[MC-DPL4R]: DirectPlay 4 Protocol: Reliable Specification

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

Revision Summary

Date	Revision History	Revision Class	Comments
08/10/2007	0.1	Major	Initial Availability
09/28/2007	0.2	Minor	Updated the technical content.
10/23/2007	0.2.1	Editorial	Revised and edited the technical content.
11/30/2007	1.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
01/25/2008	2.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

This document specifies the DirectPlay 4 Protocol. This protocol guarantees message delivery and provides **throttling** for applications that use DirectPlay 4.

1.1 Glossary

DirectPlay4: A programming library that implements the IDirectPlay4 programming interface. **DirectPlay4** provides peer-to-peer session-layer services to applications, including session lifetime management, data management, and media abstraction. **DirectPlay4** first shipped with the DirectX4 multimedia toolkit. Improved versions continued to ship up to, and including, DirectX9. **DirectPlay4** was subsequently deprecated. The **DirectPlay4** library continues to ship in current versions of Windows operating systems, but is no longer supported by Microsoft development tools.

DirectX Development Kit: A set of libraries (**DirectX Runtime**) and supporting infrastructure for building applications for those libraries.

DirectX Runtime: A set of libraries created for the family of Windows operating systems that provide interfaces to ease the development of video games.

Service Provider: A DirectPlay **service provider** is a **transport layer** entity that implements the IDirectPlaySP interface. The **service providers** that shipped with DirectPlay 4 are modem, serial, IPX, and TCP/IP.

Session Layer: An International Organization for Standardization (ISO) Open Systems Interconnection (OSI) **session layer**.

System Message: A message sent by one instance of DirectPlay to another instance of DirectPlay for the purposes of session management.

Throttling: The reduction in the rate of sending data when a network link saturation condition is detected.

Transport Layer: ISO OSI **transport layer**.

User Message: A message sent between instances of an application using the DirectPlay network library as a transport.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MC-DPL4CS] Microsoft Corporation, "DirectPlay 4 Protocol: Core and Service Providers Specification" September 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[US PATENT 6438603 B1] "Methods and Protocol for Simultaneous Tuning of Reliable and Non-Reliable Channels of a Single Network Communication Link", 2002 Ogus http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=6438603.PN.&OS=PN/6438603&RS=PN/6438603

1.2.2 Informative References

There are no informative references for this protocol specification.

1.3 Protocol Overview (Synopsis)

The DirectPlay 4 Protocol: Reliable Specification ([MC-DPL4R]) describes the reliable transport mechanism that may be used with the DirectPlay 4 Protocol. This mechanism provides for reliable delivery of messages, and message throttling, and for applications written for the IDirectPlay4 interface. Early implementations of **DirectPlay4** did not provide reliable delivery and throttling and as a result, in some scenarios, no reliable delivery services were available. The reliable transport mechanism combines the reliable and non-reliable delivery of messages into a single channel of data. This facilitates a single set of flow control logic that provides both reliable and non-reliable data delivery services, as specified in [US PATENT 6438603 B1].

The reliable transport mechanism used by the DirectPlay 4 Protocol is an envelope that encapsulates all messages sent between connected peers when active. That is, the mechanism becomes active when the normal DirectPlay4 connection process has completed. All messages sent to the host during the connection process do not use the reliable transport mechanism.

The application determines whether the reliable transport mechanism is activated. Use of reliable transport is exclusive of the use of some of the normally available security functionality in DirectPlay4. However, when reliable transport is activated, user-level security is not available.

1.4 Relationship to Other Protocols

The DirectPlay 4 Protocol is an envelope that wraps both system and **user messages**. This protocol is media independent as it resides in the **session layer** of the protocol stack. DirectPlay typically can be run using **service providers** for TCP/IP, IPX, modem, and serial links. From the perspective of service providers, this is an application-layer protocol. From the perspective of applications, the DirectPlay 4 Protocol is a session-layer protocol.

The DirectPlay 4 Protocol is transmitted via the TCP and UDP protocols, as specified in the DirectPlay 4 Protocol: Core and Service Providers Specification ([MC-DPL4CS]). At the discretion of the game, all of the messages listed in [MC-DPL4CS] may be transmitted via the DirectPlay 4 Protocol, as described in this document ([MC-DPL4R]).

1.5 Prerequisites/Preconditions

The DirectPlay 4 Protocol requires the DirectX 5 runtime <1>.

1.6 Applicability Statement

The DirectPlay 4 Protocol is activated only at the request of an application that is written for the IDirectPlay4 interface and written with the DirectX 5 Development Kit or a later version of the **DirectX Development Kit**.

1.7 Versioning and Capability Negotiation

There is only one version of the DirectPlay 4 Protocol. It is activated at the request of the application. It is assumed that the application has taken measures to ensure that the appropriate version of the **DirectX runtime** has been installed on all peers involved in a DirectPlay session.

If the versions of DirectPlay on all systems do not support the DirectPlay 4 Protocol, and the application has requested that the DirectPlay 4 Protocol be used, the connection process between peers fails gracefully. That is, a node with an earlier version of the DirectX runtime than version 5 that is attempting to join a DirectPlay session that has the DirectPlay 4 Protocol activated will be rejected during its join attempt.

For version negotiation between versions of DirectPlay, see [MC-DPL4CS].

1.8 Vendor-Extensible Fields

The DirectPlay 4 Protocol has no vendor-extensible fields.

1.9 Standards Assignments

There are no standards assignments for this protocol.

2 Messages

In the DirectPlay 4 Protocol, the terms "frame" and "packet" are used interchangeably. Frames and packets refer to a single payload that is passed to a lower-layer transport, which is typically constrained by the Maximum Transmission Unit (MTU) size of the network. Messages are higher-layer payloads that may be fragmented. Messages that do not fit in a single frame may span multiple frames.

The DirectPlay 4 Protocol supports three types of frames: Data, ACK, and NACK. The frame type is determined by the settings for the **ACK** (ACKNOWLEDGE) and **EXT** (EXTENSION) bits of the **flags** field, and the **nNACK** bits of the **extended flags** field in the packet.**Note** In the current implementation, the **extended flags** field is reserved for future use and MUST be set to 0.

Frame Type	ACK **bit**	EXT **bit**	nNACK **bits**
Data Frame (section 2.2.1)	Not Set	Not Set	Not Set
ACK Frame (section 2.2.3)	Set	Not Set	Not Set
NACK Frame (section 2.2.2)	Set	Set	Nonzero

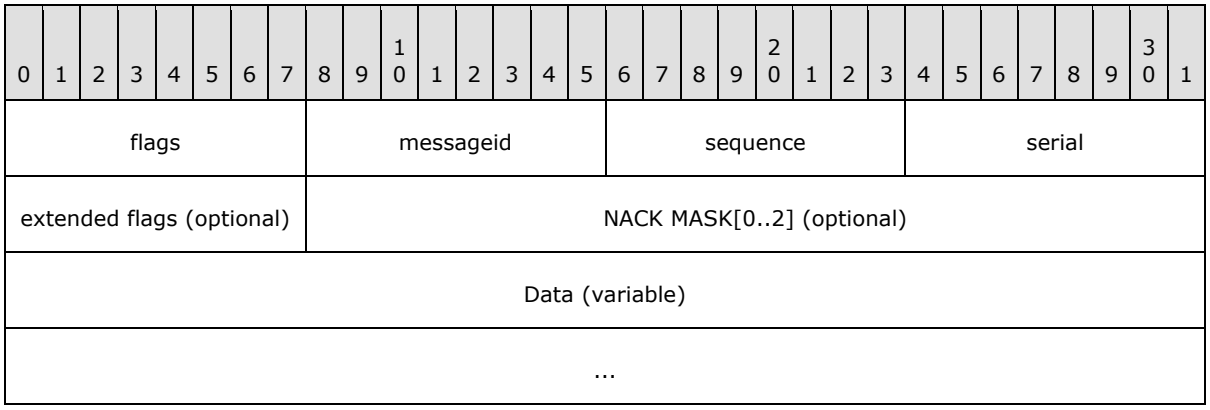
2.1 Transport

Messages are transported over DirectPlay service providers. They can use UDP, IPX, serial, and modem, or a third-party service provider.

2.2 Message Syntax

2.2.1 Frame Format, Data Frame

Data frames are messages that deliver user-specified data. The Data frames format specifies message boundaries and sequencing. Each message is identified by a **messageid** and each part of the message is identified by a sequence number. Each unique instance of a piece of a message is uniquely identified by a serial number. The sequence space is small and MAY wrap within a single message. However, a particular message is not permitted to use more than 75% of the sequence space. Message serial numbers help statistics-gathering mechanisms to differentiate between original instances of a message fragment and their retries.



flags (1 byte): A bitmask that contains values from the following table that are combined using the bitwise OR operation.

Note For a description of how these flags determine the frame type, see section 2.

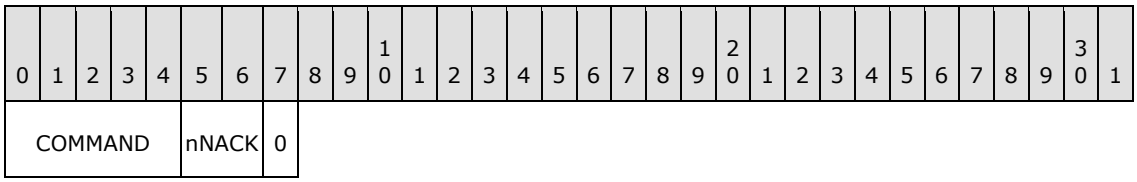
Value	Meaning
0x80	EXT(EXTENSION) - The extended flags field is present. In the current implementation, this bit MUST NOT be set because piggyback ACK/NACK was not implemented.
0x40	BIG - Big frame formatted is used. This value MUST BE 0 because the big frame format was never implemented.
0x20	CMD(COMMAND) - This bit MUST be set for Data Frames. If the EXT bit is also set, the packet MUST be ignored. In all other cases, this bit SHOULD be used only for advisory purposes.
0x10	STA(START) - The Start of a message that may span multiple protocol frames.
0x08	EOM(END OF MESSAGE) - The End of a message that may span multiple protocol frames.
0x04	SAK(SEND ACKNOWLEDGE) - A request for immediate acknowledgment after this frame is received.
0x02	ACK(ACKNOWLEDGE) - Acknowledges receipt of some frames; may specify the non-receipt of frames if the extended flags are present. In the current implementation, this bit MUST NOT be set because piggyback ACK/NACK was not implemented.
0x01	RLY(RELIABLE, ~UNRELIABLE) - Indicates the message is reliable if set and non-reliable if not set.

messageid (1 byte): A sequentially assigned value, starting at 0. The first message sent MUST be messageid 0. The second message sent is messageid 1, and so on. There MUST NOT be more than 24 outstanding messageid(s) on any link.

sequence (1 byte): The sequence number uniquely identifies the packet. It specifies the sequence space in which ACKs and NACKs are made. There MUST NOT be more than 24 outstanding sequence numbers on any link.

serial (1 byte): The serial number of a packet is the count of times this packet has been sent. If the serial number is 0, this is the first instance of the packet. If the serial number is 1, this is the first retry of this packet.

extended flags (1 byte): This field is optional. However, in the current implementation, this flag is reserved for future use and MUST be set to 0.



COMMAND (5 bits): MUST be 0x00.

nNACK (2 bits): The size in bytes of the **NACK MASK**.

Value	Meaning
0x00	There is no **NACK MASK** present; this is an INVALID value.

Value	Meaning
0x01	There is 1 byte of **NACK MASK** present.
0x02	There are 2 bytes of **NACK MASK** present.
0x03	There are 3 bytes of **NACK MASK** present.

0 (1 bit): MUST be 0.

NACK MASK[0..2] (3 bytes): A **NACK MASK** is sent in any message header when there are expected yet unreceived packets in the message stream. The mask of bits specifies which packets have not been received by the receiver.

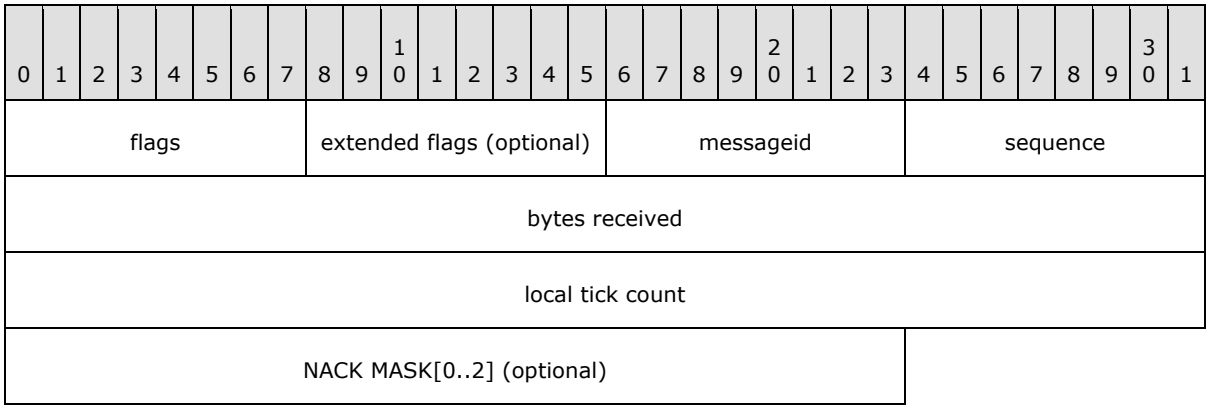
Creation of the mask is based on the sequence number in the NACK packet which identifies the first packet that was not received. The bits in the **NACK MASK** bitmask specify which other packets relative to that sequence number have also not been received. For example, if the sequence number is 7 and the **NACK MASK** is 0x03, then packets with sequence numbers 7, 8, and 9 have not been received at the receiving end of the link.

The size of the **NACK MASK** is specified in the **nNACK** bits of the **extended flags** field, if that field is present. When **nNACK** is greater than 0, there is a **NACK MASK**. When the **extended flags** field is not present, there is no **NACK MASK** field.**Note** The **NACK MASK** field is optional. However, because the **extended flags** field is not currently implemented, the **NACK MASK** field also never occurs in the existing implementation.

Data (variable): The higher layer data payload. The length of this field MUST be inferred from the remaining size of the packet reported by the lower level transport. The size is the total number of bytes in the packet minus the 4 to 8 bytes of previous Data Frame fields.

2.2.2 Frame Format, NACK Frame

NACK, or Negative Acknowledge, frames specify which data frames were expected, but were not received by the receiving end of the link. Messages that are sent unreliably MUST NOT generate a NACK.

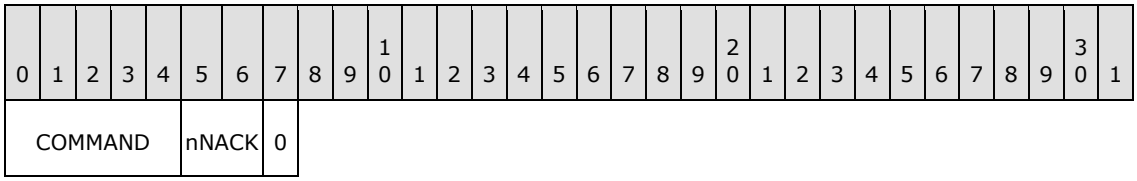


flags (1 byte): A bitmask that contains values from the following table that are combined using the bitwise OR operation.

Note For a description of how these flags determine the frame type, see section 2.

Value	Meaning
0x80	EXT(EXTENSION) - The extended flags field is present.
0x40	BIG - Big frame formatted is used. This value MUST BE 0 because the big frame format was never implemented.
0x20	CMD(COMMAND) - This bit MUST be set for Data Frames. If the EXT bit is also set, the packet MUST be ignored. In all other cases, this bit SHOULD be used only for advisory purposes.
0x10	STA(START) - The Start of a message that may span multiple protocol frames.
0x08	EOM(END OF MESSAGE) - The End of a message that may span multiple protocol frames.
0x04	SAK(SEND ACKNOWLEDGE) - A request for immediate acknowledgment after this frame is received.
0x02	ACK(ACKNOWLEDGE) - Acknowledges receipt of some frames; may specify the non-receipt of frames if the extended flags are present.
0x01	RLY(RELIABLE, ~UNRELIABLE) - Indicates the message is reliable if set and non-reliable if not set.

extended flags (1 byte): This field is optional, and when present, MUST be nonzero. However, in the current implementation, this field is reserved for future use and MUST be set to 0.



COMMAND (5 bits): MUST be 0x00.

nNACK (2 bits): The size in bytes of the **NACK MASK**.

Value	Meaning
0x00	There is no **NACK MASK** present; this is an INVALID value.
0x01	There is 1 byte of **NACK MASK** present.
0x02	There are 2 bytes of **NACK MASK** present.
0x03	There are 3 bytes of **NACK MASK** present.

0 (1 bit): MUST be 0.

messageid (1 byte): The messageid from the sent data frame.

sequence (1 byte): The sequence ID from the sent data frame. The sequence uniquely identifies which is the first non-received frame.

bytes received (4 bytes): Specifies the total number of bytes received on the link at the time this NACK frame was sent.

local tick count (4 bytes): Specifies the tick count on the local tick clock when the NACK frame was sent.

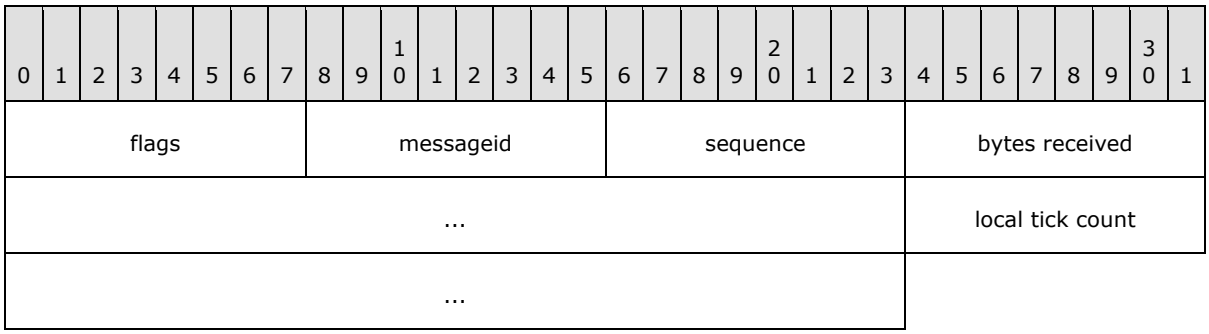
NACK MASK[0..2] (3 bytes): A **NACK MASK** is sent in any message header when there are expected yet unreceived packets in the message stream. The mask of bits specifies which packets have not been received by the receiver.

Creation of the mask is based on the sequence number in the NACK packet which identifies the first packet that was not received. The bits in the **NACK MASK** bitmask specify which other packets relative to that sequence number have also not been received. For example, if the sequence number is 7 and the **NACK MASK** is 0x03, then packets with sequence numbers 7, 8, and 9 have not been received at the receiving end of the link.

The size of the **NACK MASK** is specified in the **nNACK** bits of the **extended flags** field, if that field is present. When **nNACK** is greater than 0, then there is a **NACK MASK**. When the **extended flags** field is not present, then there is no **NACK MASK** field.**Note** The **NACK MASK** field is optional. However, because the **extended flags** field is not currently implemented, the **NACK MASK** field also never occurs in the existing implementation.

2.2.3 Frame Format, ACK Frame

ACK, or Acknowledge, frames specify which data frames have successfully arrived at the receiving end of the link.



flags (1 byte): A bitmask that contains values from the following table that are combined using the bitwise OR operation.

Note For a description of how these flags determine the frame type, see section 2.

Value	Meaning
0x80	EXT(EXTENSION) - This value MUST be set to 0.
0x40	BIG - Big frame formatted is used. This value MUST be 0 because the big frame format was never implemented.
0x20	CMD(COMMAND) - This bit MUST be set for Data Frames. If the EXT bit is also set, the packet MUST be ignored. In all other cases, this bit SHOULD be used only for advisory purposes.
0x10	STA(START) - The Start of a message that may span multiple protocol frames.
0x08	EOM(END OF MESSAGE) - The End of a message that may span multiple protocol frames.

Value	Meaning
0x04	SAK(SEND ACKNOWLEDGE) - A request for immediate acknowledgment after this frame is received.
0x02	ACK(ACKNOWLEDGE) - Acknowledges receipt of some frames; may specify the non-receipt of frames if the extended flags are present.
0x01	RLY(RELIABLE, ~UNRELIABLE) - Indicates the message is reliable if set and unreliable if not set.

messageid (1 byte): The messageid from the sent data frame.

sequence (1 byte): The sequence ID from the sent data frame. The sequence uniquely identifies which is the last successfully received frame. Acknowledgment of a particular sequence also acknowledges receipt of all prior unacknowledged sequence numbers.

bytes received (4 bytes): Specifies the total number of bytes received on the link at the time this ACK frame was sent.

local tick count (4 bytes): Specifies the tick count on the local tick clock when the ACK frame was sent.

3 Protocol Details

The DirectPlay 4 Protocol is inherently peer-to-peer. It was also designed to ride on top of existing connection and session logic of the existing DirectPlay4 implementation. Therefore, it needs no explicit connection logic or session management logic, because that is handled by the DirectPlay4 core, as specified in [MC-DPL4CS].

3.1 Common Details

3.1.1 Abstract Data Model

First Message: The messageid of the first reliable outstanding message on the outbound link. A message is outstanding if it has been transmitted but not acknowledged.

Last Message: The messageid of the last reliable outstanding message on the outbound link.

Outstanding Message Mask: A mask of bits that specifies those messages between First Message and Last Message that have not been acknowledged.

First Unreliable Message: The messageid of the first unreliable outstanding message on the outbound link.

Last Unreliable Message: The messageid of the last unreliable outstanding message on the outbound link.

Unreliable Outstanding Message Mask: A mask of bits that specifies those messages between First Unreliable Message and Last Unreliable Message that have not been acknowledged.

Unreliable Receive Queue: A queue for assembling parts of unreliable messages before they are finally assembled and indicated.

Reliable Receive Queue: A queue for assembling reliable messages and maintaining their order for indication.

First Reliable Receive: The first messageid in the reliable receive queue.

Last Reliable Receive: The last messageid in the reliable receive queue.

Reliable Receive Mask: A mask of bits that indicates which messageids relative to First Reliable Receive are reliable.

Bytes Received: Count of bytes received on this link.

Average Latency: Average latency of messages sent on this link.

3.1.2 Timers

The DirectPlay 4 Protocol utilizes the system specified in [US PATENT 6438603 B1] for estimating available bandwidth and tuning the link appropriately. However, using the scheme described in the patent is NOT required to achieve interoperability.

Timers for retries can be set according to the expected delivery time and acknowledge time, plus some amount of extra time to allow for inconsistencies in link operation.

The current implementation sets the retry time-out to the average roundtrip time plus three standard deviations in the average roundtrip time. A simpler method, such as the average roundtrip time and some extra percentage, such as 20%, may also be used.

3.1.3 Initialization

All sequence numbers and serial numbers MUST start at 0.

The count of bytes received should correspond as closely as possible to the total bytes received from the sender on the receiver's link, and should include the duplicate (retransmitted) packets.

The tick count is a 32-bit value counting 1/1000ths of a second of elapsed time. It can start at any value but must increase at the specified rate.

NACKs are NOT triggered, but only occur when missing frames are detected.

Note However, an implementation MAY attempt to trigger NACKs using a timer-based algorithm approach.

ACKs are triggered by either the SAK bit in the message header or by the end of message.

ACKs MAY wait for a timer before sending, unless the SAK bit has been specified by the sender.

3.1.4 Higher-Layer Triggered Events

There is no direct high-layer interaction with this protocol.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Data Frame Processing

When a Data frame (section 2.2.1) arrives, the value of its **messageid** field MUST be compared to the list of receiving messages using 8-bit unsigned integer math, and according to the following rules:**Note** When processing Data frames, the receiver MUST keep track of the last **messageid** that was received reliably.

- If there are no ongoing receives or completed receives, then this **messageid** MUST become the first ongoing receive.
- If the value of the **messageid** field is outside the range of the **messageid** of the earliest ongoing or completed receive (and that **messageid**+23), the Data frame MUST be ignored.
- If no message with the **messageid** value is currently being received, the **messageid** of this message MUST be added to the list of receiving messages.

The only exception to this rule is when a reliable Data frame is received and the value of its **messageid** is the same as the last reliable message received. In this case, an ACK message MUST be sent back to the sender. This exception handles the situation where an ACK that was sent for a previously received message was lost. Since a message that was previously received is now being received again, it is possible that the ACK that was previously sent was lost and needs to be retransmitted.

When the **messageid** of a message is initially added to the list of receiving messages, a new message buffer MAY be allocated for the message. This buffer is used for assembling the message according to the sequence numbers associated with the **messageid** as necessary. When all of the

following Data frames have been received, the Data (variable) areas of those messages are concatenated in sequence order to make up the receive data for that message:

- A Data frame with a particular **messageid**, a starting sequence, and a STA bit set.
- A Data frame with the same **messageid**, an ending sequence, and an EOM set.
- All Data frames with the same **messageid** in the contiguous intervening sequence space.

Note Using 8-bit unsigned integer math, if a message is received non-reliably and not all parts of the message have been received, and the currently receiving **messageid** is equal to the **messageid**+24 of this non-reliable message, the non-reliable message should be discarded.

During assembly of the receive data, a NACK or ACK MUST be sent in the following circumstances:

- If the sequence number is not the current expected sequence number, a NACK packet MUST be sent back to the sender. It is at the discretion of the implementation to send the NACK immediately or to wait for more time or more receive data before sending the NACK.<2> Delaying the NACK may result in efficiency gains.
- If the Data frame has its SAK bit set, an ACK or NACK MUST be sent immediately back to the sender.<3>
- When the first multipart message is received, an ACK MUST be sent back to the sender.

After the receive data is fully assembled, the data is indicated to the higher layers, and the **messageid** of the message MUST be removed from the list of receiving messages.

3.1.5.2 ACK and NACK Processing

When an ACK frame (section 2.2.3) arrives, the data associated with the messageid in the ACK message and any earlier messageids MAY be discarded.

When a NACK frame (section 2.2.2) arrives, the data associated with the messageid-1 in the NACK frame (section 2.2.2) MAY be discarded. Any other data is indicated as received, but the presence of a 0 bit in the NACK mask may also be discarded. Any other data MUST be retransmitted.

3.1.5.3 Bytes Received Processing

Any time that data arrives from a remote sender, the size of the entire Data Frame (section 2.2.1) is accumulated in a per link value called bytes received. This value is included in any ACK or NACK message on that link.

3.1.6 Timer Events

Timers are recommended as specified in [US PATENT 6438603 B1]. However, this scheme is not required to achieve interoperability.

3.1.7 Other Local Events

There are no other local events.

4 Protocol Examples

4.1 One-Way Traffic Between Node A and Node B

In the following examples, one-way traffic between Node A and Node B is covered. Node A sends four messages to Node B:

1. Message 1: Three Data frames (section 2.2.1) in length, reliable.
2. Message 2: One Data frame in length, non-reliable.
3. Message 3: One Data frame in length, reliable, will drop in first transmission.
4. Message 4: Two Data frames in length, non-reliable.

4.1.1 Message 1

Node A sends Message 1 to the receiver, Node B. This message is three Data frames (section 2.2.1) in length and reliable:

```
Message 1:  DATA FRAME: 1
Flags:      STA, RLY messageid: 0, sequence: 0, serial: 0
Message 1:  DATA FRAME: 2
Flags:      RLY messageid: 0, sequence: 1, serial: 0
Message 1:  DATA FRAME: 3
Flags:      EOM, RLY messageid: 0, sequence: 2, serial: 0
```

All three Data frames arrive on the receiver, Node B, and the receiver indicates this to the higher layers.

The receiver sends back an ACK frame to the sender, Node A:

```
Flags:      ACK messageid: 0, sequence: 2, bytes received: 4500
```

4.1.2 Message 2

Node A sends Message 2 to the receiver, Node B. This message is one Data frame (section 2.2.1) in length and non-reliable:

```
Message 2:  DATA FRAME
Flags:      STA, EOM messageid: 1, sequence: 3, serial: 0
```

There MAY or MAY NOT be an acknowledgment of a non-reliable frame by the receiver; this is at the discretion of the implementation. However, sending an acknowledgment allows the sender, Node A, to manage resources more efficiently.<4>

4.1.3 Message 3

Node A sends Message 3 to the receiver, Node B. This message is one Data frame (section 2.2.1) in length, reliable, and will drop the first transmission:

```
Message 3:  DATA FRAME
Flags:      STA, EOM, RLY messageid: 2, sequence: 4, serial:0
```

If Message 3 is dropped, but the sender, Node A, still sends Message 4 (section 4.1.4) to the receiver, Node B, this would result in the following:

```
Message 4:  DATA FRAME: 1
Flags:      STA messageid: 3, sequence: 5, serial: 0
Message 4:  DATA FRAME: 2
Flags:      EOM messageid: 3, sequence: 6, serial: 0
```

This MAY trigger a NACK on the receiver, Node B.

The receiver, Node B, sends back a NACK to the sender, Node A:

```
Flags:      NACK messageid: 2, sequence: 4,
            nNACK: 0, no NACK mask present, bytes received 8954
```

In response to the NACK, the sender, Node A, attempts a retransmission of Message 3 to Node B:

```
Message 3:  DATA FRAME (retry)
Flags:      STA, EOM, RLY messageid: 2, sequence: 4, serial: 1
```

4.1.4 Message 4

The sender, Node A, sends Message 4 to the receiver, Node B. This message is two Data frames (section 2.2.1) in length and non-reliable:

```
Message 4:  DATA FRAME: 1
Flags:      STA messageid: 3, sequence: 5, serial: 0
Message 4:  DATA FRAME: 2
Flags:      EOM messageid: 3, sequence: 6, serial: 0
```

This MAY trigger a NACK on the receiver. For more information, see section 4.1.3.

5 Security

5.1 Security Considerations for Implementers

There are no security considerations for this protocol.

5.2 Index of Security Parameters

There are no security parameters for this protocol.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows NT
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.5: The DirectPlay 4 Protocol is only available in Windows operating systems that have had the DirectX 5 runtime (from the DirectX 5 development kit), or later version of the runtime installed. The computer systems that support this configuration consist of all Windows PC-based operating systems, including Windows Me. Additionally, applications may have installed the DirectX 5 runtime (or later version) on a particular Windows system, in which case those computers would also have the DirectPlay 4 Protocol available.

<2> Section 3.1.5.1: The Windows DirectPlay 4 implementation sends NACKs immediately.

<3> Section 3.1.5.1: The Windows DirectPlay 4 implementation sets the SAK bit when any one of the following applies:

- The EOM bit is not set or the higher layer requested the message to be sent using reliable delivery, and the time since the last SAK request is greater than 50 milliseconds and greater than one quarter of the estimated round-trip latency of the connection.
- The packet is being retransmitted.
- There have been 24 packets since the last packet was sent with the SAK bit set.

<4> Section 4.1.2: The Windows DirectPlay 4 implementation acknowledges all non-reliable Data frames that have the SAK or EOM bit set.

7 Index