IEEE P1284.4

Draft Standard for Data Delivery and Logical Channels for IEEE Std. 1284 Interfaces

Draft D4.00
December 1, 1999

Sponsor
Microprocessor and Microcomputer Standards Committee of the IEEE Computer Society
Prepared by the IEEE P1284.4 Working Group of the Microprocessor and Microcomputer Standards Committee

Draft Revision History

Revision	Date Issued	Description
4.00	December 1, 1999	• After BRC edits - Ready for delivery to IEEE • Change bars are relative to revision 3.00
3.93 3.92 3.91	November 29, 1999	• Created during BRC review process
3.90	November 27, 1999	• Editor's proposed revisions as per ballot review and other comments

3.00	August 12, 1999	• General cleanup
		• Changed initial revision from 0x10 to 0x20
		• Eliminated vendor specific transactions (0x50-0x6F). Now reserved.
		• Added "NextSocketID" parameter to GetServiceNameReply to increase efficiency of service enumeration.
		• Defined operation when maximum packet sizes indicate no data can be transferred:
		  • Max packet size = 0: No packets will be transferred in that direction
		  • Max packet size = 6: Packets with zero-length payload may be transferred in that direction
		• New URL for Winsock-2 API specification
		• Removed AsyncSelect from sample API
		• Added Init back-off strategy and service enumeration to Implementation Issues Annex

2.00	April 5, 1999	• (Change bars indicate changes from 1.7) • Added end-of-message flag • Added section title to all section references • Added AsyncSelect • Added description of error conditions • Rearranged Transaction Channel flow control sections • Made API clause closer match to Winsock-2 API • General cleanups
1.80	May 13, 1998	• As reviewed in .4 review mtg. on May 12, 1998.
1.70	April 13, 1998	• (Change bars also indicate changes from 1.5 to 1.6) • Simplified by merging "address server" into "service discovery" and "CBT" into "1284.4" • Cleaned up text.
1.60	February 18, 1998	• From other changes noted but not made during the May 1997 meeting.
1.50	August 12, 1997	• As reviewed and edited "on the fly" during the May 1997 meeting.
1.40	May 12, 1997	• Updated to reflect changes from March/April 1997 meeting in Austin. • Most sections were reviewed and edited "on the fly" during the meeting. Sections 5.4, 5.5, 5.6, 7, 8 and the Annexes were not reviewed. • Updated SPI section.

1.30	March 31, 1997	• Updated to reflect changes from February 1997 meeting in Orlando. • Renamed “command channel” to “transaction channel”. • Allowed multiple outstanding commands on the transaction channel. • Removed MLC annex. • Lots of editing for readability and better understanding.
1.20	February 24, 1997	• Updated to reflect changes from January 1997 meeting in Albuquerque. • ConfigSocket merged into OpenChannel • Combined CBT commands and replies into transactions • Moved address server to CBT commands • Command channel flow control made the same as all other channels • Added new flow control mechanism
1.10	January 3, 1997	• Update to reflect changes up to 5.6 per IEEE P1284.4 meeting in New Orleans, 11-4 to 11-5 1996. • Cleaned up Error handling by adding replies to each command and clarifying use of the Error command. • Cleaned up footnotes to add normative info to the actual standard. • Added Init Revision negotiation • Revised and added crediting and flow control information. • Began error handling and FSM updates: categorized commands, updated tables. • Updated command and replies for clarification and errors.
1.00	October 30, 1996	Update to reflect changes covered through 5.4.1.1 per IEEE P1284.4 meeting in NY, 10-1-96. Added Service Discovery information based on approved proposals in clause 7 and Annex B. Converted all appropriate host/peripheral to primary/secondary peer terminology. Added CBT as name for IEEE 1284.4 transport. Added MLC© Legacy Issues to annex C. Continued cleanup and formatting to IEEE Standards Style Manual.
0.20	September 24, 1996	Incorporated HP’s MLC 3.8 specification technology for committee review. Incorporated data link requirements for committee review. Formatted & reworded as per committee review on 6/26/96 Converted packet offsets to hex Forced reserved values to be 0x00. Began reformatting to IEEE standards.
0.01	June 26, 1996	Formatting Changes
0.00	May 21, 1996	Initial Version

Introduction

(This introduction is not a part of IEEE P1284.4, Standard for Data Delivery and Logical Channels for IEEE Std. 1284 Interfaces.)

This standard was formally started as an IEEE effort in January 1996. Members of the IEEE P1284.4 Working Group would like to thank the Hewlett-Packard Company, and the IEEE 1284.1 and 1284.3 committees for their efforts in establishing the foundation for this standard and allowing this committee to include their work as the basis for much of this standard.

At the time this standard was completed, the key contributors to the Working Group on "Standard for Data Delivery and Logical Channels for IEEE Std. 1284 Interfaces" were as follows:

Larry A. Stein, *Chair*

Mark T. Edmead, *Secretary*
Bob McComiskie, *Secretary*

Brian Batchelder, *Editor*
Walt Scheiderich, *Editor*

Jim Anderson
Lee Farrell
Robert Gross
Joe Halpern
Reed Hinkel
Monte Johnson
Laurie Lasslo
Greg LeClair
Paul Lindemuth
Raymond Lutz
Mike Moldovan
Ron Norton
Rick Pennington
Greg Shue
Lance Spaulding
Bill Stanley
Ron Talwalkar
Randy Turner
William Wagner
Andrew Warner
Forrest D. Wright
Atsushi Yuki
Steve Zilles

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention. A patent holder (Hewlett-Packard Company) has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and nondiscriminatory, reasonable terms and conditions to all applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Contents

1. Overview

1.1 Scope

The packet protocol described by this standard allows a device to carry on multiple, concurrent exchanges of data and/or control information with another device across a single point-to-point link. The protocol is not a device control language. The protocol provides basic transport-level flow control and multiplexing services. The multiplexed information exchanges are independent and blocking of one has no effect on any other. The protocol shall operate over IEEE Std. 1284-1994 interfaces and may operate over other interfaces, such as RS-232, Universal Serial Bus (USB), and IEEE 1394-1995.

1.2 Purpose

IEEE Standard 1284-1994 defines and describes an updated PC parallel interface by adding multiple modes of operation, which provide for higher speed, bi-directional communication between devices. The IEEE Std. 1284-1994 standard does not provide anything beyond a physical protocol. The 1284.4 standard specifies a point-to-point protocol with one or more layers above the physical interface and below the application. These layers take on the functions and characteristics of the transport and session layers of the OSI model.

2. Definitions

2.1 General terminology

The following terms and acronyms are used in this standard. The definitions are not intended to be absolute, but they do reflect the use of the terms in this standard.

2.1.1 Big-endian: A data format where the most significant byte of a multi-byte object is at the lowest address and the least-significant byte is at the highest address**.**

2.1.2 Byte: An 8-bit value. In this standard, bits are read left to right with the MSB (Most Significant Bit) labeled 7 to LSB (Least Significant Bit) labeled 0 as shown in the following figure:

Bit 7 - MSB	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 – LSB

2.1.3 Connection: A persistent communications path between two endpoints.

2.1.4 Device Control Language: A language used to monitor and/or control the state of a device.

2.1.5 Endpoint: A consumer or producer of data on a communication link.

2.1.6 Flow control: The function performed by a receiving entity to limit the amount of data that is sent by a transmitting entity.

2.1.7 ISO: Acronym for International Organization for Standardization. A voluntary, non-treaty organization whose members are designated standards bodies of participating nations, and non-voting observer organizations.

2.1.8 Logical channel: One connection over a single physical communication link. There may be multiple logical channels over a single physical communication link.

2.1.9 OSI model: Acronym for Open Systems Interconnect model. A seven-layer network communications model developed by an ISO subcommittee that governs communications interchange between systems. The model is an internationally accepted framework of standards for intersystem communications.

2.1.10 Packet: A group of bytes, including address, data, and control elements.

2.1.11 Point-to-point communications: Communications that take place between exactly two devices.

2.1.12 Service discovery: The function of providing transport clients with the ability to dynamically query service availability within a peer transport entity.

2.2 1284.4-specific terminology

2.2.1 Application: A software component that uses the 1284.4 transport.

2.2.2 Blocked channel: A channel with at least one blocked endpoint.

2.2.3 Blocked endpoint: An endpoint that is unable to send data on a connection due to lack of credit. A blocked endpoint becomes unblocked when it is granted credit to send data on that connection.

2.2.4 Channel: An independently flow-controlled connection between a socket on the primary device and a socket on the secondary device. It provides a logical conduit for moving data between the two endpoints.

2.2.5 Client: Name used to refer to the software component on one device that uses the services provided by a server on another device. The communication between the client and the server is managed by the protocol defined in this standard.

2.2.6 Command: The instruction sent from an initiator to a target directing the target to execute a specified process.

2.2.7 Conversation: Canonical name given to the interactions between two 1284.4 peers during the period of time bracketed between corresponding **Init** and **Exit** transactions.

2.2.8 Credit: A value issued by a 1284.4 implementation on behalf of an endpoint indicating the number of packets that may be received by that endpoint.

2.2.9 End-of-message packet: The last packet of a message in the data stream.

2.2.10 Initiator: The 1284.4 implementation that initiated a transaction by sending a command.

2.2.11 Message: A set of ordered data (possibly empty) that includes a message boundary indication. Message data may span multiple packets. A packet shall not hold data from more than one message.

2.2.12 Multiple Outstanding Transactions: A state where more than one transaction has been issued and is pending completion.

2.2.13 Out-of-band packet: A packet of exceptional data in the data stream. This type of packet may contain special information relating to the data at this position in the data stream, for example, “End of Job.” Information contained in the out-of-band packet is outside of the scope of this standard.

2.2.14 Piggyback Credit: Term used to describe the inclusion of credit inside a 1284.4 packet header. By using piggyback credit, 1284.4 can reduce the overhead of **Credit** transactions when processing a channel on which data is flowing bidirectionally.

2.2.15 Primary device: The device that issued the **Init** transaction that started the current 1284.4 conversation. There is no priority implied by the assignment of primary and secondary

devices once the conversation has been established. The assignment of primary status is used only to identify the device that contains the primary sockets.

2.2.16 Primary Socket Identifier (PSID): A socket number identifying a particular endpoint on the primary device.

2.2.17 Reply: The response sent from a target to an initiator indicating that the target has successfully or unsuccessfully executed the process specified by the command originally sent from the initiator to the target.

2.2.18 Reserved: Any protocol elements identified as "reserved" are intended for future standardization. Reserved elements shall not be used. Reserved fields or bits shall be set to 0 and shall not be checked. Values not defined in this standard are reserved.

2.2.19 Secondary device: The device that was the target of the **Init** transaction that started the current 1284.4 conversation. There is no priority implied by the assignment of primary and secondary devices once the conversation has been established. The assignment of secondary status is used only to identify the device that contains the secondary sockets.

2.2.20 Secondary Socket Identifier (SSID): A socket number identifying a particular endpoint on the secondary device.

2.2.21 Server: The software component on one device that provides services for use by clients on the same or another device. The transfer of data between the client and the server is managed by the protocol defined in this standard.

2.2.22 Service Name: The canonical name of a particular service available for use by a 1284.4 client.

2.2.23 Socket: Canonical name given to a 1284.4 endpoint to which or from which data is transmitted.

2.2.24 Socket ID (identifier): A byte used to uniquely identify the socket. A socket ID can be well known or can be dynamically assigned.

2.2.25 Target: The peer 1284.4 implementation to which the initiator's command is being sent.

2.2.26 Transaction: A 1284.4 command exchange, including both the command and the reply.

2.2.27 Transaction Channel: The channel used for 1284.4 commands and replies. The primary socket identifier and the secondary socket identifier for the Transaction Channel are both 0x00.

3. Features and compliance

3.1 Overview

It is the intention of this document to complement various types of bi-directional, point-to-point links, such as those specified in IEEE Std. 1284-1994, by providing a simple, flexible and scaleable transport protocol. For the purposes of this standard, a point-to-point link may be a virtual connection over other physical topologies. It is assumed that this point-to-point link is error free.

A device that implements 1284.4 functionality provides the ability to carry on multiple, concurrent, independent exchanges of information, whether control or data, over a single point-to-point link. This service is provided by use of the packet-based, non-blocking, flow control system defined in this specification.

3.2 Features

IEEE 1284.4 specifies a packet-based, bi-directional, connection-oriented, peer-to-peer communication protocol for a point-to-point link encompassing the following features:

a) Multiple, concurrent, logical channels: non-blocking, independent information exchange.

b) Flow control to keep the channels independent
   1) Negotiation of packet sizes.
   2) Transmission rate control. Also known as pacing.
   3) Specification of minimum and maximum packet sizes.

c) Service discovery
   1) Dynamic query and mapping of channel address space.
   2) Service names maintained in a 3rd-party, open, persistent registry (see Annex B - Service Names Registry).

d) Low system overhead: small, fixed header allowing easy encoding and decoding of packets.

e) Extensibility
   1) Control byte provides simple out-of-band and end-of-message signaling.
   2) Within a conversation, services can be assigned to previously unallocated sockets.

f) Information exchange independence: the transport protocol is not dependent upon the data being exchanged.

g) Physical link independence: the protocol can be used in conjunction with any physical link, provided a data link layer of sufficient functionality is available (see clause 6 - Data link service requirements).

3.3 Compliance criteria

A 1284.4-compliant device shall provide all commands and functionality in the following clauses:
- clause 4 - Theory of operation of the 1284.4 protocol and
- clause 5 - 1284.4 Transactions.

A compliant device shall use a data link that meets the requirements enumerated in clause 6 - Data link service requirements.

1284.4-compliant devices are not required to implement multiple outstanding transactions (see 4.5.6 - Multiple Outstanding Transactions).

3.4 MLC compatibility criteria

IEEE 1284.4 is based upon the MLC protocol developed by Hewlett-Packard Company™. MLC devices are not necessarily compliant with IEEE 1284.4. For details on the MLC protocol, refer to annex A - Bibliography.

4. Theory of operation of the 1284.4 protocol

4.1 Overview

1284.4 is a credit-based transport protocol. It provides a mechanism to establish, configure, and control communication between two endpoints. It provides this functionality through a well-defined packet structure and transaction protocol. 1284.4 transactions are symmetrical: each command has a corresponding reply.

1284.4 transports client data on logical channels. These logical channels are multiplexed on a single physical link. Client data is exchanged over data channels. 1284.4 transactions are exchanged over the 1284.4 Transaction Channel. All client data and 1284.4 transactions are encapsulated in packets. Packet transfer is managed using a credit-based flow control process to pace the exchange.

1284.4 also provides a service discovery mechanism to dynamically map service names to socket numbers.

4.2 General packet structure

All 1284.4 packets follow the general structure illustrated in Table 1. Each packet shall contain the 6-byte header and 0 or more data bytes. The packet header contains 5 fields detailed in Table 2: PSID, SSID, Length, Credit, and Control. To eliminate ambiguity, all numbers in the following descriptions that are in hexadecimal are designated beginning with “0x” and are in big-endian byte order. Packet fields are listed in the order they are transmitted reading from left to right. The length includes the header. The maximum amount of data that may be transmitted in the Payload field is 65,529 bytes (0xFFF9). The offset and length values specified in Table 2 are unsigned integers.

Table 1--1284.4 packet structure

Header					Payload
PSID 1 byte	SSID 1 byte	Length 2 bytes	Credit 1 byte	Control 1 byte	0-0xFFF9 bytes

Table 2--1284.4 packet field descriptions

Name	Offset	Length	Description
Primary Socket ID (PSID)	0x00	1 byte	An unsigned byte indicating the primary socket ID of the packet.
Secondary Socket ID (SSID)	0x01	1 byte	An unsigned byte indicating the secondary socket ID of the packet.
Length	0x02	2 bytes	An unsigned word that specifies the length, in bytes, of the entire packet. The length includes the packet header. Valid values range from 0x0006 to 0xFFFF.
Credit	0x04	1 byte	A piggyback credit field that specifies the number of additional packet credits the sender of the packet is issuing to the receiver of the packet. These credits are for data to be sent in the future by the receiver of this packet on the channel indicated by the PSID

			and SSID fields.
Control	0x05	1 byte	A bit field containing information for controlling operation of the 1284.4 protocol. It is a bit-encoded field with the following definitions: Bit(s) Definition 7-2 Reserved (0) 1 0 = Normal packet 1 = End-of-message packet 0 0 = Normal packet 1 = Out-of-band packet
Payload	0x06	0-65,529 bytes	If PSID and SSID indicate the 1284.4 Transaction channel, the payload field contains a 1284.4 command or reply. If PSID and SSID indicate any other channel, the payload field contains application data for that channel.

4.3 Communication procedures

4.3.1 1284.4 conversation management

A conversation is the set of interactions between two 1284.4 peers during the period of time bracketed between corresponding **Init** and **Exit** transactions.

The following sections describe how 1284.4 conversation management is handled, including the 1284.4 transactions that are used to establish and terminate the conversation. Once the conversation is established, 4.3.2 - Channels explains how the channels are opened, data is transferred, and channels are closed. Details of the transactions are discussed in clause 5 - 1284.4 Transactions.

4.3.1.1 Conversation initiation

When one 1284.4 peer wishes to establish a 1284.4 conversation, it does so by initiating an **Init** transaction on the 1284.4 transaction processor channel. The target of the **Init** transaction will complete the transaction.

The device that generates the successful **Init** transaction becomes the primary device for the duration of the 1284.4 conversation. The device that completes the **Init** transaction becomes the secondary device for the duration of the 1284.4 conversation.

During this exchange each side of the link indicates which revision of the protocol it supports. All 1284.4 implementations shall support the 0x20 revision of the protocol defined by this standard.

4.3.1.2 Conversation termination

The **Exit** transaction is used as a companion to the **Init** transaction. It is used to terminate 1284.4 conversations. The normal termination process is to first complete all outstanding transactions and close all open channels, and then to initiate an **Exit** transaction.

4.3.2 Channels

A channel is defined as a logical connection between two socket endpoints. The minimum number of supported channels is one: the 1284.4 Transaction Channel. The maximum number of channels is 65026 ((255*255)+1). Implementations may support fewer channels.

4.3.2.1 Socket IDs

A primary socket ID (PSID) and a secondary socket ID (SSID) are used to identify the endpoints of a specific channel on which communication occurs. Socket IDs range from 0x00 to 0xFF. Communication between 1284.4 peers occurs on the 1284.4 Transaction Channel. The PSID and SSID for the Transaction Channel are both 0x00.

Other sockets are dynamically allocated by 1284.4 peers and assigned to applications wishing to communicate with services on the other side of the link. Service Discovery (see 4.4) provides a mechanism used to locate those services. Unlike other channels, the Transaction Channel is assumed to be open when 1284.4 is initialized. No open or close of the Transaction Channel is allowed. See 4.3.2.2 - Packet size and 4.5.5 - 1284.4 transaction channel flow control for details on the transaction channel.

4.3.2.2 Packet size

The maximum size of packets in both primary-to-secondary and secondary-to-primary directions on the 1284.4 transaction channel is fixed at 64 bytes (6 byte header plus up to 58 bytes for the commands and replies). Fewer than 64 bytes may be sent in a 1284.4 transaction packet.

The maximum packet size on all other channels is limited by the two-byte length field to a maximum of 65,535 bytes (6 byte header plus up to 65,529 bytes of data). The actual maximum packet sizes supported will be implementation specific. When a channel is opened the maximum sizes of both primary-to-secondary and secondary-to-primary packets are negotiated and can be different. Packets smaller than the negotiated maximum size may be sent.

The minimum packet size is 6 bytes (6-byte header plus zero bytes of data).

4.3.2.3 Opening a channel

Once the 1284.4 conversation has started, channels may be established between endpoints on the 1284.4 peers. Channels are established using the **OpenChannel** transaction. Data packets shall only be transmitted on a channel while that channel is open.

4.3.2.4 Channel credit

Credit is the fundamental mechanism by which the flow of data on 1284.4 channels is controlled (see 4.5 - Data transfer and flow control). A receiving 1284.4 peer, having knowledge of how many packets it can accept, uses credit to control the number of packets the sender can transmit across each open channel. Using credit to control the flow of data allows 1284.4 to avoid the overhead of interlocked send-receive-acknowledgment transmissions. This provides a simple mechanism for the receiver to prevent the sender from overwhelming its resources with packet transmissions. Credit is granted by **OpenChannel**, **Credit**, or **CreditRequest** transactions, or by piggyback credit in a packet.

4.3.2.5 Channel data exchange

Once a channel is open and credit has been granted, data packets may be freely exchanged between the two endpoints as long as the sender has credit to do so. Credit may be piggybacked in the data packets. If the data stream is bi-directional (i.e., data is flowing in a primary-to-secondary direction as well as a secondary-to-primary direction), piggybacking credit reduces the crediting traffic.

4.3.2.6 Closing a channel

When either 1284.4 peer determines the channel is no longer necessary, it closes the channel by initiating a **CloseChannel** transaction.

4.3.3 Special Data Packets

4.3.3.1 End-of-message Packets

A message is a set of ordered data (possibly empty) that includes a message boundary indication. Message data may span multiple packets. A packet shall not hold data from more than one message.. Setting the "End-of-message Packet" flag in the packet header indicates the message boundary. A packet with the end-of-message flag set may or may not contain data. The message boundary is after the last byte of data in the packet, or after the last byte of data previously transferred if the flagged packet has no data. The sending client indicates which packets are to be marked end-of-message. The 1284.4 peer transport shall pass the end-of-message indication to the receiving client. All data within each channel shall be delivered in order. All other processing of end-of-message data shall be in the same manner as non end-of-message data. End-of-message packets are not supported on the 1284.4 Transaction Channel.

4.3.3.2 Out-of-band Packets

Packets can be marked as out-of-band by setting the "Out-of-band Packet" flag in the packet header. The sending client indicates which packets are to be marked out-of-band. The 1284.4 peer transport will pass the out-of-band indication to the receiving client. Out-of-band data shall not be split across multiple packets. Out-of-band data shall not be combined with in-band data. All data within each channel shall be delivered in order, regardless if it is in-band or out-of-band. All other processing of out-of-band data shall be in the same manner as non out-of-band data. Out-of-band packets are not supported on the 1284.4 Transaction Channel.

4.3.4 Errors

4.3.4.1 Error indications

Two mechanisms are provided for communicating errors to the 1284.4 peer. Errors detected within a transaction can be communicated in the transaction reply. Other errors can be communicated using the **Error** packet.

4.3.4.2 Response time

There are no response time requirements specified by the 1284.4 protocol. Implementers should consider using timers to avoid potential deadlock situations.

4.3.4.3 Other Error Conditions

Packets other than the **Init** command received outside of a conversation shall be ignored. The implementer shall handle any other unspecified error conditions.

4.4 Service discovery

4.4.1 Overview

Service discovery provides a standard method for 1284.4 client applications to locate a particular service application. It provides the ability to dynamically determine service name to socket ID mapping.

4.4.2 Service names

Canonical names of services are maintained by an administrative authority. Names may be registered by vendors through the administrative authority, along with a description of the service functionality. IEEE 1284.4 service names shall be expressed within the 7-bit ASCII character set

with a maximum length of 40 characters. Service names are limited to the set of uppercase letters, digits, and the punctuation character hyphen. They must start with a letter, and end with a letter or digit. The registry process is defined in Annex B - Service Names Registry.

4.4.3 Service Discovery protocols

Service Discovery is supported through the **GetSocketID** and **GetServiceName** transactions. These complementary transactions allow name-to-socket and socket-to-name conversions.

4.5 Data transfer and flow control

4.5.1 Introduction

Within the context of 1284.4, flow control ensures that a sender only sends data when the receiver is ready to accept the data. This prevents a channel from blocking other channels. It also allows the transport layer to avoid retransmitting data. To provide flow control in 1284.4, the receiver informs the sender how much data can be sent. The sender then limits the amount it sends to this value.

The maximum amount of data that may be sent is communicated in two steps. First, the maximum size of the packets to be exchanged is negotiated. Second, during data transfer, the receiver of the data communicates to the sender the number of packets that can be sent (credits). The product of the maximum packet size and the number of credits specifies the maximum amount of data that can be sent by the sender before additional credit is granted. The maximum size of packets remains constant while the channel is open. The receiver enables further data transfer by granting more credits to the sender.

4.5.2 Packet size negotiation

Packet size negotiation sets the maximum size of packets, per channel, in both the primary-to-secondary and secondary-to-primary directions. This is done when the connection is established, using the **OpenChannel** transaction. The packet size can differ with direction. A zero-length packet size indicates that no data packets will be sent in that direction. The maximum size of packets remains constant while the channel is open. Packet size negotiation allows optimization of data transfers within the current 1284.4 conversation.

4.5.3 Credit

The receiver grants credit to the sender to indicate that the receiver is prepared to accept a specific number of data packets. Credit is associated with packets on a one-for-one basis; i.e. each credit grants the sender permission to send one packet. An endpoint may only send data across a channel if it has the credit to do so.

Credit used for 1284.4 communications is accumulative. When a sender is granted credit, it adds the additional credit to any unused credit it has already been granted. As the sender sends packets, it subtracts the corresponding credits from its available credit. Once a sender's credit reaches zero, it shall not send any more packets until it is granted more credit for that channel.

Initial credit can be granted in the **OpenChannel** transaction. Once a channel is open, credit can be granted via a **Credit** transaction or requested via a **CreditRequest** transaction. To avoid the overhead of these explicit credit commands, credit can also be granted by piggybacking credit in 1284.4 packets. All outstanding credit for a given channel shall be discarded upon the completion of a **CloseChannel** transaction.

Flow control of the 1284.4 conversation is controlled by 1284.4 peers managing the credit by using the modes detailed in 4.5.4 - 1284.4 data flow control and 4.5.5 - 1284.4 transaction channel flow control, and preventing deadlock as described in 4.5.7 - Deadlock avoidance and blockage prevention.

4.5.4 1284.4 data flow control

The sender controls a parameter to manage the receiver's granting of credit. The sender sets this parameter using a field in the **OpenChannel** and **CreditRequest** commands. The receiver grants credit via the **OpenChannelReply**, **Credit**, or **CreditRequestReply** commands, or by piggybacking credit in a data packet.

The credit control parameter is called MaximumOutstandingCredit. MaximumOutstandingCredit is the maximum amount of credit that the sender will need at any given time. The receiver shall attempt to maintain the sender's credit balance equal to the value of this parameter.

The initial value of the MaximumOutstandingCredit parameter is set by the **OpenChannel** transaction. The sender may subsequently change the credit parameter at any time by sending a **CreditRequest** command. Changing the MaximumOutstandingCredit parameter does not change the available credit that the sender has already accumulated.

The MaximumOutstandingCredit parameter defines the following modes:

a) "No credit" mode: when the MaximumOutstandingCredit parameter is set to 0x0000 the sender is asking the receiver to send no credit.

   This may be used at the start of communications by an **OpenChannel** command when the sender wants to open a channel prior to having data ready to send and does not want to burden the receiver with having to supply credit until it is needed. It may also be used at the end of a channel packet exchange, e.g. the sender wishes to keep the channel open but has no more data to send at that time.

b) "Unlimited credit" mode: when the MaximumOutstandingCredit parameter is set to 0xFFFF, the sender is asking the receiver to send as much credit as possible.

   This may be used when the sender has a large or unknown number of packets to send. As it approaches the end of job, the sender may monitor the number of packets left to be sent so that it can switch to "no credit" mode.

c) "Maintain credit" mode: when the MaximumOutstandingCredit is in the range 0x0001 to 0xFFFF, the receiver will try to keep the sender's outstanding credit at the amount specified by the MaximumOutstandingCredit parameter.

   This mode may be used when the sender will be sending bursts of packets, where the burst size will never exceed the MaximumOutstandingCredit parameter. Since the maximum number of credits needed at any time is known, the receiver can efficiently manage its buffers.

The sender is not required to use all of the above modes. For example, the sender may use unlimited credit mode for every channel and allow the receiver to clean up resources only as channels are closed. The sender may also start in "no credit" mode, switch to "maintain credit" mode to send bursts of packets, and finish by switching back to "no credit" mode. The three modes provide the sender with flexibility in managing its need for credit and not burden the receiver with unnecessary requests or crediting.

The credit modes are hints for the receiver. The receiver may grant more than the requested number of credits, even in "no credit" mode.

Table 3 and Table 4 are examples of flow control of 1284.4 data packets associated with opening a channel between primary and secondary 1284.4 peers, granting initial credit, sending data, granting additional credit and closing the channel. Time increases with each row as you move down the tables from top to bottom. In addition, only the sending of packets is shown. The reception of packets is implied (e.g., an **Init** command is sent by the primary 1284.4 peer but the tables do not show the secondary 1284.4 peer receiving it). The tables do not show the transactions required to establish the 1284.4 conversation. Transaction channel flow control is detailed in 4.5.5 - 1284.4 transaction channel flow control.

Table 3 is an example of a unidirectional channel. Unidirectional channels are used when data flows in only one direction across the link. Common uses of unidirectional channels include print data and scan data.

Table 3--Flow control of 1284.4 data packets (unidirectional channel)

Time	Primary 1284.4 peer	Secondary 1284.4 peer	Primary peer data credit	Secondary peer data credit
0			0	0
1	**OpenChannel** (unlimited credit mode)		0	0
2		**OpenChannelReply** grant 5 credits	0+5=5	0
3	Primary To Secondary Data Packet 1		5-1=4	0
4	Primary To Secondary Data Packet 2		4-1=3	0
5	Primary To Secondary Data Packet 3		3-1=2	0
6		**Credit** grant 2 credits	2	0
7	**CreditReply**		2+2=4	0
8	Primary To Secondary Data Packet 4		4-1=3	0
9	Primary To Secondary Data Packet 5		3-1=2	0
10	**CloseChannel**		2	0
11		**CloseChannelReply**	0	0

Table 4 is an example of a bidirectional channel. Bidirectional channels are used when data flows in both directions across the link. Common uses of bidirectional channels include control, status, and other request-reply service.

Table 4--Flow control of 1284.4 data packets (request-reply channel)

Time	Primary 1284.4 peer	Secondary 1284.4 peer	Primary peer data credit	Secondary peer data credit
0			0	0
1	**OpenChannel** (maintain credit mode - max outstanding credit = 1)		0	0
2		**OpenChannelReply** grant 1 credit (maintain credit mode - max outstanding credit = 1)	0+1=1	0
5	Primary To Secondary Packet1 Piggyback 1 credit		1-1=0	0+1=1
8		Secondary To Primary Packet1 piggyback 1 credit	0+1=1	1-1=0
5	Primary To Secondary Packet2 Piggyback 1 credit		1-1=0	0+1=1
8		Secondary To Primary Packet2 piggyback 1 credit	0+1=1	1-1=0
10	**CloseChannel**		1	0
11		**CloseChannelReply**	0	0

4.5.5 1284.4 transaction channel flow control

Flow control for the transaction channel is similar to flow control for data channels, with the following exceptions:

a) 1284.4 command and reply packets have a maximum size of 64 bytes.
b) The transaction channel is always open. Neither **OpenChannel** nor **CloseChannel** transactions of the transaction channel are permitted.
c) After a successful Init, there is credit on the transaction channel for 1 command in each direction, regardless of the previous credit values. Commands contain piggyback credit of exactly one (1), which is used for the reply to that command. Replies contain piggyback credit of exactly one (1), which is used for the next command.
d) The transaction channel defaults to “maintain credit” mode, with a MaximumOutstandingCredit of one (1) (see 4.5.4 - 1284.4 data flow control).
e) Transaction channel credit shall be used to complete outstanding transactions before it is used to issue new transactions. Only when there is more credit than there are pending replies can a new transaction be issued.
f) **Error** and **Init** transactions can always be sent. Credit balances are unaffected by the **Error** command. The **Init** transaction resets the credit balance to one (1).

In the following tables, only the sending of packets is shown. The reception of packets is implied (e.g., an **Init** command is sent by the primary 1284.4 peer but the table does not show the secondary 1284.4 peer receiving it).

Table 5 shows an example of the commands and credits associated with establishing a 1284.4 conversation, and opening and closing a channel between a primary and secondary 1284.4 peer. Note that after a successful **Init** transaction both the primary and secondary peers have one credit for issuing a command.

Table 5--Flow control of 1284.4 transaction packets

Time	Primary 1284.4 peer	Secondary 1284.4 peer	Primary peer transaction channel credit	Secondary peer transaction channel credit
0			Undefined	Undefined
1	**Init**		Unchanged	1
2		**InitReply**	1	1
3	**OpenChannel**		1-1=0	1+1=2
4		**OpenChannelReply**	0+1=1	2-1=1
5		**CloseChannel**	1+1=2	1-1=0
6	**CloseChannelReply**		2-1=1	0+1=1

4.5.6 Multiple Outstanding Transactions

It is permitted to issue multiple outstanding transactions; i.e. an initiator may issue another transaction without waiting for the completion of a previous transaction. This is only permitted when the target has granted extra credit on the transaction channel. An initiator can issue only one outstanding transaction referencing a particular channel. The transactions do not need to be completed in order. The reply carries enough information to match it to the appropriate command. 1284.4 implementations are not required to support multiple outstanding transactions. The **CreditRequest** transaction is used to request extra credit on the transaction channel.

Flow control for multiple outstanding transactions differs from 4.5.5 - 1284.4 transaction channel flow control as follows:

a) 1284.4 initiators may request multiple outstanding transactions by setting MaximumOutstandingCredit to a value greater than one (1) using a **CreditRequest** command. Targets that support multiple outstanding transactions can then enable them by granting additional credit on the Transaction channel in the **CreditRequestReply** or a subsequent **Credit** command. Implementations that don't support multiple outstanding commands shall not grant any additional credit.

b) Each 1284.4 peer can issue only one outstanding transaction referencing a particular channel. For example, it is only acceptable to issue more than one **Credit** transaction concurrently if the **Credit** transactions refer to different channels.

Table 6 shows an example of multiple outstanding transactions. Included are the commands and credits associated with establishing a 1284.4 conversation, discovering the socket IDs for two services, and opening channels to those services.

Table 6--Flow control of 1284.4 transaction packets (multiple outstanding transactions)

Time	Primary 1284.4 peer	Secondary 1284.4 peer	Primary peer transaction channel credit	Secondary peer transaction channel credit
0			Undefined	Undefined
1	**Init**		Unchanged	1
2		**InitReply**	1	1
3	**CreditRequest** Request MaximumOutstandingCredit of 3 on transaction channel		1-1=0	1+1=2
4		**CreditRequestReply** Grant 0 additional credits on transaction channel	0+1=1	2-1=1
5		**Credit** Grant 2 additional credits on transaction channel	1+1+2=4	1-1=0
6	**CreditReply**		4-1=3	0+1=1
7	**GetSocketID** "Service A"		3-1=2	1+1=2
8	**GetSocketID** "Service B"		2-1=1	2+1=3
9		**GetSocketIDReply** "Service A"	1+1=2	3-1=2
10		**GetSocketIDReply** "Service B"	2+1=3	2-1=1
11	**OpenChannel** "Service A"		3-1=2	1+1=2
12	**OpenChannel** "Service B"		2-1=1	2+1=3
13		**OpenChannelReply** "Service B"	1+1=2	3-1=2
14		**OpenChannelReply** "Service A"	2+1=3	2-1=1

4.5.7 Deadlock avoidance and blockage prevention

1284.4 implementations shall avoid deadlocking on credit. Deadlock occurs whenever the receiver is waiting for the sender to request credit while the sender is waiting for the receiver to grant credit. This is an exceptional condition, since either the sender failed to request credit or the receiver failed to grant credit. All senders shall implement a “fail-safe” mechanism to request credit if a deadlock occurs. This mechanism can be as simple as using a relatively long deadlock time-out before requesting credit. The length of this time-out should be sufficiently long to avoid asking for credit before the receiver has had the chance to consume the previously sent data and grant more credit.

1284.4 implementations shall prevent blocking of a channel due to activity on other channels. Examples of buffer allocation issues are included in E.4 - Allocating buffers to avoid channel interaction.

5. 1284.4 Transactions

5.1 Overview

The transactions described in this section of the document are composed of transport commands and replies. The initiator of a transaction is the 1284.4 peer that sends the command starting the transaction and receives the reply completing the transaction. The target of a transaction is the 1284.4 peer that receives the command starting the transaction and sends the reply completing the transaction. The initiator assumes the transaction to have started when it sends the command and to be complete when it receives the reply. The target assumes the transaction to have started when it receives the command and to be complete when it sends the reply.

Transactions can be simultaneously initiated by both peers. Neither peer shall assume that the next incoming packet will be the reply to the command it just sent.

5.2 Transaction summary

Table 7 summarizes the transactions exchanged between 1284.4 peers. These transactions consist of matched commands and replies. Reply codes are formed by setting bit 7 of the corresponding command code. Command codes can range from 0x00 to 0x7F. Reply codes can range from 0x80 to 0xFF. Unassigned values are reserved. All of these commands are sent over the 1284.4 Transaction Channel.

Table 7--Summary of transactions

Transaction	Command	Reply	Purpose
Init	0x00	0x80	Establish a 1284.4 conversation.
OpenChannel	0x01	0x81	Open a channel between a primary and secondary socket and set the initial crediting mode.
CloseChannel	0x02	0x82	Close a channel between a primary and secondary socket.
Credit	0x03	0x83	Grant credit on a specific channel.
CreditRequest	0x04	0x84	Request credit for, and set the credit mode of, a specific channel.
	0x05	0x85	Deprecated – do not use [a]
	0x06	0x86	Deprecated – do not use [a]
	0x07	0x87	Deprecated – do not use [a]
Exit	0x08	0x88	Terminate the current 1284.4 conversation
GetSocketID	0x09	0x89	Request the socket ID of the specified service.
GetServiceName	0x0A	0x8A	Request the name of the service on the specified socket.
	0x0B-0x7E	0x8B-0xFF	Reserved
Error	0x7F		Indicates an error has occurred that cannot be reported in a 1284.4 reply.

[a] These values are used by Hewlett-Packard's MLC protocol and are deprecated for compatibility purposes.

The following table summarizes the possible result values for all of the 1284.4 transactions.

Table 8-Transaction result value summary

Value	Result
0x00	The command was successful.
0x01	Unable to begin 1284.4 conversation at this time. The initiator can retry at a later time.
0x02	Unable to support requested revision. The initiator can attempt to negotiate to a common revision.
0x03	The command was rejected because the requested channel was the 1284.4 transaction channel, which cannot be closed.
0x04	Sufficient resources to support the requested connection are not currently available. The channel is not open.
0x05	The connection has been denied. The channel is not open.
0x06	This channel is already open. The channel remains open, with credit unchanged by the **OpenChannel** transaction.
0x07	Credit overflow – the specified credit causes the total outstanding credit for this direction on this channel to exceed 0xFFFF. On a credit overflow failure the target shall keep its credit count as it was before the **Credit** command. The credit is ignored.
0x08	The command was rejected because this channel is not open.
0x09	There is no service on the specified socket. The channel is not open.
0x0A	Service name to socket ID mapping failed, either because the service was not available (**GetSocketID**) or there was no service on the specified socket (**GetServiceName**).
0x0B	The **Init** transaction can not be completed due to an outstanding **Init** transaction initiated by this 1284.4 peer. The initiator of this transaction should wait a random length of time and then try again to establish the conversation.
0x0C	One or both of the packet sizes requested was invalid. (0x0001-0x0005) The channel is not open.
0x0D	The packet sizes requested in both directions were 0x0000. No data can be transferred. The channel is not open.
0x0E	The command was rejected because the requested credit mode is not supported on the specified channel.

5.3 Conversation Control Transactions

A conversation is the set of interactions between two 1284.4 peers during the period of time bounded by the completion of corresponding **Init** and **Exit** transactions. This section describes the 1284.4 transactions that establish, manage and terminate the conversation.

5.3.1 Init

The **Init** transaction is used to establish a 1284.4 conversation. A conversation shall be established before any subsequent 1284.4 communications take place.

The initiator of the conversation sends an **Init** command with the requested 1284.4 revision. If the target supports the requested 1284.4 revision and can begin a 1284.4 conversation, the target shall send an **InitReply** with a matching revision and a success result value. If the target does not support the requested 1284.4 revision, the target shall send an **InitReply** with the highest revision it does support (less than that requested by the initiator), and a result value indicating that it is unable to support the requested revision.

When an **Init** transaction is completed with a result value indicating success, the 1284.4 conversation has been established at the revision level requested by the initiator. The initiator of the successful **Init** transaction becomes the primary device for that conversation. The target of the successful **Init** transaction becomes the secondary device for that conversation.

A random back-off strategy shall be implemented to successfully establish the 1284.4 conversation and to establish the primary and secondary devices if **Init** transactions are initiated by both 1284.4 peers at the same time. If an **Init** command is received by a 1284.4 peer that has initiated its own outstanding **Init** transaction, it shall send an **InitReply** with a failure result value as shown in Table 11. After the 1284.4 peer receives the **InitReply** completing its outstanding **Init** transaction, the peer shall then wait a random length of time and again initiate an **Init** transaction. This process continues until one of the **Init** transactions completes successfully. It is illustrated in Figure 1.

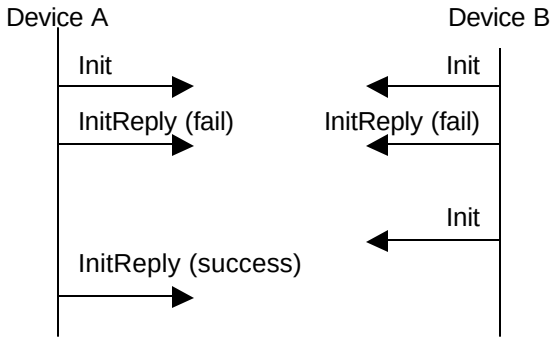

Figure 1 - Random back-off of simultaneous Init transactions

Init and **InitReply** may be sent at any time, even if there is no credit on the 1284.4 transaction channel. Sending the command during a 1284.4 conversation will cause that conversation to be reset and a new conversation to start. The reset implicitly terminates all pending transactions for the conversation, and closes all channels associated with the conversation.

Init and **InitReply** consume no credit. The initiator of an **Init** transaction may not receive an **InitReply**. In this situation, it is valid to send another **Init**, because **Init** consumes no credit. The piggyback credits received with **Init** and **InitReply** become the initial credit balance of the 1284.4

transaction channel. If more than one **Init** command is received, only the credit from the final **Init** is valid.

If there is no reply to the **Init** command, the initiator shall wait some time before initiating another **Init** command. This will avoid issuing a second **Init** command before the target has had enough time to reply to the first **Init** command. Since it is still possible that the target may reply to each command, any extra **InitReply** replies received by the primary 1284.4 peer before the replies to any other commands and before receiving any commands from the secondary 1284.4 peer shall be ignored.

Table 9 and Table 10 show the contents of the **Init** command and reply packets including the packet header.

Table 9--Init command

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0008
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x00
Revision [a]	0x07	0x20

[a] Indicates the revision of 1284.4 to which the initiator is attempting to negotiate. This provides some indication as to which commands and replies are recognized by the initiating 1284.4 peer. The initial and current revision of the 1284.4 protocol is 0x20. Future revisions shall be greater than 0x20. All implementations are required to support the base (0x20) revision of the 1284.4 protocol.

Table 10--InitReply packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0009
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x80
Result	0x07	(see Table 11)
Revision [a]	0x08	0x20

[a] When the Result field indicates success, Revision indicates the actual revision to be used, which shall match the Revision requested by the initiator. When the Result field indicates that the target can not support the Revision requested by the initiator, Revision indicates the highest revision of 1284.4, less than the requested revision, the target of the **Init** transaction implements.

Table 11--InitReply result values

Value	Result
0x00	The command was successful.
0x01	Unable to begin 1284.4 conversation at this time. The initiator can retry at a later time.

0x02	Unable to support requested revision. The initiator can attempt to negotiate to a common revision.
0x0B	The **Init** transaction can not be completed due to an outstanding **Init** transaction initiated by this 1284.4 peer. The initiator of this transaction should wait a random length of time and then try again to establish the conversation.

5.3.2 Exit

The **Exit** transaction is used to terminate the current 1284.4 conversation. Either 1284.4 peer can initiate an **Exit** transaction, regardless of which peer started the current conversation. To terminate normally, complete all outstanding 1284.4 transactions and close all open channels before initiating the **Exit** transaction.

If a 1284.4 peer determines that it is not possible to terminate normally, it is acceptable to initiate the **Exit** transaction without completing outstanding 1284.4 transactions or closing open channels. The **Exit** transaction causes all outstanding transactions to be aborted, all open channels to be closed, and the conversation to be terminated.

The initiator of the **Exit** transaction begins the transaction by sending an **Exit** command. No other 1284.4 transactions shall be initiated once the **Exit** transaction begins. Upon receiving the **Exit** command, the target of the transaction optionally completes any outstanding transactions, and then completes the **Exit** transaction by sending the **ExitReply**. It is not necessary for the initiator to wait for the **ExitReply**. The 1284.4 conversation is considered terminated after the completion of the **Exit** transaction.

If both 1284.4 peers initiate **Exit** transactions at the same time, both **Exit** transactions shall be completed. The 1284.4 conversation is considered terminated after the completion of both **Exit** transactions.

Each peer shall have no more than one **Exit** transaction outstanding at any time (see 4.5.5 - 1284.4 transaction channel flow control).

Data received prior to issuing or receiving an **Exit** command shall be delivered. Data received after issuing an Exit command may be delivered. Any packets received after receiving an **Exit** command shall be ignored, since the conversation has been terminated.

Table 12 and Table 13 show the contents of the **Exit** and **ExitReply** packets, including the packet header.

Table 12--Exit packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0007
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x08

Table 13--ExitReply packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0008
Credit	0x04	0x00
Control	0x05	0x00
Command	0x06	0x88

Result [a]	0x07	(See Table 14)

[a] Indicates the outcome of the **Exit** command. This field shall take on the value shown in Table 14.

Table 14--ExitReply result values

Value	Result
0x00	The command was successful.

5.3.3 Error

The **Error** packet is used to report errors that can not be reported in a 1284.4 transaction.

The **Error** packet does not have a reply. It may be sent at any time, even if there is no credit on the 1284.4 transaction channel. **Error** consumes no credit. Piggyback credit can not be granted via the **Error** packet.

Error packets do not implicitly close connections nor terminate the conversation.

Table 15 shows the contents of the **Error** packet, including the packet header.

Table 15--Error packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x000A
Credit	0x04	0x00
Control	0x05	0x00
Command	0x06	0x7F
ErrorPSID [a]	0x07	0x00-0xFF
ErrorSSID [b]	0x08	0x00-0xFF
ErrorCode [c]	0x09	(See Table 16)

[a] Specifies the PSID of the packet in which the error was detected.
[b] Specifies the SSID of the packet in which the error was detected.
[c] Specifies what type of error has occurred. This field can take on the values listed in Table 16.

Table 16--Error command ErrorCode values

ErrorCode	Error description
0x80	A malformed packet was received. All fields in the packet shall be ignored.
0x81	A packet was received for which no credit had been granted. The packet was ignored.
0x82	A 1284.4 reply was received that could not be matched to an outstanding command. The reply was ignored. Credit granted in the reply was ignored.
0x83	A packet of data was received that was larger than the negotiated maximum size for the socket indicated. The data was ignored
0x84	A data packet was received for a channel that was not open.
0x85	A reply packet with an unknown Result value was received.
0x86	Piggybacked credit received in a data packet caused a credit overflow for that channel.
0x87	A reserved or deprecated 1284.4 command or reply was received. Any piggybacked credit was ignored.

5.4 Connection Control Transactions

1284.4 connections consist of a channel specified by primary and secondary socket identifiers. This section describes the 1284.4 transactions that are used to discover endpoints, and establish, manage and terminate connections.

5.4.1 GetSocketID

This transaction is used to request the socket ID for a specified service.

The initiator of the **GetSocketID** transaction begins the transaction by sending a **GetSocketID** command with the specified service name. Upon receiving the **GetSocketID** command, the target of the transaction locates the specified service and sends the **GetSocketIDReply**. Requesting the socket ID of a zero-length service name shall result in an unsuccessful completion.

Once a service has been reported to be mapped to a particular socket ID, that mapping shall remain unchanged until the 1284.4 conversation is terminated. No other service shall be mapped to that particular socket ID. The mapped service may be unbound from that particular socket ID, but may not be mapped to any other socket ID. If it is again bound during the same conversation, it shall be mapped to the same socket ID.

Multiple **GetSocketID** transactions may be outstanding at any time, if credit to issue them is available on the transaction channel (see 4.5.5 - 1284.4 transaction channel flow control).

Service names are between 1 and 40 (0x28) bytes and are character restricted as per 4.4.2 - Service names. Service names are registered using the mechanism defined in Annex B - Service Names Registry.

Table 17 and Table 18 show the contents of the **GetSocketID** and **GetSocketIDReply** packets, including the packet header.

Table 17--GetSocketID command packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0007 + length of service name
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x09
ServiceName	0x07	(service name)

Table 18--GetSocketIDReply packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0009 + length of service name
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x89
Result [a]	0x07	(see Table 19)

SocketID [b]	0x08	0x00-0xFF
ServiceName [c]	0x09	(service name)

[a] Indicates the outcome of the **GetSocketID** transaction. This field can take on the values shown in Table 19.
[b] The socket ID of the specified service. This field is not valid if the specified service does not exist.
[c] The service name as specified in the **GetSocketID** command packet.

Table 19--GetSocketIDReply result values

Value	Result
0x00	The command was successful.
0x0A	Service name to socket ID mapping failed, either because the service was not available (**GetSocketID**) or there was no service on the specified socket (**GetServiceName**).

5.4.2 GetServiceName

This transaction is used to request the name of the service on the specified socket.

The initiator of the **GetServiceName** transaction begins the transaction by sending a **GetServiceName** command with the specified socket ID. Upon receiving the **GetServiceName** command, the target of the transaction identifies the service on the specified socket and sends the **GetServiceNameReply**.

Once a service has been reported to be mapped to a particular socket ID, that mapping shall remain unchanged until the 1284.4 conversation is terminated. No other service shall be mapped to that particular socket ID. The mapped service may be unbound from that particular socket ID, but may not be mapped to any other socket ID. If it is again bound during the same conversation, it shall be mapped to the same socket ID.

Multiple **GetServiceName** transactions may be outstanding at any time, if credit to issue them is available on the transaction channel (see 4.5.6 - Multiple Outstanding Transactions).

Service names are between 1 and 40 (0x28) bytes and are character restricted as per 4.4.2 - Service names. Service names are registered using the mechanism defined in Annex B - Service Names Registry.

Table 20 and Table 21 show the contents of the **GetServiceName** and **GetServiceNameReply** packets, including the packet header.

Table 20--GetServiceName command packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0008
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x0A
SocketID [a]	0x07	0x00-0xFF

[a] The socket ID of the requested service.

Table 21--GetServiceNameReply packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0009 + length of service name
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x8A
Result [a]	0x07	(see Table 22)
SocketID [b]	0x08	0x00-0xFF
ServiceName [c]	0x09	(service name)

[a] Indicates the outcome of the **GetServiceName** transaction. This field can take on the values shown in Table 22.
[b] The socket ID as specified in the **GetServiceName** command packet.

[c] A successful completion with a zero-length service name indicates an unnamed service on this socket. This field is not valid if there is no service mapped to this socket.

Table 22--GetServiceNameReply result values

Value	Result
0x00	The command was successful.
0x0A	Service name to socket ID mapping failed, either because the service was not available (**GetSocketID**) or there was no service on the specified socket (**GetServiceName**).

5.4.3 OpenChannel

This transaction is used to open and configure a channel between two 1284.4 endpoints.

The initiator of the transaction starts by sending an **OpenChannel** command with the socket IDs specifying the desired channel. Upon receiving the **OpenChannel** command, the target of the transaction checks the validity of the command, allocates the necessary resources for the channel, and sends the **OpenChannelReply**. The target shall choose valid maximum packet sizes for the channel, less than or equal to the maximum packet sizes requested by the initiator. A maximum packet size of 6 indicates data will not be sent in that direction, although data packets without a payload may still be sent to carry piggyback credit (see 4.5.3 - Credit). A maximum packet size of zero (0) indicates that packets will not be sent in that direction. An **OpenChannel** command with maximum packet sizes of zero (0) or 6 in both directions is not supported as it results in a channel on which no data can be transferred. The channel is considered to be open only after the completion of a successful **OpenChannel** transaction.

It is possible for both 1284.4 peers to initiate **OpenChannel** transactions for the same channel simultaneously. This does not cause an error. If the initiator of an **OpenChannel** transaction receives an **OpenChannel** command for the same channel before receiving the **OpenChannelReply** completing the **OpenChannel** transaction it initiated, it shall send a successful **OpenChannelReply** with:

1. Packet sizes equal to the minimum requested in the two commands and
2. The same value for MaximumOutstandingCredit provided in it's **OpenChannel** command.

The channel is considered to be open after the initiator has received the **OpenChannelReply** completing its transaction and sent the **OpenChannelReply** completing the peers transaction. The following table is an example of simultaneous **OpenChannel** transactions. Only the sending of packets is shown. The reception of packets is implied.

Time	Primary 1284.4 peer	Secondary 1284.4 peer
0		
1	**OpenChannel** • Primary SocketID = 0x80 • Secondary SocketID = 0x01 • MaximumPrimaryToSecondaryPacketSize = 0x8000 • MaximumSecondarytoPrimaryPacketSize = 0x0400 • MaximumOutstandingCredit = 0xFFFF	**OpenChannel** • Primary SocketID = 0x80 • Secondary SocketID = 0x01 • MaximumPrimaryToSecondaryPacketSize = 0x4000 • MaximumSecondarytoPrimaryPacketSize = 0x1000 • MaximumOutstandingCredit = 0x0001
2	**OpenChannelReply** • Result = 0x00 • Primary SocketID = 0x80 • Secondary SocketID = 0x01 • MaximumPrimaryToSecondaryPacketSize = 0x4000 • MaximumSecondarytoPrimaryPacketSize = 0x0400 • MaximumOutstandingCredit = 0xFFFF	
3		**OpenChannelReply** • Result = 0x00 • Primary SocketID = 0x80 • Secondary SocketID = 0x01 • MaximumPrimaryToSecondaryPacketSize = 0x4000 • MaximumSecondarytoPrimaryPacketSize = 0x0400 • MaximumOutstandingCredit = 0x0001

Since the 1284.4 transaction channel is always open, it does not require an **OpenChannel** transaction. An **OpenChannel** transaction for the transaction channel shall be handled the same as any other **OpenChannel** transaction received for a channel that is already open.

Multiple **OpenChannel** transactions may be outstanding at any time, if each transaction refers to a different channel and credit to issue them is available on the transaction channel (see 4.5.6 - Multiple Outstanding Transactions).

Table 23 and Table 24 show the contents of the **OpenChannel** and **OpenChannelReply** packets, including the packet header.

Table 23--OpenChannel command packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x000F
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x01
SocketID (primary) [a]	0x07	0x01-0xFF
SocketID (secondary) [b]	0x08	0x01-0xFF
MaximumPrimaryToSecondaryPacketSize [c]	0x09	0x0000 or 0x0006-0xFFFF
MaximumSecondaryToPrimaryPacketSize [d]	0x0B	0x0000 or 0x0006-0xFFFF
MaximumOutstandingCredit [e]	0x0D	0x0000-0xFFFF

[a] Specifies the socket of the requested channel on the primary 1284.4 peer.

[b] Specifies the socket of the requested channel on the secondary 1284.4 peer.
[c] Indicates the maximum size of data packets in the primary-to-secondary direction. A value of 0x0000 indicates primary-to-secondary data packets are not supported.
[d] Indicates the maximum size of data packets in the secondary-to-primary direction. A value of 0x0000 indicates secondary-to-primary data packets are not supported.
[e] Indicates the maximum outstanding credit requested by the initiator of this transaction for initiator to target data. See 4.5.3 - Credit for details on credit.

Table 24--OpenChannelReply packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0012
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x81
Result [a]	0x07	(see Table 25)
SocketID (primary) [b]	0x08	0x01-0xFF
SocketID (secondary) [c]	0x09	0x01-0xFF
MaximumPrimaryToSecondaryPacketSize [d]	0x0A	0x0000 or 0x0006-0xFFFF
MaximumSecondaryToPrimaryPacketSize [e]	0x0C	0x0000 or 0x0006-0xFFFF
MaximumOutstandingCredit [f]	0x0E	0x0000-0xFFFF
Credit [g]	0x10	0x0000-0xFFFF

[a] Indicates the outcome of the **OpenChannel** transaction. This field can take on the values shown in Table 25.
[b] Specifies the socket of the requested channel on the primary 1284.4 peer.
[c] Specifies the socket of the requested channel on the secondary 1284.4 peer.
[d] Indicates the maximum packet size that will be used to send data from the primary to the secondary 1284.4 peer. A value of 0x0000 indicates primary-to-secondary data packets for this channel will not be used. If the result value indicates an error, this field has no meaning.
[e] Indicates the maximum packet size that will be used to send data from the secondary to the primary 1284.4 peer. A value of 0x0000 indicates secondary-to-primary data packets for this channel will not be used. If the result value indicates an error, this field has no meaning.
[f] Indicates the maximum outstanding credit requested by the target of this transaction for target to initiator data. See 4.5.3 - Credit for details on credit.
[g] Indicates initial credit that the target of this command is granting the initiator. See 4.5.3 - Credit for details on credit.

Table 25--OpenChannelReply result values

Value	Result
0x00	The command was successful.
0x04	Sufficient resources to support the requested connection are not currently available. The channel is not open.
0x05	The connection has been denied. The channel is not open.
0x06	This channel is already open. The channel remains open, with credit unchanged by the **OpenChannel** transaction.
0x09	There is no service on the specified socket. The channel is not open.
0x0C	One or both of the packet sizes requested was invalid. (0x0001-0x0005) The channel is not open.

0x0D	The packet sizes requested in both directions were 0x0000. No data can be transferred. The channel is not open.

5.4.4 Credit

The **Credit** transaction is part of the 1284.4 flow control mechanism. This transaction is used to grant credit for a specific channel to the sending endpoint. See 4.5 - Data transfer and flow control for more details on flow control.

Multiple **Credit** transactions may be outstanding at any time, if each transaction refers to a different channel and credit to issue them is available on the transaction channel (see 4.5.6 - Multiple Outstanding Transactions).

Table 26 and Table 27 show the contents of the **Credit** and **CreditReply** packets including the packet header.

Table 26--Credit command packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x000B
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x03
SocketID (Primary) [a]	0x07	0x00-0xFF
SocketID (Secondary) [b]	0x08	0x00-0xFF
AdditionalCredit [c]	0x09	0x0000-0xFFFF

[a] Specifies the socket on the primary 1284.4 peer of the channel for which credits are being issued.
[b] Specifies the socket on the secondary 1284.4 peer of the channel for which credits are being issued.
[c] Indicates additional credit that the sender of this command is granting to the recipient of the command.

Table 27--CreditReply packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x000A
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x83
Result [a]	0x07	(See Table 28)
SocketID (primary) [b]	0x08	0x00-0xFF
SocketID (secondary) [c]	0x09	0x00-0xFF

[a] Indicates the outcome of the **Credit** command. This field can take on the values shown in Table 28.
[b] Specifies the socket on the primary 1284.4 peer of the channel for which credits are being issued.
[c] Specifies the socket on the secondary 1284.4 peer of the channel for which credits are being issued.

Table 28--CreditReply result values

Value	Result
0x00	The command was successful.
0x07	Credit overflow – the specified credit causes the total outstanding credit for this direction on this channel to exceed 0xFFFF. On a credit overflow failure the target shall keep its credit count as it was before the **Credit** command. The credit is ignored.
0x08	The command was rejected because this channel is not open.

5.4.5 CreditRequest

The **CreditRequest** transaction is used to request credit from the receiving endpoint. The transaction is also used to modify the MaximumOutstandingCredit parameter (see 4.5.4 - 1284.4 data flow control). If the credit mode is set to "unlimited" or "maintain credit" modes, the target of the **CreditRequest** transaction shall grant some or all of the requested credit as quickly as possible, either through the **CreditRequestReply** or a subsequent **Credit** transaction. Credit may be requested at any time.

Multiple **CreditRequest** transactions may be outstanding at any time, if each transaction refers to a different channel and credit to issue them is available on the transaction channel (see 4.5.6 - Multiple Outstanding Transactions).

A MaximumOutstandingCredit value of zero (0) is not supported on the 1284.4 Transaction Channel.

Table 29 and Table 30 summarize the contents of the **CreditRequest** and **CreditRequestReply** packets, including the packet header.

Table 29--CreditRequest command packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x000B
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x04
SocketID (primary) [a]	0x07	0x00-0xFF
SocketID (secondary) [b]	0x08	0x00-0xFF
MaximumOutstandingCredit [c]	0x09	0x0000-0xFFFF

[a] Specifies the socket on the primary 1284.4 peer of the channel for which credit is being requested.
[b] Specifies the socket on the secondary 1284.4 peer of the channel for which credit is being requested.
[c] Indicates the maximum outstanding credit requested by the sender of this command. See 4.5.3 - Credit for details on credit.

Table 30--CreditRequestReply packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x000C
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x84
Result [a]	0x07	(See Table 31)
SocketID (primary) [b]	0x08	0x00-0xFF
SocketID (secondary) [c]	0x09	0x00-0xFF
AdditionalCredit [d]	0x0A	0x0000-0xFFFF

[a] Indicates the outcome of the **CreditRequest** command. This field can take on the value shown in Table 31.
[b] Specifies the socket on the primary 1284.4 peer of the channel for which credit is being requested.
[c] Specifies the socket on the secondary 1284.4 peer of the channel for which credit is being requested.
[d] Indicates additional credit that the sender of this reply is granting to the sender of the **CreditRequest** command. A value of 0x0000 is a valid credit field value and indicates no additional credit is being granted.

Table 31--CreditRequestReply result values

Value	Result
0x00	The command was successful.
0x08	The command was rejected because this channel is not open.
0x0E	The command was rejected because the requested credit mode is not supported on the specified channel.

5.4.6 CloseChannel

This transaction is used to close a channel between two 1284.4 endpoints. Either 1284.4 peer can initiate a **CloseChannel** transaction, regardless of which peer initiated the **OpenChannel** transaction for the channel.

The initiator of the **CloseChannel** transaction begins the transaction by sending a **CloseChannel** command with the socket IDs of the channel it desires to close. Upon receiving the **CloseChannel** command, the target of the transaction waits for any outstanding transactions for this channel to complete, and then sends the **CloseChannelReply**. Once the target of the transaction has sent the **CloseChannelReply**, it may de-allocate the resources for the channel. The channel is considered to be closed only after the completion of a successful **CloseChannel** transaction. Once the **CloseChannel** transaction starts, no new 1284.4 transactions may be initiated for the specified channel, but all 1284.4 transactions for that channel must be completed before the **CloseChannel** transaction completes.

Prior to issuing a **CloseChannel** command, all data received from the peer device by the 1284.4 transport shall be delivered to the receiving endpoint. Prior to responding to a **CloseChannel** command, all data received from the peer device by the 1284.4 transport shall be delivered to the receiving endpoint if it is possible to do so without delaying the response to the **CloseChannel** command.

If both 1284.4 peers initiate **CloseChannel** transactions for the same channel at the same time, both **CloseChannel** transactions shall complete.

Since the 1284.4 transaction channel is needed for the 1284.4 conversation, it can not be closed.

Multiple **CloseChannel** transactions may be outstanding at any time, if each transaction refers to a different channel and credit to issue them is available on the transaction channel (see 4.5.6 - Multiple Outstanding Transactions).

Table 32 and Table 33 show the contents of the **CloseChannel** and **CloseChannelReply** packets, including the packet Header.

Table 32--CloseChannel command packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x0009
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x02
SocketID (primary) [a]	0x07	0x01-0xFF
SocketID (secondary) [b]	0x08	0x01-0xFF

[a] Specifies the socket on the primary 1284.4 peer of the channel that is being closed.
[b] Specifies the socket on the secondary 1284.4 peer of the channel that is being closed.

Table 33--CloseChannelReply packet

Packet field	Packet offset	Field contents
PSID	0x00	0x00
SSID	0x01	0x00
Length	0x02	0x000A
Credit	0x04	0x01
Control	0x05	0x00
Command	0x06	0x82
Result [a]	0x07	(See Table 34)
SocketID (primary) [b]	0x08	0x01-0xFF
SocketID (secondary) [c]	0x09	0x01-0xFF

[a] Indicates the outcome of the **OpenChannel** transaction. This field can take on the values shown in Table 34.
[b] Specifies the socket on the primary 1284.4 peer of the channel that is being closed.
[c] Specifies the socket on the secondary 1284.4 peer of the channel that is being closed.

Table 34--CloseChannelReply result values

Value	Result
0x00	The command was successful.
0x03	The command was rejected because the requested channel was the 1284.4 transaction channel, which cannot be closed.
0x08	The command was rejected because this channel is not open.

6. Data link service requirements

6.1 Overview

The following section describes the data link services required by 1284.4. This specification describes the requirements but not the implementation of these services. 1284.4 places no other requirements on the data link. IEEE P1284.3/D6.00 (December 3, 1998) is an example of a data link that meets all of these requirements.

6.2 Required Services

6.2.1 Register/unregister

A 1284.4 implementation shall be able to register itself with the data link. The 1284.4 implementation will identify itself as being the processor for the 1284.4 protocol by using the 1284.4 protocol ID. Only one instance of the 1284.4 protocol is permitted to be registered with a particular data link at any time. The 1284.4 implementation shall be able to unregister itself to allow another 1284.4 implementation to register itself as the 1284.4 processor for the data link.

The following is an example of register and unregister services. In the example, a request is a call from a data link client to the data link and an indication is a call from the data link to one of its clients.

DL_Register.request ProtocolID, AsyncBlk, Handle

ProtocolID
The identification number of the protocol served by this client of the data link. Only one 1284.4 client is allowed. The protocol ID for the 1284.4 protocol is registered by the Internet Assigned Number Authority and published in RFC 1700 - PPP Assigned Numbers List. Its value is 0x0285.

AsyncBlk
A block of memory that contains whatever information is needed to deliver data link indications to the 1284.4 transport. The delivery mechanism is implementation specific.

Handle
The address of a container to hold the client's handle. This handle is used to identify the 1284.4 transport in future data link requests.

DL_Unregister.request Handle

Handle
The client's handle as returned in DL_Register.request.

6.2.2 Packet Data Transfer

The 1284.4 implementation shall be able to send packets of data to the 1284.4 peer. The data shall be delivered to the 1284.4 peer, and only to the 1284.4 peer. It is assumed to be error free and in the same order as it was presented to the data link. Data packet boundaries shall be maintained by the data link. If no 1284.4 peer exists, the receiving data link shall discard the data. Data from other data link clients shall not be delivered to either 1284.4 peer. The data link shall accept a destination device address with all data to be sent and return the source device address with all

delivered data. The following is an example of data transfer services. A 1284.4 implementation uses the request to send data. The peer data link uses the indication to deliver the data to the 1284.4 peer.

DL_Data.request Handle, DeviceAddress, Data

Handle
The client's handle as returned in the DL_Register.request.

DeviceAddress
Operating system dependent address of destination device for this data.

Data
The packet of data to be delivered.

DL_Data.indication DeviceAddress, Data

Data
The packet of data passed to the peer data link through DL_Data.request. The packet of data shall be in the same order and form as it was presented to the remote data link in the DL_Data.request.

DeviceAddress
Operating system dependent address of source device for this data. Can be used in DL_Data.request to deliver data back to the 1284.4 peer.

6.2.3 Asynchronous operation

The 1284.4 implementation shall be able to operate asynchronously. It shall not at any time be required to poll the data link.

Annex A
Bibliography

(informative)

[1] IEEE Std. 1284-1994, Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers. This standard defines a signaling method for asynchronous, half-duplex, fully interlocked, bi-directional parallel communications between hosts and printers or other peripheral devices.
[2] MLC Packet Protocol Revision 3.8, Hewlett-Packard Company. More information regarding this document can be obtained from *stds-1284@ieee.org*.
[3] IEEE Draft Std. P1284.3/D6.00 (December 3, 1998), Standard for Interface and Protocol Extensions to IEEE Std. 1284-1994 Compliant Peripheral and Host Adapter Ports. This standard extends the IEEE Std. 1284-1994 to provide port sharing and data link functionality.
[4] IEEE Std. 1284.1-1997, Standard for Transport Independent Printer/System Interface
[5] IANA Protocol and Service Names List, available both on their web-site at: *http://www.iana.org/* and on their FTP site at: *ftp://ftp.isi.edu/in-notes/iana/assignments/service-names*
[6] Winsock-2 API available at http://www.stardust.com.

Annex B
Service Names Registry

(normative)

B.1 Purpose and scope

This annex is provided to explain how 1284.4 implementers can register service names to provide public knowledge of services that are available and prevent different services from erroneously using the same name. IANA (Internet Assigned Numbers Authority) maintains a list of service names on the list titled "Protocol and Service Names" available both on their web-site at: *http://www.iana.org/* and on their FTP site at: *ftp://ftp.isi.edu/in-notes/iana/assignments/service-names*. Implementers should only register service names as needed; different names for the same service should be avoided. The registry provides a method to avoid service name collisions.

B.2 service-names list

The service-names list contains an alphabetical listing of each service name following the format as described in the service-names file:

> *A protocol or service may be up to 40 characters taken from the set of uppercase letters, digits, and the punctuation character hyphen. It must start with a letter, and end with a letter or digit.*

Each service name is followed by a brief description of its use. Examples of protocols and services include:

Service Name	Service Description
BOOTP	Bootstrap Protocol
BOOTPC	Bootstrap Protocol Client
BOOTPS	Bootstrap Protocol Server
BR-SAT-MON	Backroom SATNET Monitoring
CFTP	CFTP
CHAOS	CHAOS Protocol
CHARGEN	Character Generator Protocol
IEEE-1284-4-TRANSACTION	IEEE 1284.4 Transaction Processor (Socket 0x00)

B.3 Registry process

Implementers of 1284.4 are encouraged to follow the following process to register their service name with IANA.

1. Check the service-names file available at *http://www.iana.org/ and ftp://ftp.isi.edu/in-notes/iana/assignments/service-names* to verify that the proposed service name has not already been registered.

2. Send e-mail using the template below to *iana@iana.org* and *stds-1284@ieee.org* requesting the new service-name. The IANA web-site may provide updates to the template.

Service name registry request template:

Subject: Protocol and Service Names Registry Request

Please add the following service name and description to the Protocol and Service Names list.

Service-name (40 characters maximum, letters/digits/hyphen only): <service-name>
Description (brief description, 1 sentence recommended) : <description>

Contact Name (person to be contacted in case of any problems): <Name>
Contact Name e-mail: <e-mail address>
Contact Name Company: <company name>

An example:
Subject: Protocol and Service Names Registry Request

Please add the following service name and description to the Protocol and Service Names list.

Service-name: XYZ-INPUT
Description: Input data stream for an XYZ

Contact Name: John Doe
Contact Name e-mail: john@anycompany.com
Contact Name Company: AnyCompany, Inc.

IANA will update the list as requests are approved. The 1284.4 registry coordinator will intervene only if IANA has a problem completing a request. Questions regarding the registry process can be directed by e-mail to the coordinator at *stds-1284@ieee.org*.

Annex C
1284.4 Architecture
(informative)

The following figure illustrates an example of the protocol architecture of several devices using 1284.4 and other related protocols. The top of the figure shows devices on various physical links. The bottom of the figure shows applications and drivers in a host. The center of the figure shows the various protocols required to connect the applications and drivers to the devices.

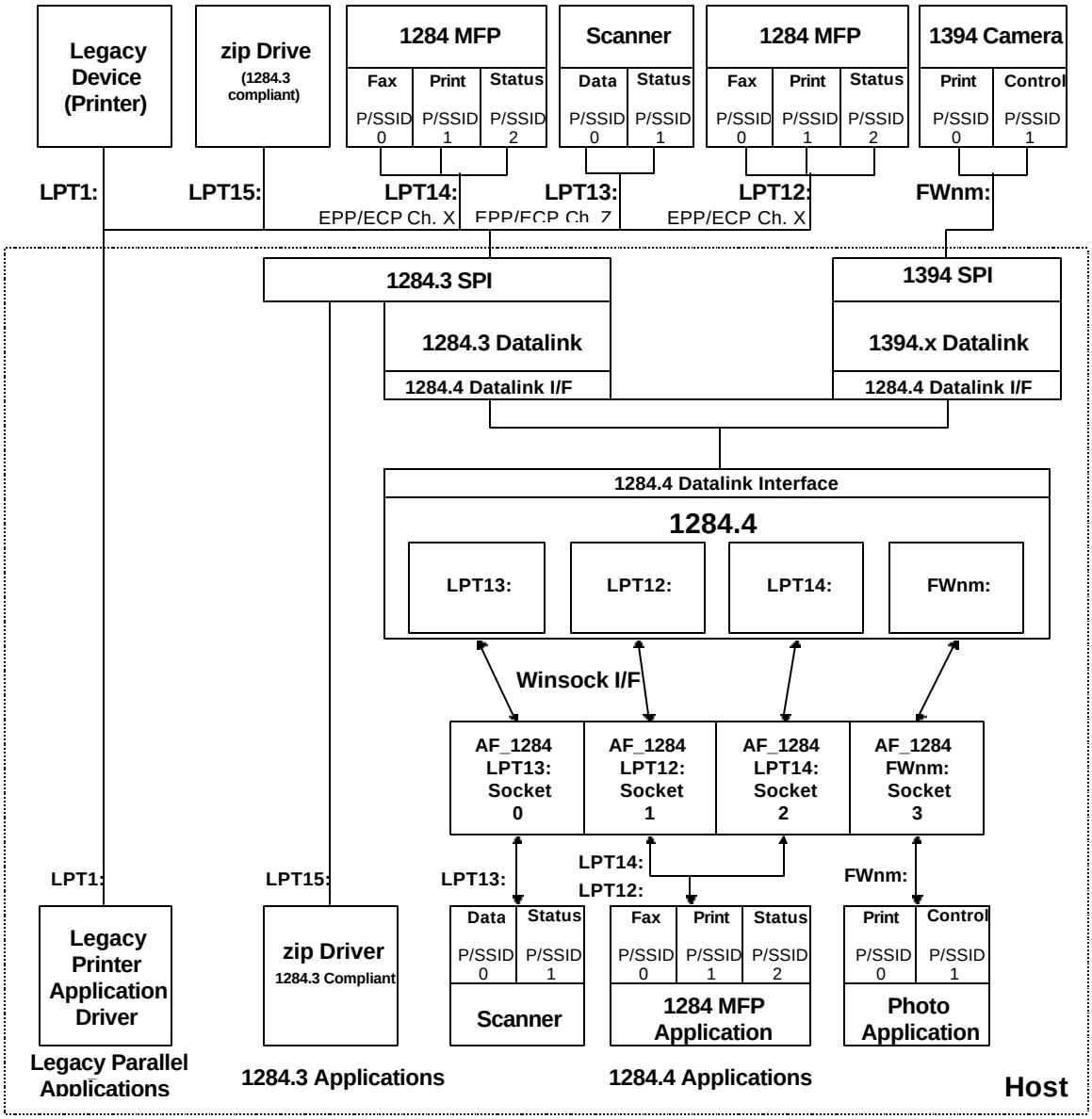

Figure C.1 - 1284.4 Device Architecture

Annex D
Example 1284.4 API

(informative)

D.1 API Overview

This example Application Programming Interface (API) is sockets-based interface for use by clients and servers to access the services provided by 1284.4. The API is modeled after the Winsock-2 interface available at http://www.stardust.com.

1284.4 sockets and API sockets are not necessarily equivalent. 1284.4 sockets refer to an endpoint to which or from which data is transmitted. API sockets refer not only to an endpoint, but also to a particular connection to that endpoint. A 1284.4 connection is specified by a pair of 1284.4 socket identifiers, known as the primary and secondary socket IDs. A single API socket uniquely identifies an API connection. More than one API socket can be mapped to the same local 1284.4 socket by the 1284.4 implementation, as long as each API socket mapped to the same local 1284.4 socket specifies a connection to a different remote 1284.4 socket.

See Figure C.1 for a sample mapping of Winsock-2 sockets to 1284.4 connections.

D.2 API Services

In the following entry point definitions, IN refers to parameters passed to the API and OUT refers to parameters passed from the API.

D.2.1 Socket

Socket establishes a new endpoint. The 1284.4 layer manages the allocation of 1284.4 sockets. Neither clients nor servers need be aware of the management of 1284.4 sockets.

***socket* = Socket (**
IN ***address_family***
IN ***socket_type*)**

socket
A new socket allocated by the 1284.4 transport layer.

address_family
An address family specification. The address family for 1284.4 is "AF_12844".

socket_type
A type specification for the new socket.

The following *socket_type* is supported by 1284.4:

- SOCK_STREAM: Provides connection-based byte streams with an out-of-band data transmission mechanism. The transport is free to divide or aggregate client buffers. In-band data shall not be combined with out-of-band data. End-of-message data shall not be combined with non end-of-message data.

Since the transports will not exchange the socket_type at the time the connection is opened, the transport clients must set the same socket_type for the sockets at both ends of the connection.

D.2.2 Bind

Bind assigns a name to a socket. A server uses bind to register its name so that it can be found through service discovery. Bind will fail if *service_name* is already bound to another socket.

Bind (
IN ***socket*****,**
IN ***service_name*****)**

socket
A socket previously allocated by a call to socket().

service_name
The name by which the server wishes to be known. The name must follow the restrictions in 4.4.2 - Service names or must be an explicit 1284.4 socket number. The format of the explicit 1284.4 socket number is operating system dependent.

D.2.3 Listen

Listen allocates space for queuing incoming connection requests.

Listen (
IN ***socket*****)**

socket
A socket previously allocated by a call to socket() and assigned a name or explicit socket number by a call to bind().

D.2.4 Accept

Accept accepts new connections on a socket. When a connect request is received for that socket from the remote transport, the 1284.4 transport layer allocates a new socket for that connection and the call to accept() completes.

new_socket **= Accept (**
IN ***socket*****)**

new_socket
A new socket allocated for this connection by the 1284.4 transport layer.

socket
A socket previously allocated by a call to socket(), assigned a name or explicit 1284.4 socket number by a call to bind(), and placed in a listen state by a call to listen().

D.2.5 Connect

Connect opens a connection to a server.

Connect (

IN ***socket*****,**
IN ***device_address*****,**
IN ***service_name*****)**

socket
A socket previously allocated by a call to socket().

device_address
Operating system dependent address of the device to which the connection is to be opened. The 1284.4 implementation may not interpret the device address. It may instead be passed to the appropriate data link layer for interpretation.

service_name
Name or explicit 1284.4 socket number of the service to which the connection is to be opened.

D.2.6 Send

Send data over an open connection.

Send (
IN ***socket*****,**
IN ***data*****,**
IN ***out-of-band_flag*****,**
IN ***end-of-message_flag*****)**

socket
A socket to a connection opened by a call to accept() or connect().

data
Buffer of data to be sent over the connection.

out-of-band_flag
Flag indicating that the data is out-of-band. 1284.4 shall process out-of-band data in the same manner as in-band data. In-band data shall not be combined with out-of-band data. The receiver of out-of-band data shall be notified that the data is out-of-band.

end-of-message_flag
Flag indicating that the data is the end of a message. 1284.4 shall process end-of-message data in the same manner as non end-of-message data. End-of-message data shall not be combined with non end-of-message data. The receiver of end-of-message data shall be notified that the data is the end of a message.

D.2.7 Receive

Receive data over an open connection. The receive call completes when data is received or when the connection is terminated by the remote client.

Receive (
IN ***socket*****,**
OUT ***data*****,**

OUT ***out-of-band_flag*,**
OUT ***end-of-message_flag*)**

socket
A socket to a connection opened by a call to accept() or connect().

data
Buffer for data to be received over the connection.

out-of-band_flag
Flag indicating that the data is out-of-band.

end-of-message_flag
Flag indicating that the data is the end of a message.

D.2.8 Shutdown

Disable sends and/or receives on a socket

Shutdown (
IN ***socket*,**
IN ***direction*)**

socket
A socket to a connection opened by a call to accept() or connect().

direction
The direction(s) to be disabled. Can be SEND, RECEIVE, or BOTH.

Shutdown() does not close the socket. Resources allocated to the socket will remain reserved until the socket is closed using CloseSocket().

D.2.9 CloseSocket

CloseSocket releases a socket. If the socket has an open connection, the connection is closed before the socket is released.

CloseSocket (
IN ***socket*)**

socket
A socket previously allocated by a call to socket() or accept().

D.2.10 SetSockOpt

SetSockOpt sets socket options. Since the client is not required to call SetSockOpt, the implementation shall establish reasonable default values for each of the socket options.

SetSockOpt (
IN ***socket*,**

IN ***option_name*,**
IN ***option_value*)**

socket
A socket previously allocated by a call to socket() or accept().

option_name
The socket option for which the value is to be set.

option_value
The buffer in which the value for the requested option is supplied.

Option_name	Description
outgoing packet length	If there is no open connection to this socket, sets the maximum outgoing packet length 1284.4 shall attempt to establish when opening a connection to this socket. If there is an open connection to this socket, the maximum outgoing packet length may not be changed.
incoming packet length	If there is no open connection to this socket, sets the maximum incoming packet length 1284.4 shall attempt to establish when opening a connection to this socket. If there is an open connection to this socket, the maximum incoming packet length may not be changed.
maximum outstanding credit	If there is no open connection to this socket, sets the MaximumOutstandingCredit value 1284.4 shall request when opening a connection to this socket (see 4.5.4 - 1284.4 data flow control). If there is an open connection to this socket, 1284.4 shall use **CreditRequest** to request a change to the actual MaximumOutstandingCredit value for the open connection.

D.2.11 GetSockOpt

GetSockOpt returns the current socket options.

GetSockOpt (
IN ***socket*,**
IN ***option_name*,**
OUT ***option_value*)**

socket
A socket previously allocated by a call to socket() or accept().

option_name
The socket option for which the value is to be retrieved.

option_value
The buffer in which the value for the requested option is to be returned.

Option_name	Description

maximum outgoing packet length	If there is no open connection to this socket, returns the maximum outgoing packet length 1284.4 shall attempt to establish when opening a connection to this socket. If there is an open connection to this socket, returns the maximum outgoing packet length 1284.4 established when opening the connection.
maximum incoming packet length	If there is no open connection to this socket, returns the maximum incoming packet length 1284.4 shall attempt to establish when opening a connection to this socket. If there is an open connection to this socket, returns the maximum incoming packet length 1284.4 established when opening the connection.
maximum outstanding credit	If there is no open connection to this socket, returns the MaximumOutstandingCredit value 1284.4 shall request when opening a connection to this socket (see 4.5.4 – 1284.4 data flow control). If there is an open connection to this socket, returns the current MaximumOutstandingCredit value for the open connection.

D.3 API Usage

D.3.1 Servers

Servers first establish a socket by calling Socket() and then bind their service name to that socket by calling Bind(). Once the socket has been allocated, servers call Listen() to establish an incoming request queue, and then call Accept() to wait for a connection request. When a client requests a connection, Accept() will complete with a new socket for the now open connection. If the server can accept more concurrent connections, it may call Listen() again with the original socket. When the server is finished using a socket, it releases the socket by calling Shutdown() followed by CloseSocket().

Once a connection is open, servers can use Send() and Receive() to transfer data across the open connection.

D.3.2 Clients

Clients establish a socket by calling Socket(); they need not bind a name to that socket. Once the socket has been allocated, the client can request a connection by calling Connect(). When Connect() completes, the connection is open. When the client is finished using a socket, it releases the socket by calling Shutdown() followed by CloseSocket().

Once a connection is open, clients can use Send() and Receive() to transfer data across the open connection.

D.4 Mapping API services to 1284.4 transactions

This section describes a sample mapping of API services to 1284.4 transactions. Other mappings are possible and implementation-dependent.

D.4.1 Socket ()

No 1284.4 transactions required. An available, local 1284.4 socket is mapped to this API socket.

D.4.2 Bind ()

No 1284.4 transactions required.

D.4.3 Listen ()

No 1284.4 transactions required.

D.4.4 Accept ()

No 1284.4 transactions required.

D.4.5 Connect ()

Init, if there is no current 1284.4 conversation with the device.
GetSocketID, to map the service name to a 1284.4 socket ID.
OpenChannel, to open a channel to the server.

D.4.6 Send ()

No 1284.4 transactions are required. The data is sent when there is credit to do so.

D.4.7 Receive ()

Credit, to grant credit for available buffers, if it has not yet been granted.

D.4.8 Shutdown ()

No 1284.4 transactions required.

D.4.9 CloseSocket ()

CloseChannel, if channel is open.
Exit, if all channels are closed and the 1284.4 implementation chooses to terminate the conversation.

D.4.10 SetSockOpt ()

CreditRequest, to change the MaximumOutstandingCredit parameter if requested.

D.4.11 GetSockOpt ()

No 1284.4 transactions required.

Annex E
Implementation Issues

(informative)

E.1 Example of simple device

A simple 1284.4 printing device might implement a physical layer (e.g. IEEE 1284), a data link layer (e.g. IEEE 1284.3), 1284.4, and two services (e.g. a page description language (PDL) and a device management language (DML)). 1284.4 would be the only client of the data link layer. The PDL could be registered on socket 0x01 and the DML could be registered on socket 0x02. The PDL service could support one concurrent connection and the DML could support two or three concurrent connections. A host printing application could open a connection to the PDL, send a print job using that connection, and then close the connection. It might also open a connection to the DML to control the device. Another printing application would have to wait until the first application had closed the PDL connection before it could open its own connection. There might also be a management application using an open connection to the DML to monitor device status.

E.2 Data transfer efficiency

The efficiency of data transfer using 1284.4 is mainly determined by several factors:

1) protocol overhead
2) crediting overhead

Protocol overhead refers to the extra bytes required for 1284.4 headers in the byte stream. 1284.4 headers are 6 bytes long, so each data packet carries 6 bytes of overhead, regardless of the amount of data in the packet. If the amount of data to be transferred is n bytes, the number of packets is n/(amount of data per packet). Therefore the total protocol overhead is (6*n)/(amount of data per packet). Using larger packets reduces the total protocol overhead. Implementers should consider this when choosing packet sizes.

Credit transactions or zero-payload data packets are required for flow control on channels with data flowing in only one-direction (channels with data flowing in both directions can piggyback credit with their data, which adds no overhead). Each **Credit** transaction requires the transfer of 21 bytes. Each zero-payload data packet requires the transfer of 6 bytes.

The frequency of crediting depends upon the flow control algorithm being used and by the amount of buffer space in the receiving device. Flow control algorithms that cause less frequent crediting and increased buffer space reduce the crediting overhead. Implementers should consider this when designing crediting algorithms and when allocating buffer space. Crediting also requires "turning around" half-duplex links (e.g. IEEE 1284), which also introduces overhead. Implementers should also consider this overhead.

E.3 Flow control algorithms

1284.4 does not specify a required algorithm for flow control. The algorithms for allocating buffers to channels and crediting those buffers to the peer (flow control) are up to the implementer. The flow control algorithm can be as simple as issuing credit whenever half of the buffers in the channel's buffer pool are available and uncredited. This algorithm should keep both the sending client and receiving server operating in parallel. More complex algorithms that adjust to data arrival rate, the number of open channels and the availability of resources are also possible.

E.4 Allocating buffers to avoid channel interaction

If a 1284.4 device allows more than one channel to be open concurrently, the device's buffer pool will have to be distributed between the channels. The distribution of these buffers is important, as no data can be transferred on a channel with no credit. Implementers should be careful to prevent one channel from using all of the available buffers, as this could temporarily block transfer on the other channels.

E.5 Initialization negotiation

The initiator of the **Init** transaction should request the revision of the protocol it wants to establish when first attempting to establish a conversation with a 1284.4 peer. If the target 1284.4 peer responds with a failure indicating that it is unable to support the requested revision, the initiating peer may either:

a) retry the **Init** transaction with a lower revision that it also supports, or
b) retry the **Init** transaction with the revision set to the base revision (0x20) that all 1284.4 peers are required to support.

All 1284.4 implementations are required to support the base revision so that all 1284.4 devices are guaranteed the ability to hold a conversation using the functionality defined in this standard and referred to as the base revision (0x20). However, it is left up to individual 1284.4 implementations to decide how much revision negotiation to support.

E.6 Random back-off strategy for Init

Implementations shall provide a random back-off strategy used to successfully complete initialization when both devices simultaneously issue **Init** transactions (see 5.3.1- Init). The back-off time should be at least 100msec. The algorithm must be instance-specific so that two identical devices will successfully initialize. The granularity and latency of the data-link layer should also be considered so as to provide a truly random back-off on the transmission medium.

E.7 Enumerating services

1) The complete list of current services supported by a device may be enumerated using the **GetServiceName** transaction. To retrieve the list, simply initiate **GetServiceName** once for each SocketID parameter from 0x01 to 0xFF. All 255 sockets must be searched to ensure that all currently advertised services have been enumerated.

Note: The enumerated list contains all *currently advertised* services. A device's services may appear or disappear as functionality is added to or removed from the device.