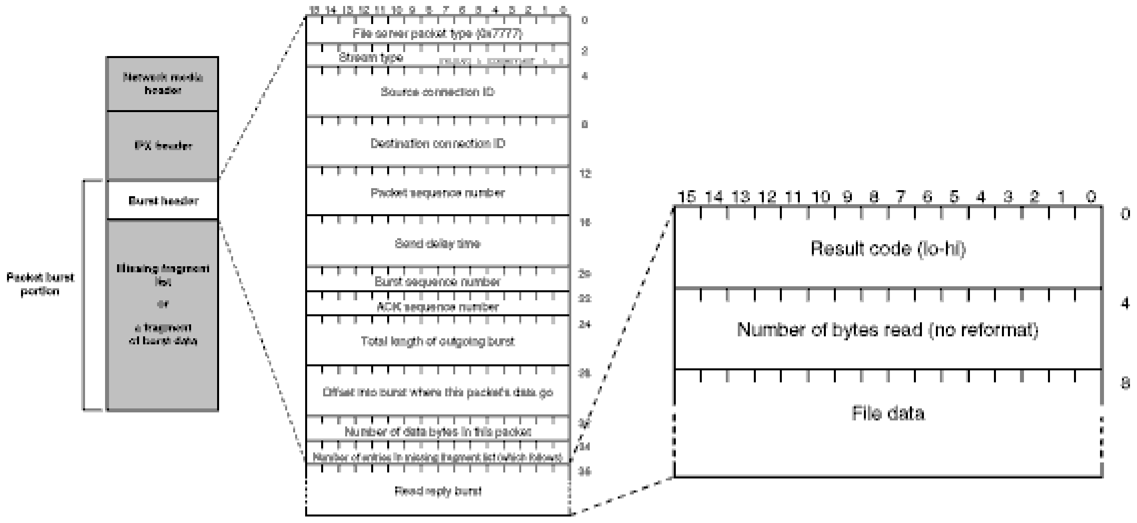


Figure A-5: The reply burst portion of the Burst Mode packet for a big file read.

If an error occurs, only the result code returns. Result code values are:

0	No error
1	Initial error
2	I/O error
3	No Data Read

Big File Write Request

Figure A-6 shows the request burst portion of the Burst Mode packet for a big file write.

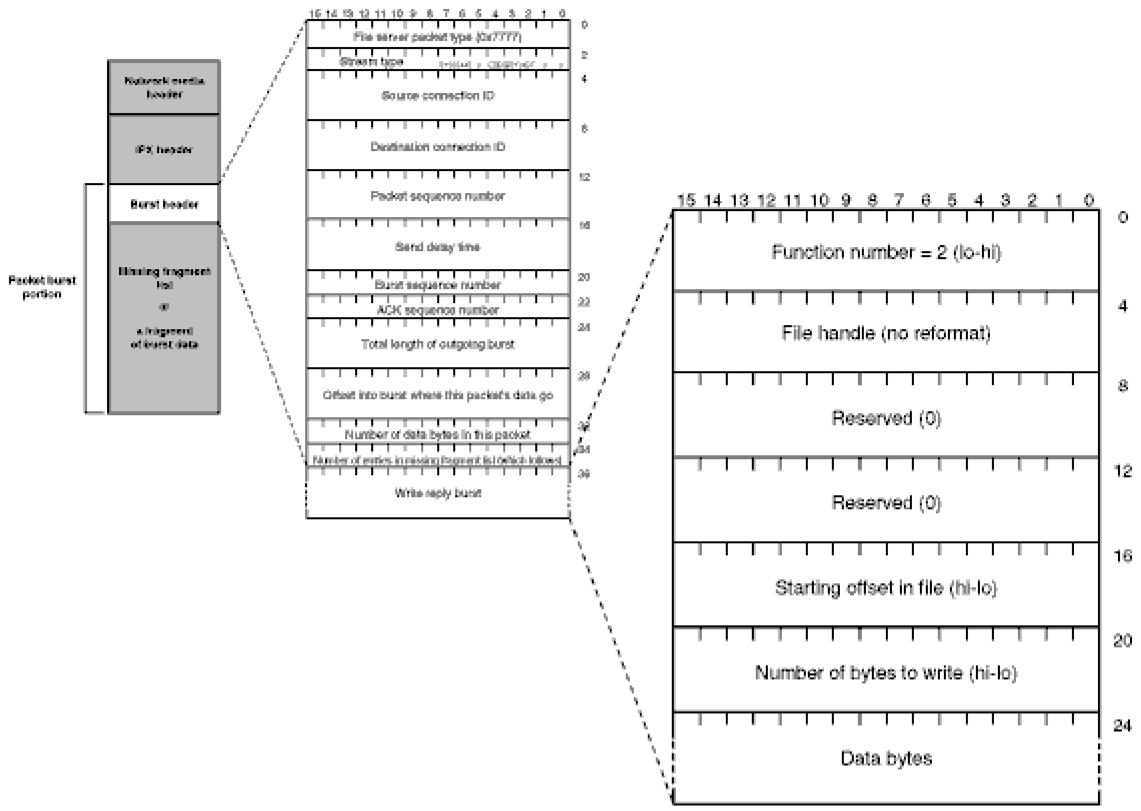
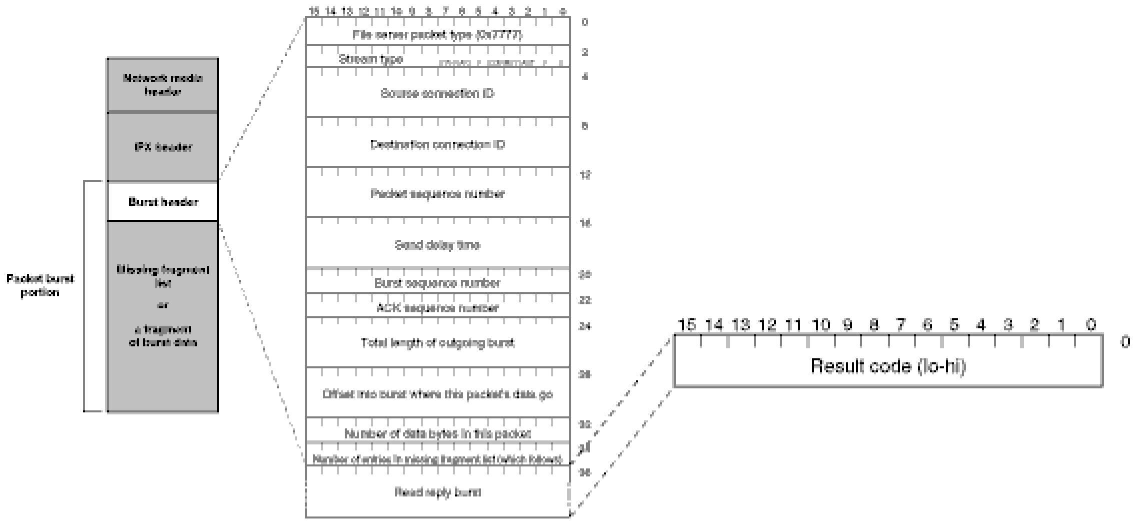


Figure A-6: The request burst portion of the Burst Mode packet for a big file write.

Big File Write Reply

Figure A-7 shows the reply burst portion of the Burst Mode packet for a big file write.



Network media header
IPX header
Burst header
Missing fragment list or a fragment of burst data
Packet burst portion
File server packet type (0x7777)
Stream type
Source connection ID
Destination connection ID
Packet sequence number
Send delay time
Burst sequence number
ACK sequence number
Total length of outgoing burst
Offset into burst where this packet's data go
Number of data bytes in this packet
Number of entries in missing fragment list (which follows)
Read reply burst
Result code (lo-hi)

Figure A-7: The reply burst part of the Burst Mode packet for a big file write.

The Result Code values are:

0	No error
4	Write Error

Write errors occur only if the disk is completely out of space. If a write to a bad disk segment is attempted, Novell Hot Fix finds suitable disk space elsewhere; the server will always write if there is room.

Connection Structures

Here is the sequence of events that occur between the client and the server when attempting to make a Burst Mode connection:

1. The workstation and server synchronize (set to zero) the packet and burst sequence numbers.
2. They exchange Burst Mode connection IDs.
3. The workstation tells the server the dynamic IPX socket number it is using for Burst Mode communication.
4. The two sides exchange information about the maximum sizes of packets and bursts each can handle; both use the smaller packet and burst size of the two.

The next two sections detail the structures used during a Burst Mode client's initial connection request and the file server's reply to that request.

Request by the Workstation

The following table details the initial workstation NCP packet burst request structure (0x2222 101 = Burst Mode Connection Request Function).

Offset	Content	Type
0	RequestType (0x2222)	word
2	SequenceNumber (LastSequence + 1)	byte
3	ConnectionNumber (Service Connection)	byte
4	TaskNumber (Current Task Number)	byte
5	Reserved (0)	byte
6	Function Code (101)	byte
7	LocalConnectionID (random)	long
11	LocalMaxPacketSize (hi-lo format)	long
15	LocalTargetSocket (from IPX open socket)	word
17	LocalMaxSendSize (hi-lo format)	long
21	LocalMaxRecvSize (hi-lo format)	long

LocalMaxPacketSize is a number returned by IPX function 1Ah. It is the number of data bytes that a physical packet will hold, excluding MAC-layer bytes.

LocalMaxSendSize and LocalMaxRecvSize specify maximum burst sizes.

Reply by the File Server

The table below shows the structure of a file server's reply to a Burst Mode connection request.

Offset	Content	Type
0	ReplyType (0x3333)	word
2	SequenceNumber (Request Sequence Number)	byte
3	ConnectionNumber (Service Connection)	byte
4	TaskNumber (Client Task Number)	byte
5	Reserved (0)	byte
6	Completion Code (0)	byte
7	RemoteTargetID (random big connection ID)	long
11	RemoteMaxPacketSize (hi-lo format)	long
15	RemoteMaxSendSize (hi-lo format)	long
19	RemoteMaxRecvSize (hi-lo format)	long

RemoteMaxPacketSize is the server counterpart of the LocalMaxPacketSize in the request packet.

RemoteMaxSendSize and RemoteMaxRecvSize specify maximum burst sizes from the server's point of view.

Appendix B: TEST.BAT

This appendix is a listing of the TEST.BAT file used to test Burst Mode performance. The three command line parameters are the input file name and path, the output file path, and an optional comment. For example:

```
TEST X:/PATH.NAM C: Site-A
```

```
ECHO %1  IS THE INPUT FILE >> OUT.LOG
```

```
          ECHO %2 IS THE OUTPUT FILE >> OUT.LOG
          ECHO %3 >> OUT.LOG
          ECHO ON

ECHO    COPY A 100 BYTE FILE 5 TIMES  >> OUT.LOG
          ECHO COPY A 100 BYTE FILE 5 TIMES
          ECHO FIRST COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEHUN NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO SECOND COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEHUN NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO THIRD COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEHUN NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FOURTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEHUN NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FIFTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEHUN NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO * >> OUT.LOG
          ECHO * >> OUT.LOG
          ECHO * >> OUT.LOG

ECHO    COPY A 1000 BYTE FILE 5 TIMES  >> OUT.LOG
          ECHO COPY A 1000 BYTE FILE 5 TIMES
          ECHO FIRST COPY
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONETHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO SECOND COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONETHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO THIRD COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONETHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FOURTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONETHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FIFTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONETHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO * >> OUT.LOG
          ECHO * >> OUT.LOG
          ECHO * >> OUT.LOG
ECHO    COPY A 10,000 BYTE FILE 5 TIMES  >> OUT.LOG
          ECHO COPY A 10,000  BYTE FILE 5 TIMES
          ECHO FIRST COPY
          TIME < CR.TXT >> OUT.LOG
          COPY %1\TENTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO SECOND COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\TENTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO THIRD COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\TENTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FOURTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\TENTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FIFTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\TENTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO * >> OUT.LOG
          ECHO * >> OUT.LOG
          ECHO * >> OUT.LOG

          ECHO    COPY A 100,000 BYTE FILE 5 TIMES  >> OUT.LOG
          ECHO COPY A 100,000 BYTE FILE 5 TIMES
          ECHO FIRST COPY
          TIME < CR.TXT >> OUT.LOG
          COPY %1\HUNTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO SECOND COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\HUNTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO THIRD COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\HUNTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FOURTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\HUNTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FIFTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\HUNTHOU NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO   * >> OUT.LOG
          ECHO * >> OUT.LOG
          ECHO * >> OUT.LOG

          ECHO    COPY A 1,000,000 BYTE FILE 5 TIMES  >> OUT.LOG
          ECHO COPY A 1,000,000 BYTE FILE 5 TIMES
          ECHO FIRST COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEMEG NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO SECOND COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEMEG NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO THIRD COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEMEG NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FOURTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEMEG NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO FIFTH COPY >> OUT.LOG
          TIME < CR.TXT >> OUT.LOG
          COPY %1\ONEMEG NUL
          TIME < CR.TXT >> OUT.LOG
          ECHO * >> OUT.LOG
          ECHO * >> OUT.LOG
          ECHO * >> OUT.LOG
```