Consultative Committee for Space Data Systems

DRAFT REPORT CONCERNING SPACE DATA SYSTEM STANDARDS

SPACE COMMUNICATIONS PROTOCOL SPECIFICATION (SCPS)—

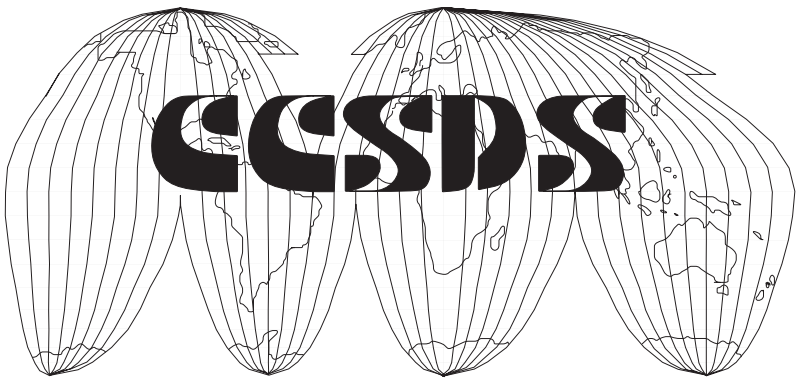
USERS GUIDE

(SCPS-UG)

CCSDS 711.0-G-0.2

DRAFT GREEN BOOK

September 1997



CONTENTS

CONTENTS (continued)

Figure Page

Table

1 INTRODUCTION

1.1 PURPOSE AND SCOPE

This draft Report is intended to serve as a reference resource for users of the CCSDS Space Communications Protocol Specification (SCPS) draft Recommendations (references [2]-[5]). It contains materials originally published as part of the first issue of the draft Recommendations, as well as some new material intended to clarify some of the protocol specifications.

Although this draft Report contains rationale and background information specific to selected specifications in each draft Recommendation, it is not intended to be an exhaustive treatment of the rationale and requirements associated with the SCPS program of work. A thorough study of SCPS rationale, requirements, along with application notes, is contained in the SCPS draft Green Book (reference [1]).

1.2 ORGANIZATION OF THE REPORT

This draft Report is organized into five sections. Section 1 contains introductory elements. Sections 2-5 contain ancillary information intended to aid users of the SCPS draft Recommendations, with each section corresponding to one of the four Red Books.

1.3 REFERENCES

NOTE – RFCs referenced in this document are available at the following URL:

http://ds.internic.net/ds/rfc-index.html

[1] *Space Communications Protocol Specification (SCPS)—Rationale, Requirements, and Application Notes*. Report Concerning Space Data System Standards, CCSDS 710.0-G-0.4. Draft Green Book. Issue 0.4. Washington, D.C.: CCSDS, October 1997.

[2] *Space Communications Protocol Specification (SCPS)—Network Protocol (SCPS-NP)*. Draft Recommendation for Space Data System Standards, CCSDS 713.0-R-3. Red Book. Issue 3. Washington, D.C.: CCSDS, September 1997.

[3] *Space Communications Protocol Specification (SCPS)—Security Protocol (SCPS-SP)*. Draft Recommendation for Space Data System Standards, CCSDS 713.5-R-3. Red Book. Issue 3. Washington, D.C.: CCSDS, September 1997.

[4] *Space Communications Protocol Specification (SCPS)—Transport Protocol (SCPS-TP)*. Draft Recommendation for Space Data System Standards, CCSDS 714.0-R-3. Red Book. Issue 3. Washington, D.C.: CCSDS, September 1997.

[5] *Space Communications Protocol Specification (SCPS)—File Protocol (SCPS-FP)*. Draft Recommendation for Space Data System Standards, CCSDS 717.0-R-3. Red Book. Issue 3. Washington, D.C.: CCSDS, September 1997.

2 SCPS-NP

The SCPS Network Protocol is described in terms of:

- the rules for formatting and parsing protocol data units (PDUs);
- the internal protocol procedures;
- the management interface with its management entity.

The internal protocol procedures define the procedures required for the transfer of information between SCPS Network entities. They apply to all SCPS Network entities through which data are to be routed, including those entities that may be called upon to provide cross support.

The rules for protocol header parsing and formatting govern the construction and interpretation of SCPS-NP PDUs. The SCPS-NP PDU headers are of variable length and format, depending upon the capabilities necessary to provide the services required for particular datagrams.

The management interface specifies the interactions between the SCPS Network and an external network management entity. These interactions are defined in terms of modifications to shared state information in the Management Information Base (MIB) and in terms of specific services provided to and expected from the network management entity.

2.1 FUNCTIONAL OVERVIEW

The SCPS Network provides unicast and multicast data services to its users. These services allow users to transmit data across a network of spacecraft and ground systems. The SCPS Network provides an unreliable end-to-end service: end-to-end reliability is provided by higher layers.

The SCPS Network provides users with a variety of addressing modes to meet the requirements for bit efficiency and end-system identification. Address modes support unique end-system identification within a SCPS Network domain, across multiple SCPS Network domains, and throughout the Internet. The address formats that support these modes are, respectively, basic addressing, extended addressing, and Internet Protocol version Six (IPv6) addressing.

The SCPS Network allows users to select the method by which a datagram will be routed through the SCPS Network. The methods include non-replicated routing and flood routing. The latter method duplicates datagrams during transmission to improve the probability of receipt at the expense of network bandwidth.

Routing of datagrams within the SCPS Network may be static (using routing tables established by an external management entity) or dynamic (using routing tables that are maintained both by management and by signaling between SCPS Network entities). The choice of static or dynamic routing is controlled via the MIB.

The SCPS Network provides a precedence (priority) mechanism. This mechanism affects the service provided to a datagram in two ways: order of service and service provided during periods of network congestion. The precedence mechanism allows high-precedence datagrams to be serviced before lower-precedence datagrams. Further, the precedence mechanism permits congestion-control mechanisms to discard, when necessary, low-precedence datagrams before higher precedence datagrams.

2.2 ADDRESSING

An address in the SCPS Network is referred to as an N-Address (for Network Address). An address family specifies the structural rules required to interpret the internal fields of an address. The SCPS Network provides users with three address families to meet the needs and requirements of various missions: the SCPS address family, the Internet Protocol (IP) address family, and the Internet Protocol version Six (IPv6) address family.

The SCPS address family contains both End System Addresses (identifying a single end system) and Path Addresses (identifying a pair of communication systems).

The IP address family defines address formats that are appropriate for routing and delivery across the Internet.

IPv6 format addresses are intended for those programs that do not have significant bit-efficiency issues and require global addressability.

The SCPS Network Protocol encodes those three address families into five address types. An address type defines the meaning that the addresses have (that is, whether they identify end systems or a path between end systems), the number of addresses that appear in a SCPS Network Protocol header (two addresses if the addresses identify end systems, only one if the address identifies a path between end systems), and the address family that is valid for the address. Table 2-1 describes the types of addresses that are supported by the SCPS Network Protocol.

Table 2-1: SCPS Network Protocol Address Types

Address Type	Address Length	Addresses per header	Address Family	Description
Extended End System Address	4 octets	2	IP or SCPS	Registered IP addresses or SCPS Address family addresses
Extended Path Address	4 octets	1	SCPS	Managed connection between source and destination(s)
Basic End System Address	1 octet	2	SCPS	Least significant octet of SCPS Extended End System Addresses
Basic Path Address	1 octet	1	SCPS	Least significant octet of SCPS Extended Path Addresses
IPv6 Address	16 octets	2	IPv6	Registered IP Version 6 addresses

The following paragraphs first specify the rules associated with each of the addressing families and then specify the format of each of the address types identified in table 2-1.

2.2.1 ADDRESS FAMILIES

2.2.1.1 SCPS Address Family

All SCPS address family addresses are drawn from the block reserved by the Internet Assigned Numbers Authority (IANA) for private Internet address spaces. This block is a subset of the full IP address space, and as such the addresses are 32 bits in length. (These addresses are represented as four eight-bit quantities separated by periods, e.g. w.x.y.z, where the range of each of the alphabetic characters is from 0 to 255 decimal.) Specifically, the range used for the SCPS Address Family is from 10.0.0.0 through 10.255.255.255. Using this address range, a SCPS address is represented in the form 10.x.y.z, where each value is eight bits in length. The w octet is constant at 10, decimal. The x and y octets are combined to form the addressing domain for various programs; the x.y field is known as the Domain Identifier (D-ID). A single mission may be allocated more than one D-ID for its needs. Determination of the registration authority to allocate and deallocate domain identifiers is beyond the scope of this Report.

The z octet is administered by the program that is allocated the Domain Identifier. z is further divided to support the types of addresses within the SCPS address family. The first seven significant bits are used to identify addresses. The eighth significant bit denotes whether the address is a multicast or unicast address and is known as the (M-Flag).

Each allocated domain, therefore, provides 128 SCPS unicast Path Addresses, 128 SCPS multicast Path Addresses, 128 SCPS unicast End System Addresses, and 128 SCPS multicast End System Addresses.

The form 10.x.y.z is the Extended form of a SCPS address. The Basic form of the SCPS address (i.e., z) may be used if it can be guaranteed (through network configuration) that the 10.x.y portion of the address will be unambiguous through the life of the datagram.

2.2.1.2 IP Address Family

SCPS-TP and SCPS-SP both use the Internet Protocol (IP) address family as their native mode of addressing. If an address is not in the private address space reserved by IANA, the address is a registered IP address. The structure of Internet addresses is defined in RFC 791, section 2.3; RFC 1112, section 4; and RFC 1122, section 3.2.1.3.

2.2.1.3 IPv6 Address Family

IPv6 address formats are a subject of ongoing work. The addresses are 16 octets in length. The current formats are specified in RFC 1884.

2.2.2 ADDRESS FORMATS

2.2.2.1 Extended End System Address

When two addresses are present in a SCPS-NP datagram, they each identify individual end systems. One address identifies the source of the data, the other identifies the destination(s). When each of these addresses is four octets in length, they are Extended End System Addresses. The format for Extended End System Addresses that belong to the SCPS Addressing Family is shown in figure 2-1.

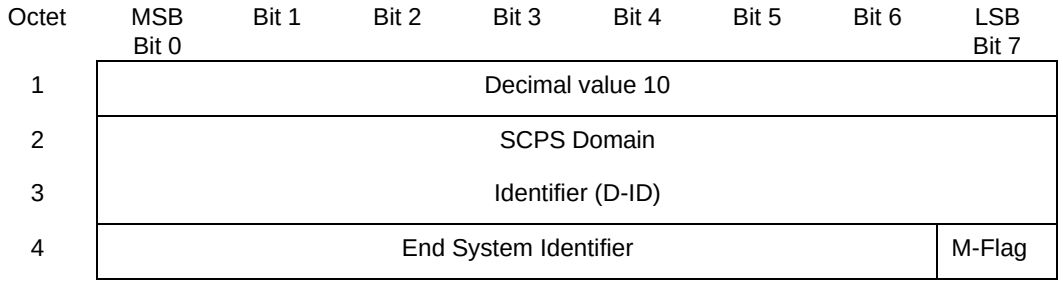


Figure 2-1: Extended End System Address Structure

The most significant octet of the SCPS Format Extended End System Address is fixed at Decimal 10. Octets 2 and 3 of the address are the Domain Identifier.

2.2.2.2 Extended Path Address

When one address is present in a SCPS-NP datagram, it identifies a “path”, which is roughly a permanent virtual circuit between a single source and one or more destinations.

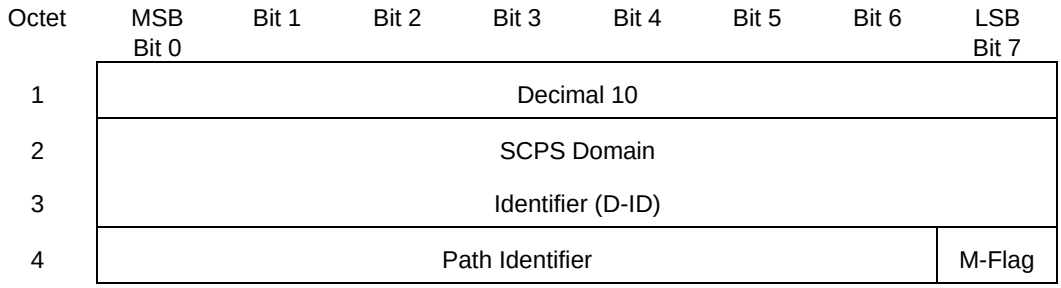


Figure 2-2: Extended Path Address Structure

2.2.2.3 Basic End System Address

When two addresses are present in a SCPS-NP datagram, they each identify individual end systems. One address identifies the source of the data, the other identifies the destination(s).

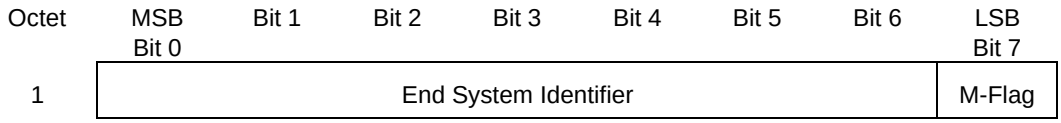


Figure 2-3: Basic End System Address Structure

The parameters of the Basic End System Address are the end system identifier (ES-ID) and the multicast flag (M-Flag). The M-Flag is a qualifier on the ES-ID to indicate that the address applies to a group of destinations.

2.2.2.4 Basic Path Address

When one address is present, it identifies a “path”, which is roughly a permanent virtual circuit between a single source and one or more destinations.

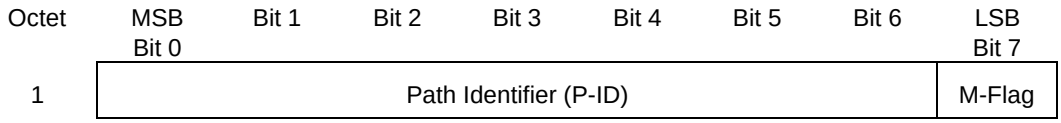


Figure 2-4: Basic Path Address Structure

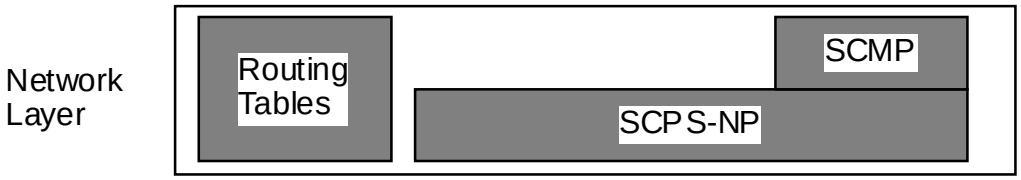
Figure 2-4 presents the structure of the Basic Path Address. The address is one octet in length. The parameters of the Basic Path Address are the path identifier (P-ID) and the multicast flag (M-Flag). The M-Flag is a qualifier on the P-ID to indicate that the address applies to a group of destinations.

2.2.2.5 IPv6 Address

IPv6 address formats are a subject of ongoing work. At this time, the addresses are 16 octets in length. The current formats are specified in RFC 1884. It should be stressed that this is work in progress, and subject to change. It is anticipated that the formats of these addresses will stabilize as the SCPS-NP review process progresses.

2.3 INTERNAL ORGANIZATION

The SCPS Network service is provided by three main elements within the network layer: the SCPS Network Protocol (SCPS-NP), the SCPS Control Message Protocol (SCMP), and Routing Tables. The SCPS-NP provides services to SCPS Network service users. The SCMP is an integral part of the SCPS Network Layer, and uses the services provided by the SCPS-NP to carry its messages. The SCMP interacts with the routing tables and protocols at other layers to convey and collect control information. Note that the conveyance of control information by SCMP to other elements within the local system is done via a set of control mechanisms that are separate and distinct from the data transfer mechanisms. These control mechanisms do not affect interoperability and are implementation dependent.



In addition to the elements within the network layer, the SCPS Network service is supported by capabilities that do not reside within the network layer. These capabilities include network management support for manual configuration of routing tables and maintenance of system configuration information, routing protocols for automatic configuration of routing tables, and name-to-address translation services. The network management and routing capabilities are typically implemented by applications that use application-layer protocols for the exchange of pertinent information. The name-to-address translation service is beyond the scope of the SCPS Network Protocol. Other capabilities, such as address resolution, in which a SCPS-NP address is resolved to a local subnetwork address, may be supported with mechanisms such as the Address Resolution Protocol (ARP), if appropriate for the particular subnetwork. Use of these protocols is not precluded by the SCPS-NP, but is subnetwork-specific.

3 SCPS-SP

3.1 INTRODUCTION

The SCPS-SP defines the services and procedures to provide secure transmission and reception of Protocol Data Units (PDUs) between communicating entities.

The SCPS-SP specification is described in terms of:

- the services provided to its users;
- the services required from a underlying network layer;
- the internal protocol procedures;
- the rules for formatting and parsing protocol data units;
- the interface with the Security Association (SA) database.

The SCPS-SP user services are defined in terms of primitives that are independent of specific implementation approaches.

The SCPS-SP presumes an underlying network layer that is capable of receiving Security Protocol Data Units (S-PDUs), applying, among other things, source and destination addressing, and transmitting them across the SCPS Network.

The internal protocol procedures define the procedures required for the transfer of information, securely, between SCPS Network entities. The rules for protocol header formatting and parsing govern the construction and interpretation of SCPS-SP PDUs. The SCPS-SP PDU headers consist of variable-length clear and protected headers.

The SCPS-SP operates with the assumption that a Security Association (SA) database exists that contains pertinent security information for use between the communicating entities such as the encipher key, the key expiration, the key length, the Initialization Vector (IV) length, the encipherment algorithm, the integrity algorithm, and the ICV length.

3.2 FUNCTIONAL OVERVIEW

The SCPS-SP provides network security services to its users. It is a connectionless security service which is a layer-3 subnetwork independent convergence protocol. The SCPS-SP provides the user with an integrity service, an authentication service, a confidentiality service, or a combination of all three services.

NOTE – The establishment mechanism for the encipherment and integrity keys is outside the bounds of this protocol specification. The keys and other security parameters may be negotiated using a key-management or SA protocol, or they may be manually pre-placed into a static SA database.

When *integrity services* are requested by a SCPS-SP user (e.g., an upper-layer protocol) or are required as a default action to enforce an administrative security policy, SCPS-SP shall calculate an ICV over the SCPS-SP clear and protected headers, the user data, which includes any upper-layer protocol headers, and potentially a secret data stream (e.g., a "secret key"). The size of the ICV is established in the SA database (*integ_alg_ICV_length*). However, the specific manner in which the ICV is calculated shall be determined by the integrity algorithm as identified by *integ_alg_id* in the SA database.

When *confidentiality services* are requested by a SCPS-SP user or are required as a default action to enforce an administrative security policy, the SCPS-SP shall use the encipherment key (*cipher_key*) in conjunction with the encipherment algorithm (*conf_alg_id*) and algorithm mode (*conf_alg_mode_id*) specified in the SA database to encipher the SCPS-SP protected header and the user data. The SA database is indexed by the source and destination addresses of the PDU.

When *authentication services* are requested by a SCPS-SP user, the source and destination network addresses are encapsulated into the SCPS-SP protected header and then either integrity and/or confidentiality services are applied, as above. Authentication *must* be requested with either integrity, confidentiality, or both. It cannot be provided without one or both of the other services.

When both integrity and confidentiality services are requested, the SCPS-SP first performs the integrity service followed by the confidentiality service. As a result, the protected header, the user data, and the ICV generated by the integrity service are enciphered.

3.3 SCPS-SP END SYSTEM AND INTERMEDIATE SYSTEM INTERACTION

The SCPS-SP may operate between two SCPS-SP End Systems, between a SCPS-SP End System and a SCPS-SP Intermediate System, or between two SCPS-SP Intermediate Systems. However, there is a distinction in SCPS-SP operation when an Intermediate System is involved in the secure communication.

A SCPS-SP-*aware* Intermediate System plays the role of a *security gateway* between security-aware and security-unaware systems. The Intermediate System provides secure operations to a *security front door* (e.g., a data handling system on a spacecraft) and allows smaller back-end systems (e.g., instruments aboard a spacecraft) the freedom to be security unaware.

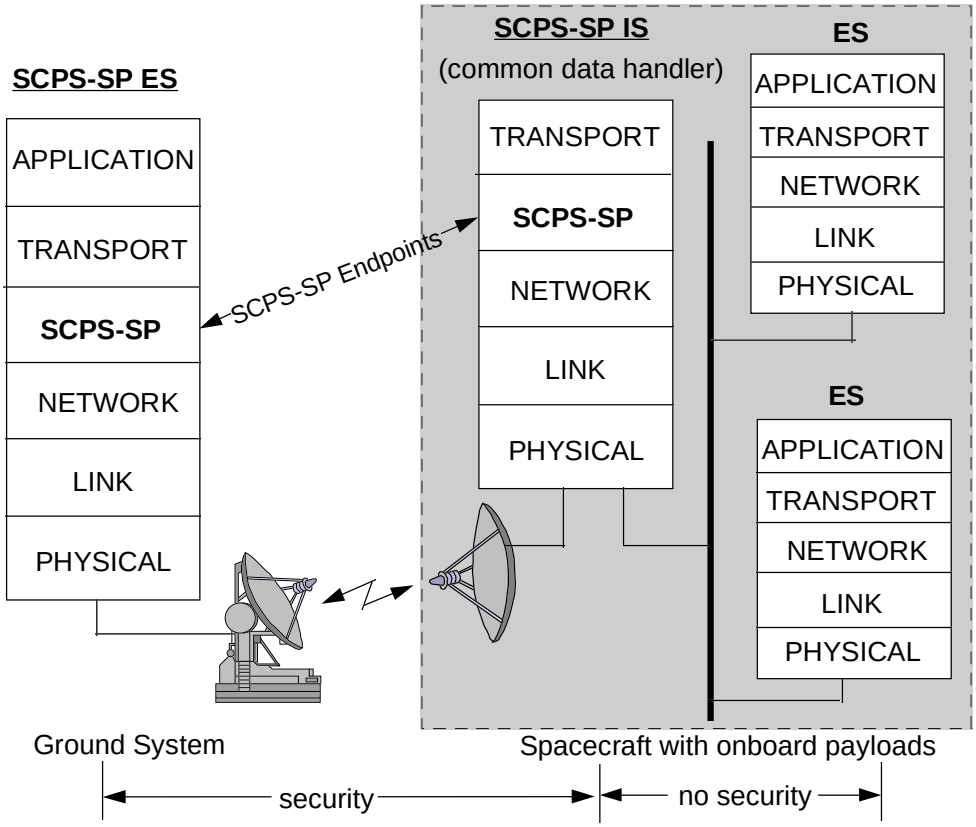


Figure 3-1: SCPS-SP End System and Intermediate System Interactions

When an Intermediate System (acting as a *security gateway*) is involved, the Intermediate System has the responsibility to implement SCPS-SP (see 3-1). The End Systems *behind* the Intermediate System create an *enclave* of systems that are unaware of the security protocol or its services. The Intermediate System is responsible for implementing the security protocol and performing all required security services on transmission and reception of PDUs. The *enclaved* End Systems would send and receive PDUs through the Intermediate System, unaware of the existence of the security protocol. However, all PDUs to and from the enclave would be routed through the SCPS-SP-aware Intermediate System. It should also be noted that since the Intermediate System performs all security processing, all the enclaved End Systems will be provided the same security services. When using such an Intermediate System *security gateway*, it is not possible to provide different security services for each of the End Systems in an enclave.

A source SCPS-SP End System or Intermediate System must know if an intended destination system employs SCPS-SP. This may be accomplished via the equivalent of a *routing table* as found in systems implementing the Internet Protocol (IP). If the intended destination does not employ SCPS-SP, the source SCPS-SP system must route the PDUs to a SCPS-SP-aware Intermediate System acting on behalf of the SCPS-SP-unaware End System. This is analogous to an Internet Protocol system routing packets to a *smart router* device because it lacks direct connectivity to a foreign destination system. The SCPS-SP source End System or

Intermediate System must establish an SA with the SCPS-SP-aware Intermediate System that acts as a security proxy for the intended End System.

The SCPS-SP assumes that a Security Association (SA) exists between the communicating systems which implement the SCPS-SP. That is, between the two SCPS-SP End Systems, a SCPS-SP End System and a SCPS-SP Intermediate System, or two SCPS-SP Intermediate Systems. The SA consists of a set of attributes that are used by the SCPS-SP to provide security services. These attributes are used by the authentication, integrity, and encipherment services to provide end-to-end security for PDUs.

4 SCPS-TP AMENDMENTS TO TCP

Section 4 of this document specifies amendments to RFC 793, the Transmission Control Protocol, and its supporting RFCs. Subsection 4.1 specifies revisions to connection management. Subsection 4.2 specifies revisions to the data transfer regime. Subsection 4.3 revises error-recovery mechanisms. Subsection 4.4 defines a Selective Negative Acknowledgment option. Subsection 4.5 specifies the header compression algorithm for use on SCPS TCP connections.

4.1 CONNECTION MANAGEMENT AMENDMENTS

4.1.1 INITIAL SEQUENCE NUMBER SELECTION

The TCP specification (refer to RFC 793, section 3.3 and RFC 1122, section 4.2.2.9) recommends that the Initial Sequence Number (ISN) be bound to a (possibly fictitious) clock that increments roughly once every four microseconds. The purpose of this binding is to ensure that segments from a previous incarnation of a connection are not misinterpreted as belonging to the current incarnation of the connection. The use of a clock is valuable in this case because it ensures that the ISN is not reset if there is a system crash. The rate of ISN increase is roughly tied to an estimate of the maximum transmission rate anticipated on a connection and ensures that the ISN clock does not wrap for over four hours.

A SCPS-TP conforming implementation is not required to use a clock as the basis for ISN selection. As long as ISN selection is robust against a possible crash, increases slightly faster than the maximum possible transmission rate, and does not wrap too quickly, then the algorithm used for ISN selection meets the intent of the requirement and is acceptable.

Note also that the ISN does not have to be updated at every clock tick. Rather, it only needs to be computed at the time a connection is established.

4.1.2 PRECEDENCE HANDLING

Replace RFC 793, section 3.6 with the following:

```
3.6  Precedence and Security

Security is handled external to the TCP, at a protocol layer that is
conceptually lower in the “stack”.  The TCP shall convey a user’s
security requests and replies to the security provider and shall
report responses and indications as required.  All other references to
security in this document shall be considered non-normative.

The precedence parameter used in TCP is as defined in the SCPS Network
Protocol.  The intent is that connection be allowed at the higher of
the precedence levels requested by the two ports attempting to
connect.
```

An endpoint shall request the precedence level specified by the calling application. (Any local policy constraints on precedence level requested by the application are outside the scope of this Report). If the remote TCP responds with a higher precedence level, the precedence of the connection is raised to the requested level, or, if local policy prohibits this, a reset is sent.

A listening endpoint that receives a connection request containing a higher precedence level than the endpoint's configuration will, if permitted by local policy, raise its precedence level to that specified by the remote endpoint and proceed with connection establishment. If local policy does not permit such an increase in precedence level, the listening endpoint will reject the connection with a reset.

If an endpoint has more than one socket with service requests to the transport protocol pending, the service order of those requests shall be determined by the precedence level of the socket and shall proceed starting with the highest precedence level.

4.1.3 NEGOTIATION OF SCPS CAPABILITIES

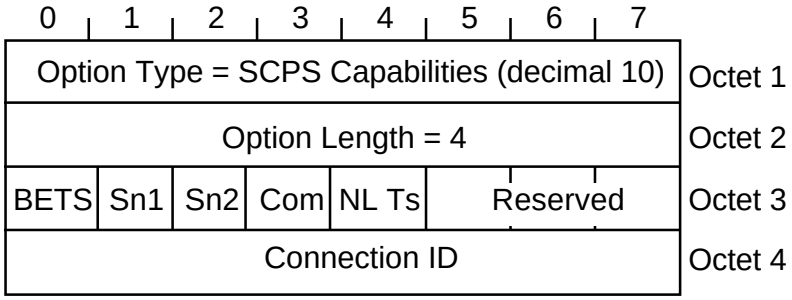
The availability of certain SCPS-specific capabilities must be negotiated at the time the connection is established. The SCPS Capabilities Option is defined for that purpose and may be included only on SYN segments. The endpoint initiating the connection specifies the desired capabilities to which it has access (via the local implementation). (The determination of which capabilities to use on a particular connection is implementation specific, possibly based on information from the routing structures. An application MUST be involved in the decision to use the Best Effort Transport Service (BETS).)

The specific information that is exchanged on the SCPS Capabilities Option is as follows:

- sender's willingness to use the BETS;
- sender's willingness to receive the short form and the long form (communicated separately) of the Selective Negative Acknowledgment (SNACK) Option;
- sender's willingness to receive SCPS Compressed TCP format headers;
- sender's desire to send SCPS Compressed TCP format headers and the connection identifier to associate with this connection;
- sender's local availability of Network-layer timestamps suitable for use in compressing the TCP Timestamps option (see discussion below).

If either endpoint does not send this option, the affected SCPS capabilities (BETS, SNACK, and SCPS Header Compression) are unavailable on the connection. If either endpoint does not send the SCPS Capabilities option, the operation of the protocol reverts to standard TCP operation (if the standard TCP capabilities have been implemented) or a RST segment is sent to abort the connection attempt (if the standard TCP capabilities have not been implemented).

The SCPS Capabilities option is formed as follows:



The value of the Type octet (decimal 10) is provisional, pending registration with the Internet Assigned Numbers Authority (IANA). The bits in Octet 3 have the following meaning:

Bit	**Meaning if = 0 ("Not OK")**	**Meaning if = 1 ("OK")**
BETS	Connection may not operate in BETS mode.	Sender willing to operate connection in BETS mode.
SN1	Do not send short form (length = 4) of SNACK option (refer to 4.5).	OK to send short form of SNACK option.
SN2	Do not send long form (length > 4) of SNACK option (refer to 4.5).	OK to send long form of SNACK option.
Com	Do not compress TCP headers.	OK to compress TCP headers—send connection identifier.
NL Ts	Network-layer timestamps not available or unsuitable for use in compressing TCP timestamps option.	Network-layer timestamps available and a timestamp accompanies this segment. If received, suitable, and available at both ends, use for compressing TCP timestamps option.

If both TCP endpoints send the BETS bit set to "OK", the connection will operate in BETS mode.

If TCP endpoint "A" sends the Com bit set to "OK" and TCP endpoint "B" sends a non-zero Connection Identifier (Octet 4), then endpoint B will send compressed TCP format headers to endpoint A.

If compressed headers are in use and both TCP endpoints indicate that use of Network Layer Timestamps (NL Ts) is acceptable, then outbound timestamps shall be carried in the timestamps field of the Network-layer header.

4.2 DATA TRANSFER AMENDMENTS

4.2.1 RECORD BOUNDARY OPTION

As specified in RFC 793, TCP provides a stream-oriented service. There is no mechanism to delimit individual records. This SCPS extension to TCP defines a Record Boundary option to allow a user to associate an end-of-record marker with a particular octet in the data stream and have that marker remain associated with the referenced octet throughout transmission and delivery.

The use of the Record Boundary option places some requirements on applications. Applications at both ends of a prospective connection must determine that a Transport service provides record boundary capability before connection establishment. (The Protocol Implementation Conformance Statement (PICS) shall state whether an implementation provides the Record Boundary option. For a run-time method of determining whether the Record Boundary option is available, refer to the Application Program Interface specification for the particular system on which the protocol is hosted.) If an application requests BETS and Record Boundaries, the application developer must ensure that the applications are robust against the possible loss of a record boundary. This may include incorporating a record length indication in the application record structure.

The Record Boundary option may be invoked when the user supplies data to the Transport service for communication to the remote Transport Service User (TSU). To invoke the Record Boundary option, a TSU provides an API-specific indication to the Transport service that the final octet of data in this service request represents the end of a record. This end-of-record indication, when supplied by the TSU, is associated with the last octet in the service data unit with which it was submitted. The ending delimitation of the record shall be preserved during transmission (and possible retransmission) and across the remote service interface interfaces. The sequence number of the octet associated with the record marker shall be the highest sequence number in the segment carrying the record boundary option, and a read shall associate the end of record with that octet. The octet associated with the record boundary option shall not be changed by the Transport Service Provider (TSP). If any portion of a segment carrying the end-of-record option falls to the right of the receiver's window, the end-of-record option is discarded (until the final octet of that segment, with which the end-of-record option is associated, is received).

The Record Boundary option (RBOpt) is two octets in length and has a Type = 0x19 (25 decimal). It may be requested by an application to be associated with any data in the stream, with the constraint that the Record Boundary is associated with the final octet of data in the particular service request (e.g., write call). Use of the record boundary is valid whenever transmission of data is valid.

The effect of the Record Boundary on the Transport Protocol is that the octet associated with the end-of-record option shall have the highest sequence number in any segment that carries it. (That is, it shall be the final octet in the segment.) This applies to the initial transmission and

any possible retransmissions. (This modifies the effect of the Nagle algorithm and constrains the Transport protocol in performing repacketization during retransmission. Refer to RFC 896 for a description of the Nagle algorithm.)

4.2.2 ADDITION OF BEST EFFORT TRANSPORT SERVICE

Amend RFC 793, section 1.5, paragraph on "Reliability" to the following:

> The TCP provides two modes of operation: Fully reliable Transport service and BETS. Fully reliable Transport service is provided for "stateful" applications that depend on all data being reliably transmitted and received. BETS is provided for two main users: those that must continue to communicate during degraded conditions with as much reliability as possible; and those that wish to have the service of a stream-oriented Transport protocol but wish to sacrifice some reliability in order to limit buffer use on resource-restricted systems.
>
> In its fully reliable mode, TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the Internet communication system. This is achieved by assigning a sequence number to each octet transmitted and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to put segments in the correct order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.
>
> As long as the TCPs continue to function properly and the Internet system does not become completely partitioned, no transmission errors will affect the correct delivery of data. TCP recovers from Internet communication system errors.
>
> In its best-effort mode, TCP recovers from data that is damaged, lost, duplicated, or delivered out of order *to the extent possible given the constraints imposed by the user application.* The best-effort mode allows the sender to control the limit on segment retransmission in order to manage retransmission buffer space. The user has the ability to specify how many times a segment will be retransmitted (including zero times, in which the segment is discarded after its initial transmission and the buffer space is recovered). The sending TCP maintains a retransmission buffer until a segment is acknowledged, until the segment exceeds its maximum retransmission threshold, or the send window fills. When a segment exceeds its maximum retransmission threshold without acknowledgment or the send window fills, the TCP treats it as having been acknowledged (rather than aborting or halting the connection). The sending application may query the SCPS-TP (TCP) implementation in an API-specific manner to determine the location and length of segments that were not acknowledged by the receiver. There may be system-dependent limits on the amount of information that is kept by SCPS-TP. The implementation shall document the specific limitations on this capability.
>
> The receiving TCP is configured by its application to operate in best-effort mode. In this mode, the receiver behaves in the same manner as fully reliable TCP until an out-of-sequence queue forms. (An out-of-sequence queue forms when a segment is lost or delayed, but one or more segments with higher sequence numbers are received. A queue is formed, but a "hole" for the missing segment is maintained. The queue

of higher-sequence number segments is the out-of-sequence queue.) When an out-of-sequence queue is formed and when the receive buffer space that has been used reaches x% of available space, the receiving TCP will (a) issue an acknowledgment that acknowledges the data in the missing segment plus any data in the out-of-sequence queue that would be acknowledgeable were the missing segment received, (b) issue an error/warning/advisory to receiving application identifying size of the hole (in octets), and then (c) deliver the subsequent in-sequence data to user.

Amend RFC 793, section 3.7, add the following paragraphs at the end of the section:

Best Effort Transport Service

Objectives

There are two objectives of the Best Effort Transport Service: to allow communication to continue in the event that the ability of the receiver to acknowledge data is interrupted independently of the ability of the sender to transmit data, and to allow a sending application to establish an appropriate tradeoff between required connection reliability and use of retransmission buffer resources. The first objective is aimed specifically at spacecraft operations in which transmitters and receivers use separate electronics, which may fail independently of each other. The Best Effort Transport Service can permit a ground system to continue commanding a spacecraft even if the spacecraft's transmitter is damaged and it cannot transmit acknowledgments. The second objective is aimed at payload users that wish to send large volumes of data and desire that the data be delivered correctly, in sequence, and without duplicates, but do not necessarily require completeness. An example of a potential user of this service is an imaging application that can tolerate the loss of an occasional scan line, but does not have the buffer space on board for fully reliable communication.

Invocation

The Best Effort Transport Service is invoked using the SCPS Capabilities option. Refer to 4.1.3 for a discussion of this option and its use.

Sender Operation

TCP uses the occurrence of excessive retransmission of a single segment as an indication of failure in the network or the remote host. There are two thresholds, R1 and R2, measuring the amount of retransmission that has occurred for an individual segment. Refer to RFC 1122, section 4.2.3.5, for a full description of these two thresholds. With Best Effort mode, R1 and R2 shall be interpreted as a count of transmissions (not time nor *re*transmissions). Further, R2 shall be local to a connection and accessible for reading and writing by the user application via a Transport-interface option.

For fully reliable operation, R1 is the threshold that indicates a degraded connection and the need for diagnosis and possible re-routing. The R2 threshold indicates a broken connection. These meanings are similar in Best Effort mode, but modified slightly; when the R1 threshold is reached, it is still an indication that the connection is degraded, but R1 may not be active on all connections, depending on the value of R2. R2 is the threshold at which attempted retransmission of a segment is discontinued.

If the value of R2 is set to a non-zero positive number, then the value of R1 shall be set to a value less than that of R2. The value of R1 shall be set by the local TCP implementation. If the value of R1 is forced to zero because R2 is set to one, no action will be taken in response to the R1 threshold being exceeded.

When Best Effort mode is enabled and the R2 limit is set to one, then segments may be discarded after being initially transmitted rather than being queued for retransmission.

When Best Effort mode is enabled and the R2 limit is greater than one and the number of transmissions of the same segment reaches or exceeds R1, then the same negative advice that is generated for fully reliable mode may be invoked. (Exceeding the R1 threshold typically results in an attempt to identify a different route to the destination.)

When Best Effort mode is enabled and either the R2 limit is reached (or exceeded) for a segment or SND.NXT = SND.UNA + SND.WND, the sending TCP shall behave exactly as if it had received a positive acknowledgment that advances SND.UNA to SEG.SEQ of the next segment in the retransmission queue and that makes no change to SND.WND. (SND.UNA is the sequence number of the first unacknowledged octet. SND.NXT is the sequence number of the next octet to be sent. SEG.SEQ is the sequence number of the first octet in a segment. SND.WND is the number of octets of unacknowledged data that the sender is authorized by the receiver to have outstanding.)

If R2 is set to zero by an application, it is an escape value that indicates that the TCP should not break a connection due to excessive retransmissions, nor should it invoke the Best Effort Transport Service. Rather, it is a request to retransmit an “infinite” number of times. (If R2 is set to zero, R1 behaves as it would for fully reliable operation in which R2 is set at a “reasonable” threshold.)

The TCP implementation shall provide sending applications with a means to determine which data elements may not have been acknowledged by the receiver. (There may be cases in which data segments were received by the receiver but their acknowledgments were lost. Therefore, possible reasons for not receiving acknowledgments are: (1) data segments were lost, or (2) acknowledgments were lost. The sender cannot tell which of the two reasons is the real reason. Also note that, when in BETS mode, acknowledgments received from the receiver may contain acknowledge data that was not received.) The implementation shall specify the missing data elements by reporting a pair of numbers for each missing data element. The first number in the pair shall be the byte offset from the start of the connection to the first byte of unacknowledged data. The second number in the pair shall be the byte offset from the start of the connection to the last octet of unacknowledged data. Once read by the application, the missing-data-element specification pair will be removed from the list. The TCP implementation may restrict the amount of memory that is available for storing these pairs of numbers and, if so, may treat the available memory as a circular buffer. If an application does not request a report of missing data elements before the available memory fills, information could be lost.

For TCP implementations that support the Record Boundary option, the implementation MAY specify missing data elements by reporting an additional pair of numbers for each missing data element. The first number of the additional pair is the record offset from the start of the connection to the beginning of the unacknowledged data. The second number of the additional pair is the record offset from the start of the connection to the end of the unacknowledged data.

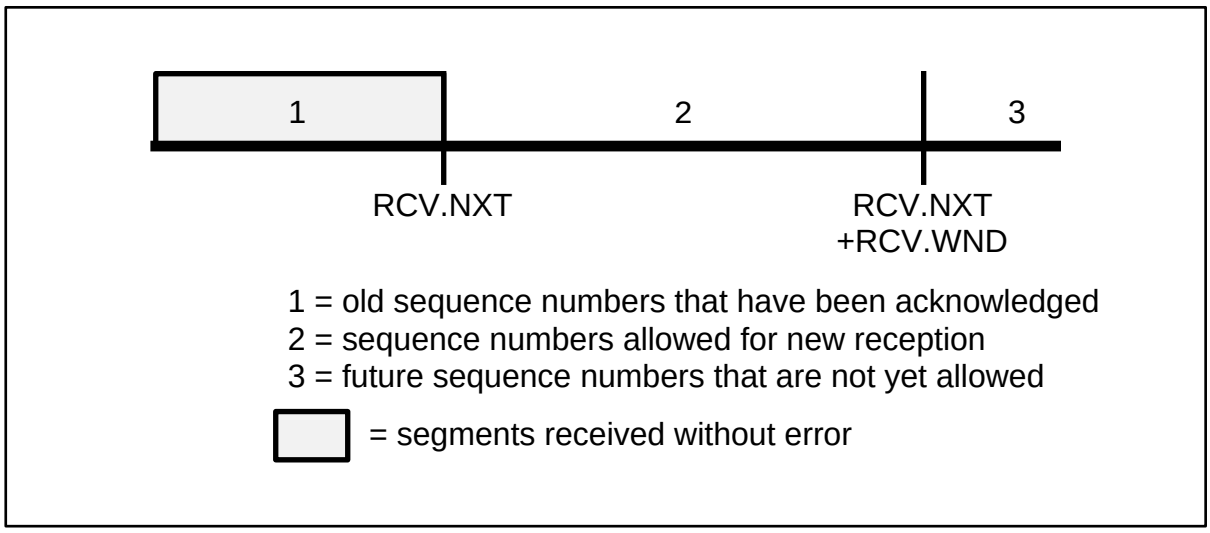


Figure 4-1: Receive Queue with No Out-of-Sequence Segments

Figure 4-1 illustrates the normal situation with a receiver that has received no out-of-sequence segments. Area 1 extends from the initial sequence number of the connection up to (but not including) the sequence number RCV.NXT. This represents data that has been received and acknowledged. Area 2 represents the range of sequence numbers that are allowed for new segments to be received. This area is bounded on the low end by the latest sequence number acknowledged (RCV.NXT), and on the high end by the size of the window (the sequence number for this bound being RCV.NXT plus RCV.WND). Area 3 shows sequence numbers that are not yet allowed because of the window size. The shaded area illustrates segments that have been received. (The fact that this area corresponds to Area 1 indicates that all data that has been received has been acknowledged. In some cases, the shaded area could extend into Area 2, indicating that data had been received, but not yet acknowledged.)

Figure 4-2 illustrates the receive queue in a situation in which a segment has either been lost or was corrupted. As before, Area 1 illustrates the sequence number range that has been acknowledged, Area 2 illustrates the sequence number range that is allowed for new reception, and Area 3 illustrates the sequence number range that is not yet allowed. Consider, however, the shaded areas representing segments received and lost. The lighter shading represents segments that have been received correctly. The darker shading represents missing segments. Note that RCV.NXT has not (and can not, in fully reliable operation) advance past the missing segment. Segments can continue to be received and enqueued for the full sequence-number range defined by Area 2. When the missing segment(s) from the dark area are received, all of the segments that are to the right of the dark area can be acknowledged, and RCV.NXT will move to the right of the last octet received. Correspondingly, Area 2 will move to the right, such that the right edge of Area 2 is defined by (the new value of) RCV.NXT plus RCV.WND.

When operating in Best Effort mode, the receiving TCP must be prepared to deal with segments that will not be retransmitted. If the receiving TCP took only the action of the fully reliable TCP while the sending TCP operated in Best Effort mode, the situation could arise

that the receiving TCP would wait infinitely for a segment that would not be retransmitted because the sender's R2 had been exceeded.

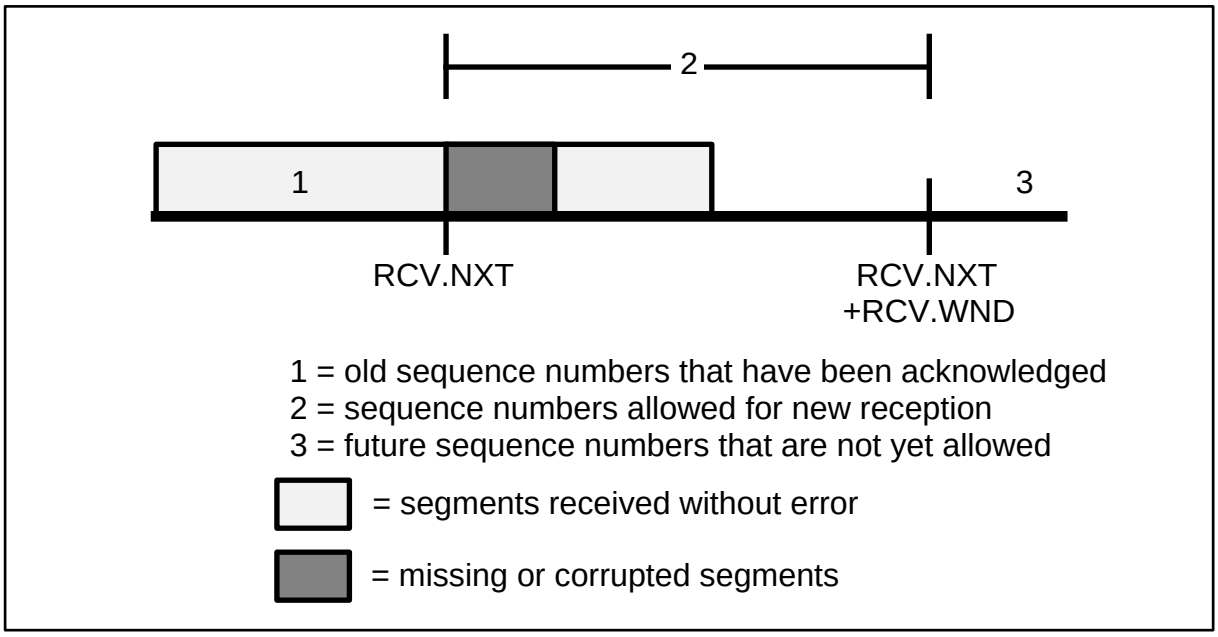


Figure 4-2: Out-of-Sequence Queue Formation

As a result of this, in Best Effort mode, the receiving TCP must have a threshold at which it ceases to wait for an out-of-sequence segment and simply "moves on." In fact, two thresholds are required: one based on the amount of out-of-sequence data received, and one based on the time that out-of-sequence data has been waiting for delivery. The size-based threshold, BE1, is defined as a value in octets that corresponds to a locally administered percentage of the receiver's buffer space. The time-based threshold, BE2, is defined as the interval in locally sized clock ticks after which out-of-sequence data will be delivered to the user. Figure 4-3 shows the out-of-sequence queue again, but redefines Area 2 to be the range of sequence numbers between RCV.NXT and the Best Effort threshold BE1. Area 2 and Area 3 both represent permitted sequence numbers for new reception, together corresponding to Area 2 from previous figures. Area 4 represents the sequence number range that is not yet valid.

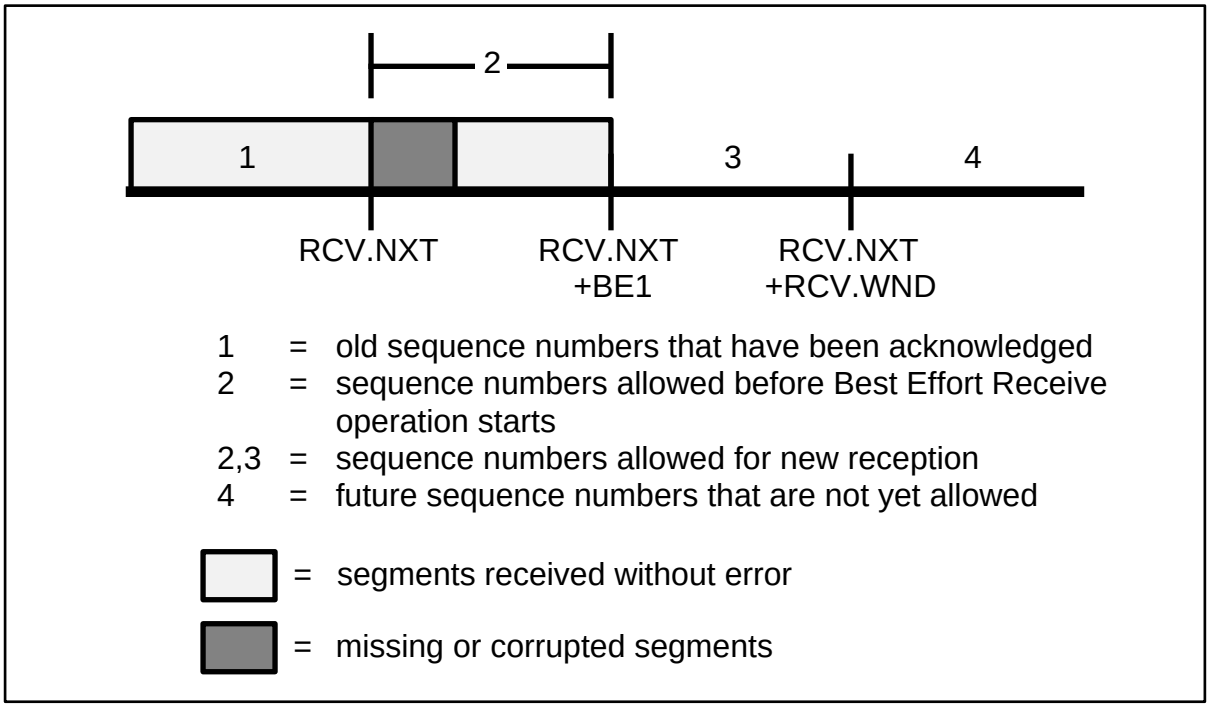


Figure 4-3: Best Effort Receive Threshold Placement

The time-based threshold BE2 is required to ensure that an out-of-sequence queue representing the end of a transmission neither waits infinitely for delivery nor requires the connection to be closed to ensure delivery. When out-of-sequence data is received but no previously enqueued out-of-sequence data exists (i.e., the transition from figure 4-1 to figure 4-2 occurs), an interval timer is started. If the out-of-sequence data is not delivered by the time the BE2 interval elapses, the actions described below are taken.

Figure 4-3 shows a situation in which the out-of-sequence queue has reached the BE1 threshold. When the receiving TCP is operating in Best Effort mode and the size of the out-of-sequence queue reaches or exceeds BE1 or the interval BE2 elapses, then the following actions shall be taken by the receiver:

a) Issue an acknowledgment that acknowledges the data in the missing segment plus any data in the out-of-sequence queue that would be acknowledgeable were the missing segment received. (In figure 4-2, the acknowledgment would move RCV.NXT to the right of the rightmost lightly shaded box. Note that *this acknowledgment is a lie* -it may be considered to be a "pseudoacknowledgment". The data from the darkened area was never received. This is what differentiates the Best Effort receiver from the fully reliable receiver.)

b) Issue an error or warning or advisory to receiving application identifying size of the hole (in octets). (In figure 4-2, the

size of the dark-shaded box would be reported to the receiving application as part of an error message. By issuing the indication in-band, the location of the missing data is implicit, and only its size need be reported.)

c) Deliver the subsequent in-sequence data to user.

Figure 4-4 shows the slightly more complex case in which the receive queue has multiple holes in it. In this case, Areas 1 through 4 are identical to those of figure 4-3, as are the reference points RCV.NXT, RCV.NXT plus BE1, and RCV.NXT plus RCV.WND. At this point in the communication, the BE1 threshold has been met, and a Best Effort acknowledgment must be issued. However, paragraph (a) above indicates that, rather than issuing an acknowledgment that moves RCV.NXT to the right edge of Area 2 (RCV.NXT plus BE1), the acknowledgment will move RCV.NXT to the sequence number indicated by "A" in figure 4-4. This has the effect of shifting Areas 1 through 4 to the right by the amount of A minus RCV.NXT octets.

If the missing segment immediately to the right of A is received before enough new data is received to reach the new Best Effort threshold, no loss of data is necessary. This is the reason that the Best Effort acknowledgment does not advance the sequence number to the end of the Best Effort threshold.

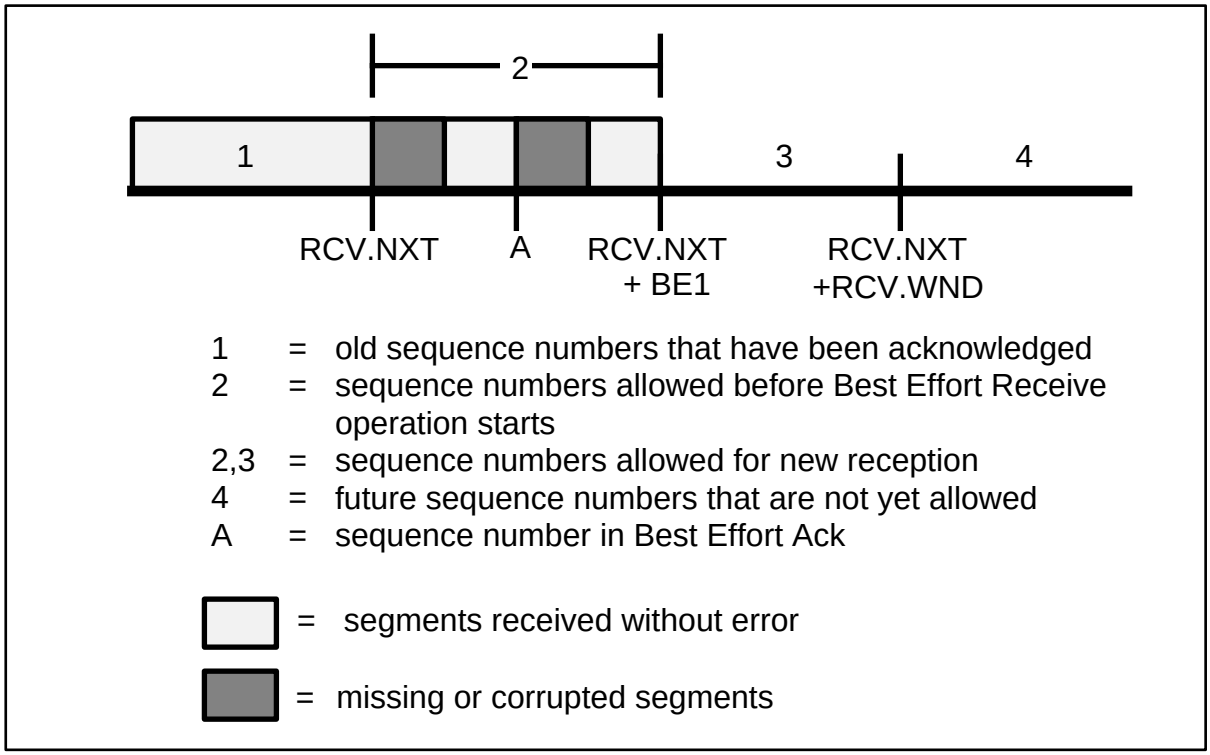


Figure 4-4: Best Effort Receive with Fragmented Out-of-Sequence Queue

Figure 4-5 shows the result of issuing the Best Effort acknowledgment (with "A" as the sequence number). Note that RCV.NXT has been

advanced to the next hole in the out-of-sequence queue, shifting Areas 1 through 4 to the right. The segment(s) to the left of RCV.NXT that are darkly shaded are missing. If these segments arrive after RCV.NXT has advanced, the receiving TCP may discard them. (Alternatively, the receiving TCP may store them via some out-of-band storage means for off-line merging with the rest of the data. However, this requires the receiving TCP to maintain the sequence number and size of each area of missing data. A Best Effort receiver that will be receiving large volumes of data will probably exist in a (ground-based) system with substantial receive buffer capacity. Therefore, it is anticipated that the threshold BE1 will be able to be set sufficiently high that most retransmissions will be cleared from the network by the time BE1 is met or exceeded.)

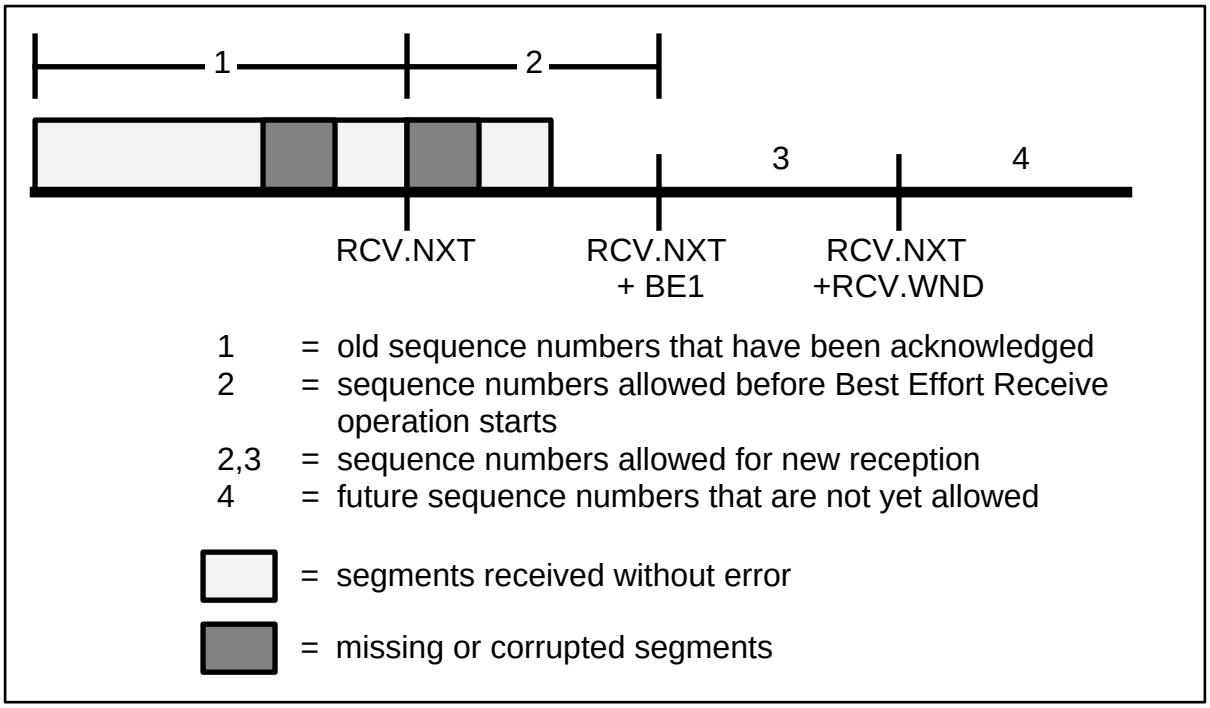


Figure 4-5: Fragmented Out-of-Sequence Queue after Best Effort Acknowledgment

4.3 ERROR RECOVERY AMENDMENTS

Amend RFC 793, section 3.7, replace the "Retransmission Timeout" section with the following paragraphs:

Because of the variability of the networks that compose a space-based internetwork, the retransmission timeout should be dynamically determined. However, it is possible that some spacecraft have such limited resources that the dynamic computation of the retransmission timeout cannot be accomplished. In these cases, the retransmission timeout may be statically configured via the MIB. The MIB supports static selection of the retransmission timeout mechanism.

When retransmission timeouts are dynamically determined, the requirements in this paragraph are in force. The Jacobson algorithm for computing smoothed round-trip time (RTT) shall be used to estimate

the RTT and its variance. The Karn algorithm shall be used to select those RTT measurements that are valid except when the TCP Timestamps option is present. In such a situation, the retransmission ambiguity problem cited by Karn's algorithm does not exist, and the round trip of a retransmitted segment shall be factored into the RTT estimate.

When the implementation has elected not to provide congestion control, the following algorithms are not available: Van Jacobson's Slow Start algorithm, Van Jacobson's Congestion Avoidance algorithm, and the exponential back-off of the retransmission timer for successive retransmissions.

Implementations that provide congestion control may do so in one of two ways: by using the standard means within TCP (i.e., Van Jacobson's Slow Start Algorithm, Van Jacobson's Congestion avoidance algorithm, and exponential back-off) or by using the TCP Vegas congestion-control mechanisms.

When congestion control is implemented but there is evidence that data loss is due to corruption, the "exponential backoff" algorithm may optionally not be invoked. This evidence of corruption rather than congestion may be obtained from inter-layer signaling or from a MIB-query. When a connection receives an indication that corruption has been experienced, it sends the corruption-experienced option (Type = 11, Length = 2, pending registration with the IANA) to the remote TCP at an implementation-defined rate not to exceed once per round-trip time, and continuing for no more than two round-trip times after the previous indication of corruption was received. Receipt of the corruption-experienced option from a remote TCP shall not cause the receiver to send a corresponding corruption-experienced option to its peer.

For retransmissions of data in which there is no indication of corruption being experienced, the exponential backoff algorithm shall be used to compute successive Retransmission Timeout values for the same segment.

When congestion control is implemented, the Jacobson algorithms for slow start and congestion avoidance or corresponding TCP Vegas algorithms shall be used when the default source of loss is congestion and there is no evidence that data loss is due to a cause other than congestion. The use of these algorithms when there is evidence of corruption is optional, but may result in reduced performance.

When the TCP receives an indication of a link outage, either from inter-layer signaling or from MIB-query, it shall suspend transmission. The retransmission timer shall be suspended. User data may continue to be enqueued by TCP for transmission, to the extent that retransmission buffer space permits (this is a local implementation option). Periodically, at a locally determined rate (recommended to be some small multiple of the previous retransmission timeout) the TCP shall send a link-probe segment, consisting of the first octet of unacknowledged data. (If no unacknowledged data exists, no link probe is sent. If data is submitted for transmission during a link outage, the link probes shall commence.) Link probe segments shall not be interpreted as retransmissions for the purposes of modification of the retransmission timeout or for invocation of slow start or congestion avoidance. When the TCP receives a "link restored" or "link redirect" control message, determines that the link is available from a MIB-query, or receives an acknowledgment that arrives at least one RTT after the commencement of the link outage, normal communication resumes.

4.4 SELECTIVE ACKNOWLEDGMENT SPECIFICATION

The TCP cumulative acknowledgment mechanism, combined with congestion control, performs poorly in the SCPS environment. A selective acknowledgment strategy provides precise information to the sender about missing data and can therefore facilitate a more prudent retransmission scheme and provide improved performance.

Two selective acknowledgment schemes have been proposed for TCP; neither is in wide use. The following paragraphs describe the two options and identify the problems with using those options in the SCPS environment. Subsequently, 4.4.5 presents the SCPS Selective Negative Acknowledgment Option, which addresses those problems.

4.4.1 THE RFC 1072 SELECTIVE ACKNOWLEDGMENT OPTION

The Selective Acknowledgment (SACK) mechanism defined in RFC 1072 consists of an option in the TCP header sent by the data receiver to the data sender. The option contains advisory information regarding non-contiguous blocks of data that have been received and queued by the receiver. The SACK option does *not* change the interpretation of the acknowledgment number in the regular TCP header. The intention is that the data-sender will optimize retransmissions based on the additional information provided by the SACK option. The option itself consists of a variable-length list of pairs of 16-bit integers. Each pair defines a block of contiguous sequence space that corresponds to data that has been received and queued by the receiver. A pair consists of a Relative Origin and a Block Size for each isolated, contiguous block of received data. The Relative Origin specifies the first sequence number in a block, relative to the acknowledgment number in the TCP header. The Block Size specifies the size in octets of the block of data.

4.4.2 PROBLEMS WITH THE RFC 1072 SACK OPTION

The ability to acknowledge multiple discontiguous blocks is desirable, and is provided by the Selective Acknowledgment mechanism proposed in RFC 1072; however, three problems exist with the scheme:

1) The SACK option is large. Because a limit exists on the size of the TCP header (64 bytes, of which 20 bytes are the basic TCP header), and because a SACK option that specifies n blocks has a length of $4n + 2$ bytes, a single SACK option is capable of specifying at most 10 blocks. If other TCP options are in use, a given SACK option may not be able to specify even 10 blocks. This limitation reduces the advantage gained by using Selective Acknowledgments over cumulative acknowledgments, especially when operating with large windows in a noisy environment.

2) The SACK option is imprecise. When RFC 1323 Window Scaling is enabled, the window size can effectively be as large as a 30-bit quantity. The SACK option uses a

16-bit field to identify the location, or Relative Origin, of a block of data within this window space. Likewise, the Block Size field, which specifies the size in bytes of a contiguous block of data is a 16-bit quantity. Clearly, these fields cannot address the entire window space when scaling is enabled. The two solutions to this problem proposed in RFC 1072 are both inadequate. The first is to expand the SACK Relative Origin and Block Size fields to 24 or 32 bits each. From the perspective of bit-efficiency, this solution is unacceptable. In addition, this approach drastically reduces the number of blocks that can be specified by a SACK option. The second proposal is to scale the SACK fields by the same value as the window. This solution introduces imprecision into the acknowledgment, since the SACK option must report block offsets and lengths in multiples of the window scale factor, which may not be a multiple of the segment size. To reconcile this imprecision, it is necessary to adopt a conservative approach and unnecessarily retransmit some data when there is doubt as to which segments are being acknowledged. This approach also makes inefficient use of the channel.

3) The SACK option is incompletely specified. RFC 1072 describes the format of the SACK option, but it does not cover other essential issues, such as when to send a SACK as the receiver of data, or how to process one as the data sender. RFC 1072 also fails to mention the interaction between the SACK option and the congestion control algorithms, which governs the system throughput in the event of packet loss in standard TCP.

4.4.3 THE RFC 1106 NEGATIVE ACKNOWLEDGMENT OPTION

RFC 1106 proposes a Negative Acknowledgment (NAK) option. This extension permits the data-receiving TCP to inform the sender that a block of data was not received and needs to be retransmitted. Like the RFC 1072 SACK option, this extension does not alter the meaning of the regular acknowledgment field in the standard TCP header, but simply provides additional information that may be ignored without affecting TCP's current behavior. The NAK option contains the sequence number of the first byte being NAKed and the number of consecutive maximum-size segments being NAKed. When the data-sending TCP receives a NAK it may elect to retransmit the NAKed segments immediately or it may ignore the NAK information.

4.4.4 PROBLEMS WITH THE RFC 1106 NAK OPTION

The RFC 1106 NAK option is reasonably bit-efficient, but it has the ability to signal only a single "hole" that exists in the sequence space in the receiver's buffer. A more powerful mechanism, capable of specifying multiple holes, is desirable in the noisy, long-delay environment to provide the sender with more complete information about the state of the receiver's potentially large out-of-sequence queue. (Note that large send and receive buffers are required when operating with large windows.)

4.4.5 SELECTIVE NEGATIVE ACKNOWLEDGMENT OPTION

The SNACK option takes into consideration the strengths and short-comings of the RFC 1072 SACK and RFC 1106 NAK options. The SNACK option is defined to address the issues identified with the former selective acknowledgment mechanisms and to meet the SCPS requirements.

The SCPS SNACK is a variable-length TCP option that signals the presence of multiple holes in the receiver's resequencing queue in a bit-efficient manner. The SNACK option consists of four fields: the kind and length fields required of all TCP options, followed by the 16-bit "Hole1" field and an optional variable-length bit-vector. (If the bit-vector is not included, the length of the option is fixed at four bytes. If the bit-vector is included, the length of the option becomes implementation dependent.) Hole1 signals the size of the first hole in the receiver's buffer, where the starting location of the hole is specified by the ACK number carried in the TCP header. The bit-vector maps the sequence space of the receiver's buffer into Maximum Segment Size (MSS)-sized blocks, beginning one byte beyond the end of the block specified by Hole1. Each "0" in the bit-vector signifies that one or more bytes are missing in the corresponding block of the receiver's resequencing queue. (Note that any zeroes to the right of the last "1" in the bit vector are NOT interpreted as a hole.) The length of the bit-vector, which may vary at the SNACK-sender's discretion, is determined from the option length. Use of the SNACK option is enabled by the SCPS Capabilities Option, which may be sent only on the SYN segment of the connection. Pending approval from the Internet Assigned Numbers Authority (IANA), SNACK uses option number 15.

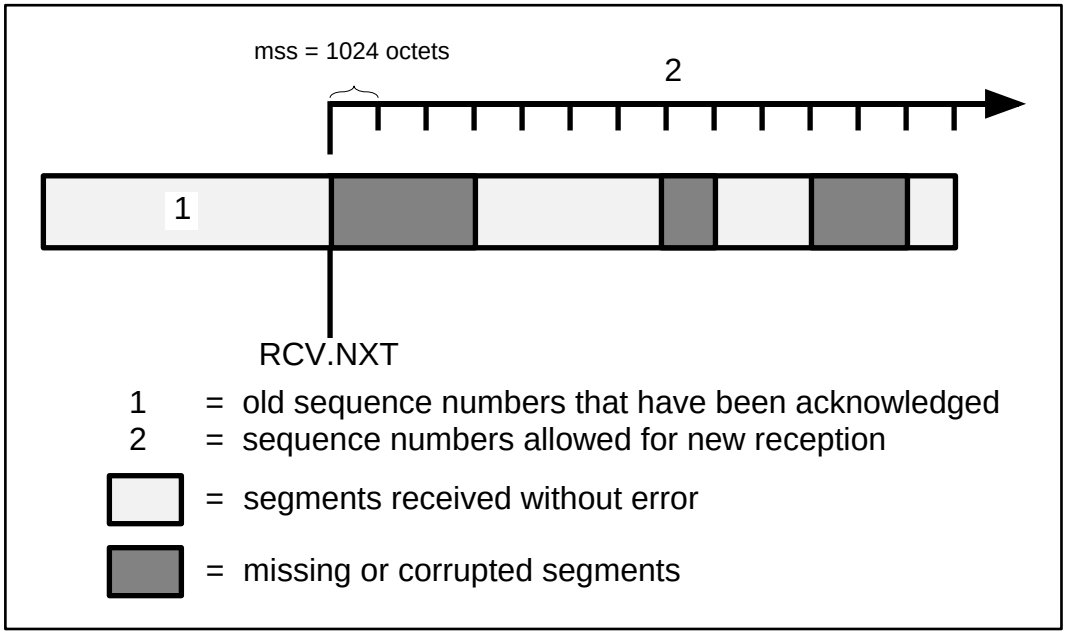


Figure 4-6: Out-of-Sequence Queue for SNACK Example

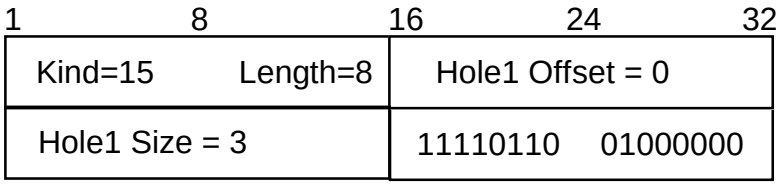


Figure 4-7: SNACK Option Resulting from Out-of-Sequence Queue Example

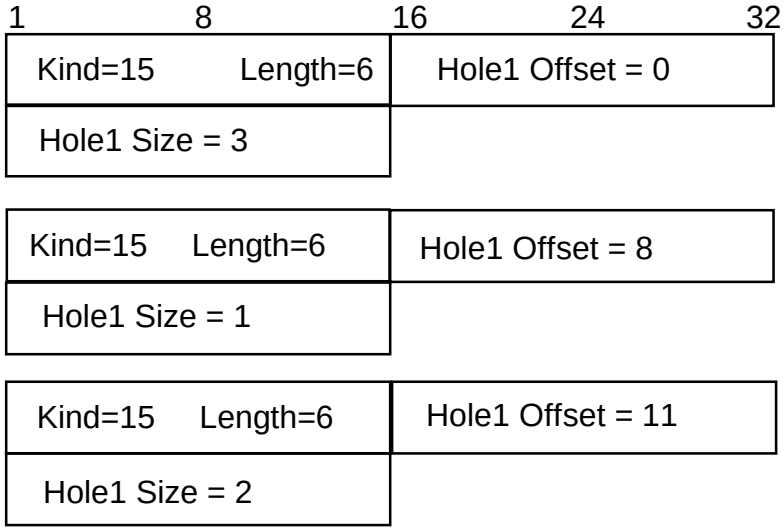


Figure 4-8: SNACK Options (without SNACK Bit-Vector) Resulting from Out-of-Sequence Queue Example

The data-receiver sends an appropriately formed SNACK option on an ACK segment whenever an out-of-sequence queue forms. There are mission-specific considerations regarding whether transmission of the SNACK option should be delayed, and by how much, in anticipation of the missing segment(s) arriving out of order. This decision is implementation specific and should take into account the probability of segment misordering by the underlying network(s). Unless segment misordering is highly unlikely, delaying transmission of the SNACK option may be beneficial. (The SNACK option forces immediate retransmission; as such, the SNACK sender should be relatively certain that the retransmission is necessary.)

Upon receipt of a SNACK option, the data-sender retransmits all segments necessary to fill the signaled holes. (This relatively aggressive retransmission scheme is appropriate when the goal is to prevent retransmission time-outs, which cost more in terms of link idle time than unnecessary retransmissions cost in terms of wasted bandwidth.) The data-sender may process the SNACK option by first retransmitting the segments corresponding to the Hole1 specification. If a bit-vector is present, the data-sender may left-shift the bit vector until the

last “1” is shifted out, retransmitting the segment corresponding to each “0” encountered in the process. (This is an example for illustrative purposes, the exact mechanism is implementation specific. It is STRONGLY RECOMMENDED that retransmissions occur in the order of ascending sequence numbers.)

4.5 SCPS-TP HEADER COMPRESSION SPECIFICATION

SCPS Header Compression is to be used on SCPS-TP TCP connections that require high bit-efficiency. The requirement for high bit-efficiency may result from the presence of low-data-rate links in one or both directions of the communication path, or from high utilization of those links. For most space missions, link bandwidth is considered a scarce resource, and the overhead of TCP segment headers is considered too high.

A significant amount of work has been done outside the SCPS activity to reduce the overhead of TCP/IP headers. This work is documented in RFC 1144. The compression specified in RFC 1144 is intended for use on low-speed serial links, and addresses the problems of maintaining interactive response for programs such as telnet. RFC 1144 header compression operates at the link layer. The link layer maintains connection state tables for inbound and outbound TCP/IP connections; the state for each connection consists of the last TCP and IP header sent (outbound) or received (inbound) on that connection. The compressor is initialized by allocating a connection number to the connection and saving the first TCP/IP header sent. The initial headers are not compressed. Subsequent headers are encoded by sending only the changes from the previous header. Additionally, the sequence number, acknowledgment number, urgent pointer, and window fields are sent as *changes* to the previous value. At the receiving side, an uncompressed TCP/IP header is created by applying the changes to the saved header to create a new TCP/IP header. This new header is saved in the connection state table and is forwarded to the destination along with the data. Since the information in the compression state tables will be corrupted if a segment is lost or damaged (misordering is typically not a problem at the link layer), the (unmodified) TCP checksum is included in all TCP segments. If the decompressor state is corrupted, the TCP checksum will fail at the receiving TCP endpoint and the corrupted segment will be discarded. Retransmissions are forwarded uncompressed by the RFC 1144 compressor, and are used to resynchronize the decompressor’s state. (Note that if a segment is lost or corrupted, *all segments following it will be decompressed improperly, causing them to be discarded by the receiving TCP endpoint.* This behavior continues until the sending TCP entity retransmits the lost segment(s), resynchronizing the compressor and decompressor. This is typically not a problem for RFC 1144 operation, since it is designed primarily for interactive operation in which there are typically only a few octets of data outstanding at one time.)

The remainder of the discussion of SCPS Header Compression is structured in the following manner: first, we discuss the problems with using RFC 1144 Header Compression in the SCPS environment; second, we describe the approach used in solving those problems; finally, we provide the details of SCPS Header Compression.

4.5.1 PROBLEMS WITH RFC 1144 HEADER COMPRESSION IN THE SCPS ENVIRONMENT

The header compression algorithm to be used with the SCPS TCP is derived from the algorithm specified in RFC 1144. There are problems operating the header compression specified in RFC 1144 in a SCPS environment, so changes were necessary.

A major problem with RFC 1144 in the SCPS environment results from the way it encodes and decodes data. RFC 1144 uses delta encoding, which encodes data in terms of the amount of change since the previous transmission. If a segment is lost or misordered, the decompression algorithm incorrectly decompresses all segments until the compression state at the receiver is resynchronized via retransmission. (Retransmissions are sent uncompressed.) The recovery from the incorrect decompression is accomplished via the normal TCP retransmission mechanisms, which means that the RFC 1144 forces go-back-n retransmission once it loses synchronization. Go-back-n retransmission is not a significant problem in a low-speed serial environment, because the bandwidth-delay product of the network is low, and little data will be outstanding at any time. As data rates increase, the data loss due to decompression errors increases. The behavior of the RFC 1144 decompressor when faced with loss or resequencing is unacceptable in a SCPS environment, because of the bit-inefficiency of the go-back-n algorithm in moderate to high bandwidth-delay product environments.

The SCPS networking environment includes satellite constellations that are changing their link-layer connectivity on a frequent basis. This subverts the RFC 1144 header compression algorithm, which maintains transport state at the link layer, assuming that the connection is a low-speed serial (phone) line from a personal computer to a gateway or a remote host. RFC 1144 compression maintains TCP state by tracking changes from one TCP segment to the next, and resynchronizes by sending an uncompressed TCP/IP datagram whenever the link-layer state is corrupted. Link-layer state becomes corrupted whenever a packet is lost (due to lower-layer error detection or to congestion) or is misordered. In a dynamic connectivity environment, that link-layer state would continually be out of date, and resynchronization would happen frequently. Since RFC 1144 uses normal TCP retransmission mechanisms to resynchronize, in addition to a possible deterioration in bit efficiency, the TCP connection would be hampered by the throughput reductions associated with frequent retransmission.

Finally, RFC 1144 assumes that IP is the Network-layer protocol. This allows compression of the IP header as well as the TCP header, but forces the compression algorithm to operate at the link layer. The SCPS effort has addressed Network-layer overhead by defining the SCPS Network Protocol (SCPS-NP).

Note that much of the text of this section, particularly the discussions of Compressor Processing, Decompressor Processing, and Error Handling, are adapted with only minor modifications from RFC 1144. While those modifications are significant, and substantially affect the ability of the compression software to operate in the SCPS environment, the SCPS

project is indebted to Van Jacobson for his thorough explanation of the original processes, which made adaptation much easier.

4.5.2 SCPS HEADER COMPRESSION—APPROACH

Like RFC 1144, SCPS Header Compression uses two main techniques to reduce the size of TCP headers: It *summarizes* information that is static for the duration of the connection (by assigning a connection identifier to replace the port numbers). It *omits* information that is not relevant to the segment being sent (such as the urgent field if the URG flag is not set). Unlike RFC 1144, it does not use delta encoding, nor does it operate at the link layer. The following paragraphs discuss how SCPS Header Compression addresses the problems identified with the use of RFC 1144 in the SCPS environment.

SCPS Header Compression attempts to address the loss and mis-sequencing problems of the RFC 1144 algorithm by operating end-to-end within the Transport layer. That is, Network-layer header compression is left to the SCPS Network Protocol, and the TCP connection implements the Transport-layer-specific compression and decompression operations. This allows the resequencing mechanisms within TCP to be used to avoid go-back-n retransmission behavior and is unaffected by changes in connectivity.

The use of SCPS Header Compression is requested by the initiating TCP endpoint by including the SCPS Capabilities option with its (uncompressed) SYN segment. If the initiating endpoint does not include this option, compression will not be performed on the connection. The responding TCP endpoint accepts the offer to perform SCPS Header Compression by including its own SCPS Capabilities option with its SYN ACK segment. Refer to 4.1.3 for a detailed discussion of the SCPS Capabilities option.

To prevent ambiguity resulting from possible misordering of segments, the full TCP sequence number must accompany any data-carrying segments. A similar ambiguity exists with acknowledgments, which are not reliably sent. In addition, the value of the window field references a particular acknowledgment number to define the upper window edge. When the possibility of loss or misordering exists, it is important that the window advertisement and the acknowledgment number that it references not be separated. Therefore, when acknowledgments are sent, the window size and acknowledgment number are both specified in the compressed header.

The SCPS Header Compression algorithm supports “piggy-backing” of acknowledgments on data-carrying segments (as in uncompressed TCP), but *it is an implementation option* whether to exercise this ability. The compressor for a particular implementation may send acknowledgments (and other information not directly related to the data being transferred) separately from the data in order to ensure a constant header size for data-carrying segments. (This can aid in packing fixed-length lower-layer frames when bulk data is to be transferred.) In accordance with the robustness principle stated in RFC 1122 and in TCP, a decompressor must be prepared to accept piggy-backed acknowledgments even if the compressor in that implementation does not generate them. (The robustness principle roughly states “be

generous in what you accept and conservative in what you send," and is intended to promote interoperability.)

The compressed SCPS header contains three mandatory fields and several optional fields. The mandatory fields are the connection identifier, a bit-field indicating what optional fields are present in the compressed header, and a checksum. (The checksum covers the compressed header, the user data, and the TCP pseudo-header. This differs from the RFC 1144 approach, in which the checksum is the unmodified TCP checksum, therefore covering all missing fields of the TCP header.) The bit-field also carries several flags (such as the Push flag) that do not correspond to fields in the compressed header.

There are two main implementation approaches for SCPS Header Compression. The first approach compresses TCP headers after they have been fully generated, and decompresses the headers into TCP headers for receive-side processing. This approach is easiest to retrofit onto an existing TCP implementation, but may not be the most efficient. The second approach generates SCPS Compressed Headers in lieu of generating a TCP header, and operates on the compressed header directly on the receive side. This approach can improve processing efficiency if an implementation is being developed "from scratch". Specifically, the second approach has direct information regarding whether the intent of a particular segment is data transfer or acknowledgment. With the first approach, the presence or absence of user data determines whether this segment is intended for data transfer, and a test for changes in the acknowledgment number indicates whether an acknowledgment has been "piggy-backed."

NOTE – the description of compressor and decompressor processing that follows in subsequent sections is written from the perspective of the first implementation approach.

SCPS Compressed Headers are differentiated from TCP headers by use of different protocol numbers in the Transport Protocol Identifier (TP-ID) field of the SCPS-NP. (Alternative methods for differentiating these packet types exist if the SCPS-NP is not being used: for example, two CCSDS Path IDs may be allocated to differentiate between compressed and uncompressed segments.) When the SCPS-NP is in use, TP-ID 6 identifies uncompressed TCP and TP-ID 5 identifies compressed segments. Refer to the SCPS-NP specification for a full description of SCPS-NP Transport Protocol Identifiers.

Upon receipt, a packet with a TP-ID indicating Compressed TCP (or an APID indicating the same) is routed to the decompressor (in the first implementation approach). The checksum is verified and a TCP segment is reconstructed using the data in the compressed header. Fields missing in the compressed header are supplied from the receiver's TCP state or from a previously reconstructed TCP segment. The reconstructed TCP segment enters the regular TCP input processing stream after the TCP checksum computation.

4.5.3 COMPRESSED HEADER FORMAT

The Compressed Packet format is illustrated in figure 4-9. The header begins with a one-octet connection identifier that is established during the exchange of the SCPS Compression Option between endpoints, as described above.

The second octet of the compressed header is a bit-vector that identifies the contents of the compressed header. This bit-vector may, in some cases, extend to the third octet of the header. When the "More" bit, the MSB of octet 2, is set to one there is a second octet of bit-vector information present. The contents of the bit-vector are described in detail in table 4-1.

The fields in figure 4-9 that are shown with dashed outlines (after the first octet of the bit-vector but before the checksum) are only included when necessary. Their presence or absence is indicated by the corresponding bits in the bit-vector. In the figure, each optional field shows three elements of information: the bit of the bit-vector that signals its presence; the name of the field; and the length of the field, in octets. These are shown in the format "Bit: Name (Length)".

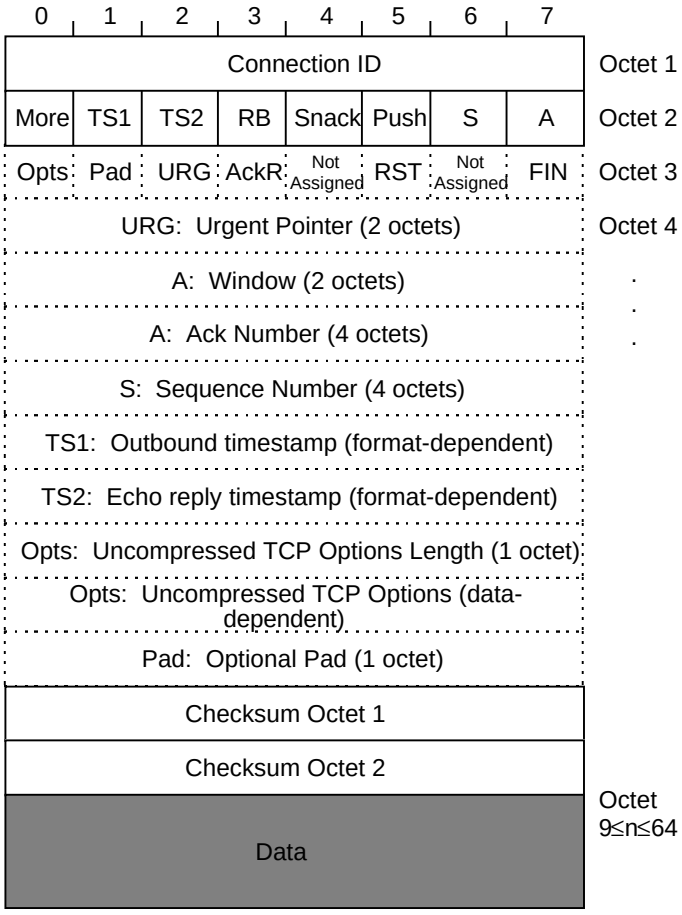


Figure 4-9: Compressed SCPS-TP Header

The two-octet urgent pointer field is included if the URG flag is set in the TCP header. The window and acknowledgment fields carry the unmodified window and acknowledgment fields from the TCP header if an acknowledgment update is being sent.

Table 4-1: Compressed Header Bit-Vector Contents

Bit Name	Meaning when set to "1"	Notes
More	Compressed Header Bit-Vector is 16 bits long rather than 8 bits long.	
TS1	TCP Timestamp Option is present.	See SCPS-TP 6.2.2.5
TS2	A timestamp reply (TS Echo Reply) appears in the compressed header.	See SCPS-TP 6.2.2.5
RB	The last octet of data accompanying this segment is the end of a user-defined record.	See SCPS-TP 3.3.1
P	The Push bit from the uncompressed TCP header is set.	
S	The compressed header contains a 4-octet sequence number.	
A	The compressed header contains a 2-octet window specification and a 4-octet acknowledgment number.	
Opts	The compressed header contains uncompressed options.	
Pad	The compressed header contains one octet of padding.	
URG	The URG bit from the uncompressed TCP header is set.	
AckR	The ACK bit from the uncompressed TCP header is set (this field is only valid when the RST bit is set).	
RST	The RST bit from the uncompressed TCP header is set.	
FIN	The FIN bit from the uncompressed TCP header is set.	

The sequence number field carries the unmodified sequence number field from the TCP header, and is present if the segment is retransmittable (i.e., user data is included or the FIN flag is set).

The TS1 field indicates that a timestamp value to be echoed is present, either as part of the Network-layer service indication or as part of the compressed header. If the SCPS Capabilities option negotiation indicated that NL Ts were *not* available, the timestamp is located in the compressed header.

The TS2 field indicates that an echo reply timestamp value is present in the compressed header.

The uncompressed TCP options length field indicates the length of any TCP options that accompany the compressed header without themselves being compressed. The uncompressed TCP options field contains the value of any TCP options that have not been compressed. (The Record Boundary flag, the TS1 and TS2 flags, and the Snack flag exist in the compressed header for the purpose of compressing TCP options.)

The Pad flag indicates whether the compressed header has optionally been padded with a zero-value octet at the end of the compressed header. This bit is optionally set by a transmitter to ensure that user data starts on an even byte boundary. Its use is implementation specific. (Note that if there are no other bits set in the second octet of the change mask, the Pad flag is NOT set to accomplish padding; rather, the MORE flag is set and the second control octet is left at a value of zero.)

The final field in the Compressed Header is the checksum field. This field is computed using the standard TCP checksum algorithm and covers the contents of the compressed header, the user data, and the TCP pseudo-header.

4.5.4 COMPRESSOR PROCESSING

NOTE – This description is written based on a compressor that receives fully formed TCP segments and translates their headers into compressed format. This is not the only approach to header compression. A more efficient method is to encode the compressed headers directly, without the intermediate step of generating the uncompressed headers. The transformation approach is presented because it lends itself to simple explanation.

The compressor is invoked with the TCP segment and with access to the TCP state for the connection to which the segment belongs.

The compressor decides how to format a compressed TCP segment in the following manner:

If the URG flag in the TCP header is set, the urgent data field is copied into the Compressed header immediately following the bit-vector; the URG bit is set in the bit-vector.

If either the window value or the acknowledgment number has changed from the last segment sent on this connection, both are copied into the Compressed header and the “A” bit is set in the bit-vector.

If data accompanies the segment or the FIN flag is set in the uncompressed header (or both), the sequence number is copied into the Compressed header and the “S” bit is set in the bit-vector.

If any of the PUSH, FIN, or RST flags are set in the original TCP header, the corresponding bits are set in the bit-vector. If the "RST" bit is set, the "AckR" bit is set to the value of the ACK bit in the uncompressed TCP header.

If the original TCP header contains options, the OPTS bit is set in the change mask. (If all options are compressible, the OPTS bit may be subsequently reset.) The length of the uncompressed options is initialized to the length of the options in the TCP header.

The options are examined to determine if any of the following options are present: the Record Boundary option, the Timestamp option, the NOP (no operation) option, or EOL (end of list).

All NOP and EOL options are discarded, and the length of the options to be carried in the uncompressed options portion of the compressed header is decremented by their length.

If the Record Boundary option is present, the "RB" bit is set in the bit-vector and the length of the uncompressed options is decremented by the length of the Record Boundary option (two octets).

If the Timestamps option is present, the "TS1" flag is set and if timestamps are *not* carried end-to-end by the Network layer, the timestamp value is copied to the compressed header. (If timestamps are carried by the Network layer, the timestamp is passed to the Network layer in the lower-layer service call and not carried in the compressed header.) The "TS2" flag is set and the echo reply portion of the Timestamps option is copied into the compressed header.

If the length of the remaining options is greater than zero, this length is copied into the compressed header and the remaining options are copied into the compressed header.

If the remaining options' length is zero, the OPTS flag in the bit-vector is reset.

If the second octet of the bit-vector is zero (no bits set), the "MORE" bit in the first octet is cleared and the second octet does not appear in the compressed header. If the length of the compressed header is odd the compressor may optionally pad the header to an even length. If the second octet of the bit-vector is not present, it may be reinstated to pad the header. If the second octet of the bit-vector is present, a zero-value octet may be added to the end of the compressed header and the "PAD" flag may be set in the bit-vector.

At this point, an Internet checksum is computed over the compressed header, data, and TCP pseudo header, and written into the compressed header. The compressed TCP segment (consisting of the compressed header and any user data accompanying the original segment) is passed to the Network layer with the TP-ID value of Compressed TCP.

4.5.5 DECOMPRESSOR PROCESSING

NOTE – as with the description of Compressor Processing, this description assumes a decompressor that is only "loosely coupled" to the TCP implementation. That is, the decompressor creates uncompressed TCP segments, rather than acting on the field information directly. This is for ease of explanation only.

The decompressor uses the previous packet received on the connection as the means of filling in fields that are not present in the compressed header. As a result, the decompressor retains the (uncompressed) header of the previous packet received on the connection as a source for these fields.

The decompressor receives packets of type "Compressed TCP" from the Network layer, along with relevant Network-layer information, such as the length of the packet, incoming source timestamp, and the source and destination addresses. The output of the decompressor is a TCP segment, for immediate processing by TCP or insertion into the out-of-sequence queue.

The decompressor uses the Connection Identifier and network addresses to find the TCP endpoint with which this packet should be associated. If the endpoint is not located, the segment is discarded and an error is logged.

Once the connection has been identified, the information from the TCP pseudo header is used to verify the checksum in the compressed packet. If the checksum fails, the packet is discarded, and an error is logged (triggering the corruption response in the TCP endpoint).

The first octet of the bit-vector is read and the most-significant bit is tested to determine if a second octet of bit-vector is present. If so, it is read; if not, a zero value is assumed.

The decompressor creates a template of a TCP header, initializing the port information to the ports corresponding to the current TCP connection. All other fields are initialized to zero.

The flags from the control mask are used to set the flags fields in the reconstructed header: If PUSH, URG, RST, or FIN are set, their corresponding flags are set in the reconstructed header. The ACK flag is set to one unless the "RST" bit is set in the bit-vector, in which case the ACK flag is set to the value of the "AckR" bit of the bit-vector. The SYN flag is reset (since all SYN segments are used to resynchronize the compressor/decompressor pair).

The bits in the bit-vector are interpreted individually in the order that the compressor set them:

– If the URG bit is set, the next two octets of the incoming packet are decoded and copied into the TCP Urgent Pointer field.

- If the “A” bit is set, the next two octets of the compressed header are decoded and copied into the window field of the reconstructed header. Then, the 32-bit acknowledgment number is copied from the compressed header to the reconstructed header.

- If the “S” bit is set, the 32-bit sequence number is copied from the compressed header to the reconstructed header.

- If the “RB” bit is set in the bit-vector, a record boundary option is added to the reconstructed header.

- If the “TS1” and “TS2” bits are set in the bit-vector, a TCP Timestamps option is generated. The TCP endpoint is tested to determine whether end-to-end NL Ts are in use. If so, the TCP Timestamps option is generated using the NL T for the TS Value field of the option (see RFC 1323). If NL Ts are not in use, the contents of the TS Value field are copied from the compressed header. The contents of the TS Echo Reply field is then copied from the compressed header. (Note that by having separate “TS1” and “TS2” bits in the compressed header, the equivalent of RFC 1072 timestamps may be compressed, if desired.)

- If the OPTS bit is set in the change mask, the uncompressed TCP length is read from the compressed header, and the uncompressed TCP options are copied to the reconstructed header.

- The reconstructed TCP header is padded, if necessary, to a multiple of 32-bits by using NOP options, and the resulting header length is encoded into the header.

At this point, all header information from the incoming compressed TCP segment has been consumed, and only data remains. The reconstructed header is saved for use in decompressing subsequent compressed packets. The reconstructed segment consisting of the reconstructed header and the data is processed as usual by TCP.

4.6 MULTIPLE TRANSMISSIONS FOR FORWARD ERROR CORRECTION

The Multiple Forward Transmissions (MFX) capability provides a form of forward error correction for TCP connections that invoke it. MFX causes the sending TCP entity to send data on a connection multiple times without waiting for retransmission timeouts or retransmission requests from the remote TCP entity.

The anticipated use for the MFX capability is in very long delay environments, or in environments in which the availability of an acknowledgment channel is infrequent. The significant factor that makes MFX attractive in this environment is that the feedback from the remote TCP entity, if any, is untimely. This same untimeliness affects the validity of congestion control information: congestion-related information received from the remote

TCP entity will probably *not* reflect the current state of the network, for the purposes of closed-loop congestion control. As a result, it is anticipated that the MFX capability will be employed only when closed-loop congestion control (such as Van Jacobson's congestion control scheme or the TCP Vegas scheme) is *not* in use. Current congestion control schemes such as those mentioned use progression of the acknowledgment number as an indication of packets having arrived successfully at the destination, without having experienced congestion. When packets are intentionally duplicated, these algorithms are unable to determine whether congestion loss has occurred or not, since if any of the multiple transmissions are received the acknowledgment number advances. Therefore, the MFX capability *should not* be used on a connection that also employs either Van Jacobson congestion control or TCP Vegas congestion control.

The TSU invokes the MFX capability for a particular connection by changing the "MFX" parameter for that connection from its default value of one. When this value is greater than one, all segments that require acknowledgment (that is, those that carry user data, or the SYN flag, or the FIN flag) shall be transmitted the number of times that the MFX parameter indicates. Multiple forward transmissions shall not count as retransmissions (with respect to the R1 and R2 thresholds - refer to section 4.2.2 of this document). Note that "pure acknowledgments", those that do not carry data, or the SYN flag, or the FIN flag, are not transmitted multiple times. The remote TCP entity is not informed that the MFX capability has been enabled for a connection. Rather, the normal TCP protocol mechanisms detect and discard any duplicate segments that might be received as a result of the multiple transmissions.

If it is necessary to retransmit a segment (as a result of retransmission timeout, or SNACK, or fast retransmit), the retransmission shall be multiply transmitted. This multiple transmission shall be counted as a single retransmission for the purposes of R1 and R2 accounting.

Successive segment transmissions by a TCP entity may be interleaved with transmissions of subsequent segments. Such interleaving is an implementation issue, and is not specified here.

5 SCPS-FP USER'S GUIDE

This section defines the minimum user implementation and the service interfaces for the following classes of SCPS-FP services:

- Configuration;
- Access;
- File Transfer;
- Record Access;
- Interrupt, Abort, and Restart;
- File Operations;
- Miscellaneous Services.

In the context of this section, a "user" may be a human or an application program. The remainder of this section documents the user interface for each user command, examples of each command, and high-level SCPS-FP protocol actions associated with each user command.

5.1 MINIMUM USER IMPLEMENTATION

The minimum implementation is defined below under REQUIRED. Other alternative implementations are listed under the OPTIONAL headings.

5.1.1 REQUIRED

Configuration	timeout, help, rhelp, autorestart, noautorest, numautor, suppress, unsuppress
Access	open, quit
File Transfer	get, put
Record Access	raread, raupdt
Interrupt, Abort, Restart	Ctrl-C, Ctrl-Y, restart
File Operation	size, delete
Miscellaneous	site

5.1.2 OPTIONAL BASED ON UNDERLYING FILE SYSTEM/OPERATING SYSTEM CAPABILITIES

The following commands SHOULD be supported by every SCPS Server-FTP and User-FTP on systems whose underlying file system and/or operating system supports a directory structure:

Configuration	<none>
Access	<none>
File Transfer	<none>
Record Access	<none>
Interrupt, Abort, Restart	<none>
File Operation	ls, mkdir, rmdir, cd
Miscellaneous	<none>

5.1.3 OPTIONAL BASED ON MISSION NEEDS

The following commands SHOULD be supported based on the needs of the mission.

Configuration	type, struct, mode, sendport, unsendport, status, rstatus, bets, nobets
Access	user
File Transfer	proxy
Record Access	<none>
Interrupt, Abort, Restart	<none>
File Operation	rename
Miscellaneous	<none>

If a command cannot be implemented due to the lack of file system, operating system and/or system resource capabilities, the FP application should notify the user that the particular service is not recognized.

For each of the above commands, the user may use upper case, lower case or a combination of both to enter the commands. All filenames may be specified with or without full pathnames. If no pathname is specified, the current working directory is accessed for the specified service.

5.1.4 CONFIGURATION SERVICES

This section describes the service interfaces for the following configuration services:

- Set data transfer type;
- Set data structure;
- Set transmission mode;
- Enable/Disable use of PORT command for each transfer;
- Enable/Disable reply text suppression;
- Enable/Disable autorestart;
- Set maximum number of automatic restarts;

- Display configuration status;
- Set idle timeout value for server;
- Request help on FP commands;
- Enable/Disable Best Effort Transport Service Option.

Recommended user/client and server configurations are also provided.

5.1.5 DEFAULT USER/CLIENT CONFIGURATION

Below are the recommended default settings for the user configuration of a client (user) SCPS-FP:

FEATURE	DEFAULT SETTING
Mode	Stream
Type	Image
Structure	File
Autorestart	Off
Use PORT	On
Max Number of Automatic Restarts	3
BETS	Off
BETS Fill Code	255

5.1.6 DEFAULT SERVER CONFIGURATION

Below are the recommended default settings for the configuration of a server SCPS-FP:

FEATURE	DEFAULT SETTING
Mode	Stream
Type	Image
Structure	File
Suppress Reply Text	On
Idle Timeout (in Seconds)	300
Autorestart	Off
BETS	Off
BETS Fill Code	255

5.1.7 SET DATA TYPE (TYPE)

With the TYPE configuration option, the user can specify the data type to be used during the file transfer: ASCII or binary. Data types are documented in RFC 959, section 3.1.1. The data type remains as set by the user until the user logs out or another TYPE command is issued. On re-login or re-connect, the data type reverts to the default configuration.

The user must have established a connection to the server to use this command.

5.1.7.1 USER INTERFACE SPECIFICATION

TYPE <file type>
where <file type> = A for ASCII
I for binary (image)

5.1.7.2 EXAMPLES

Example 1: Set data type to ASCII.

sfp> TYPE A <enter>

Example 2: Set data type to binary.

sfp> TYPE I <enter>

5.1.7.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The data types that must be supported by SCPS-FP are documented in 3.4.1 of the SCPS-FP protocol specification. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-Protocol-Interpreter (User-PI):

- user command;
- type specified by user.

In response to the user command TYPE, the User-PI saves the data type locally and sends the protocol command TYPE to the Server-PI via the control connection.

In turn the Server-PI saves the data type locally if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol spec) via the control connection.

The User-PI forwards the server's reply to the User-Interface process, which forwards it to the user. The User-Interface process waits for the next user command.

5.1.8 SET DATA STRUCTURE (STRUCT)

With the STRUCT configuration option, the user can specify the type of data structure to be used during the transfer. Data structures are documented in RFC 959, section 3.1.2. The data structure remains as set by the user until the user logs out or another STRUCT command is issued. On re-login or re-connect, the data structure reverts to the default configuration.

The user must have established a connection to the server to use this command.

5.1.8.1 USER INTERFACE SPECIFICATION

STRUCT <structure type>

where <structure type> = F (file)

5.1.8.2 EXAMPLES

Example 1: Set data structure type to 'file'.

sfp> STRUCT F <enter>

5.1.8.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The data structures that must be supported by SCPS-FP are documented in 3.4.1 of the SCPS-FP protocol specification. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- data structure specified by the user.

In response to the user command STRUCT, the User-PI saves the data structure type locally and sends the protocol command STRU to the Server-PI via the control connection.

In turn the Server-PI saves the data structure type locally if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol spec) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.1.9 SET TRANSMISSION MODE (MODE)

With the MODE configuration option, the user can specify the mode of transmission to be used during the transfer. Modes are documented in RFC 959, section 3.4. The transmission mode remains as set by the user until the user logs out or another MODE command is issued. On re-login or re-connect, the transmission mode reverts to the default configuration.

The user must have established a connection to the server to use this command.

5.1.9.1 USER INTERFACE SPECIFICATION

MODE <mode>

where <mode> = S (stream)

5.1.9.2 EXAMPLES

Example 1: Set mode to stream.

sfp> MODE S <enter>

5.1.9.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The transmission modes that must be supported by SCPS-FP are documented in 3.4.1 of the SCPS-FP protocol specification. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- transmission mode specified by user.

In response to the user command MODE, the User-PI saves the transmission mode locally and sends the protocol command MODE to the Server-PI via the control connection.

In turn the Server-PI saves the transmission mode locally if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol spec) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.1.10 ENABLE USE OF PORT COMMAND (SENDPORT)

With the SENDPORT configuration option, the user indicates that the PORT command should be used automatically with each file transfer. In this situation, the user-process selects a value for its data port rather than using its default data port. Per RFC 959, section 3.2, the user-process default data port is the same as the control connection port. The FP session remains in the SENDPORT state until the FP session is terminated or the UNSENDPORT command is issued.

The user does NOT need a connection to the server to use this command.

5.1.10.1 USER INTERFACE SPECIFICATION

SENDPORT

5.1.10.2 EXAMPLES

Example 1: Set configuration so PORT command always used.
sfp> SENDPORT <enter>
"Use of PORT command is on."

5.1.10.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

user command.

In response to the user command SENDPORT, the User-PI enables the "send port" state flag, saves it for access by a file transfer command later, and notifies the user, via the User-Interface process, that a PORT command will be generated for each subsequent file transfer. This command does NOT cause a PORT command to be sent to the server at this time. The

"send port" state flag remains enabled until the FP session is terminated or the UNSENDPORT command is issued.

The User-Interface process waits for the next user command.

5.1.11 DISABLE USE OF PORT COMMAND (UNSENDPORT)

With the UNSENDPORT configuration option, the user indicates that the PORT command should NOT be used automatically for each file transfer. In this situation the user-process default data port is used for the file transfer. Per RFC 959, section 3.2, the user-process default data port is the same as the control connection port. The FP session remains in the UNSENDPORT state until the FP session is terminated or the SENDPORT command is issued.

The user does NOT need a connection to the server to use this command.

5.1.11.1 USER INTERFACE SPECIFICATION

UNSENDPORT

5.1.11.2 EXAMPLES

Example 1: Set configuration so PORT command is NOT used automatically.
sfp> UNSENDPORT <enter>
"Use of PORT commands is off."

5.1.11.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

user command.

In response to the user command UNSENDPORT, the User-PI disables the "send port" state flag, saves it for access by a file transfer command later, and notifies the user, via the User-Interface process, that a PORT command will no longer be generated automatically for subsequent file transfers. The "send port" state flag will remain disabled until the FP session is terminated or the SENDPORT command is issued.

The User-Interface process waits for the next user command.

5.1.12 ENABLE REPLY TEXT SUPPRESSION (SUPPRESS)

With the SUPPRESS configuration option, the user may configure the server so that the server sends the reply code WITHOUT the ASCII reply text. This may be desirable when the reply text is not needed and a reduction in control message overhead is desired. The FP session remains in the suppression state until the user logs out or the UNSUPPRESS command is issued. On re-login or re-connect, the suppression state reverts to the default configuration.

The user must have established a connection to the server to use this command.

5.1.12.1 USER INTERFACE SPECIFICATION

SUPPRESS

5.1.12.2 EXAMPLES

Example 1: Enable reply text suppression:
sfp> suppress <enter>
"211 Reply Text Suppression Enabled"
Most replies (all except those with message that are parsed by the User-FTP) would come back with the reply code only. For example,
"200 <CRLF>" rather than "200 Command Okay<CRLF>"

5.1.12.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

user command.

In response to the user command SUPPRESS, the User-PI sends the protocol command SUPP to the Server-PI via the control connection.

In turn the Server-PI enables the "suppress reply text" state flag if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol spec) via the control connection. The Server-PI must suppress the reply *text* for subsequent protocol commands with the following exceptions: PASV, INTR, SIZE. The replies for these commands must not be suppressed because they provide information to the user and/or User-PI that is needed in order to continue processing.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

The "suppress reply text" state flag remains enabled until the user logs off the server or the UNSUPPRESS command is issued. On re-login, the "suppress reply text" state reverts to the default configuration.

5.1.13 DISABLE REPLY TEXT SUPPRESSION (UNSUPPRESS)

With the UNSUPPRESS configuration option, the user may configure the server so that the server sends the reply code WITH the ASCII reply text. This is used to "undo" reply suppression. The FP session remains in the non-suppression state until the user logs out or the SUPPRESS command is issued. On re-login or re-connect, the suppression/non-suppression state reverts to the default configuration.

The user must have established a connection to the server to use this command.

5.1.13.1 USER INTERFACE SPECIFICATION

UNSUPPRESS

5.1.13.2 EXAMPLES

Example 1:
Disable reply text suppression:
sfp> unsuppress <enter>
"211 Reply Text Suppression Disabled"
All replies will now come back with the reply code and reply text.

5.1.13.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

user command.

In response to the user command UNSUPPRESS, the User-PI sends the protocol command NSUP to the Server-PI via the control connection.

In turn the Server-PI disables the "suppress reply text" state flag if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP

protocol spec) via the control connection. The reply text for each command should be included in the reply.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

The "suppress reply text" state flag remains disabled until the user logs off the server or the SUPPRESS command is issued. On re-login, the "suppress reply text" state reverts to the default configuration.

5.1.14 ENABLE AUTORESTART (AUTORESTART)

With the AUTORESTART configuration option, the user may enable automatic restart of file transfer and specify whether the user should be requested to confirm the restart. If autorestart is enabled and a file transfer encounters a connection error, the file protocol will automatically try to restart the transfer at the point of error. The user does not need to specify where to restart. On re-login or re-connect, the autorestart/no autorestart state reverts to the default configuration.

When autorestart is disabled and a file transfer aborts for any reason, the Server-PI and User-PI should rollback changes. When autorestart is enabled and a file transfer aborts, the partially received file is saved using the given name—changes must not be rolled back.

The user must have established a connection to the server to use this command.

5.1.14.1 USER INTERFACE SPECIFICATION

AUTORESTART

5.1.14.2 EXAMPLES

Example 1: Enable autorestart:
sfp> autorestart <enter>
"200 ARST Command Successful"

5.1.14.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

user command.

In response to the user command AUTORESTART, the User-PI enables the “autorestart” state flag, saves the flags for access by a file transfer command later, and sends the protocol command ARST to the Server-PI via the control connection.

In turn the Server-PI enables its autorestart flag if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol spec) via the control connection.

The User-Interface process waits for the next user command. This command does NOT cause an automatic restart of a file transfer to occur at this time.
When autorestart is enabled and an automatic interrupt occurs, the user should be queried to determine if he/she wants to continue with the autorestart.

5.1.15 DISABLE AUTORESTART (NOAUTOREST)

With the NOAUTOREST configuration option, the user may disable automatic restart of file transfers. If autorestart is disabled and a file transfer encounters a connection error, the file transfer is NOT restarted. On re-login or re-connect, the autorestart/no autorestart state reverts to the default configuration.

The user must have established a connection to the server to use this command.

5.1.15.1 USER INTERFACE SPECIFICATION

NOAUTOREST

5.1.15.2 EXAMPLES

Example 1: Disable autorestart:
sfp> noautorest <enter>
“200 NARS Command Successful”

5.1.15.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

user command.

In response to the user command NOAUTOREST, the User-PI disables the “autorestart” state flag, saves the flag for access by a file transfer command later, and sends the protocol command NARS to the Server-PI via the control connection.

In turn the Server-PI disables its autorestart flag if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol spec) via the control connection.

5.1.16 CONFIGURE NUMBER OF AUTORESTARTS (NUMAUTOR)

With the NUMAUTOR configuration option, the user can specify the maximum number of times a failed file transfer should be automatically restarted. For example if a file transfer fails and autorestart is enabled, the file protocol restarts the transfer until it succeeds or the NUMAUTOR value is reached, which ever comes first. If the transfer succeeded on the nth restart and another failure occurs later during the same FP session, autorestart is applied up to (NUMAUTOR - n) times or until the file transfer succeeds, which ever comes first. The maximum number of restarts remains as set by the user until the FP session is terminated or another NUMAUTOR command is issued.

The user does NOT need a connection to the server to use this command.

5.1.16.1 USER INTERFACE SPECIFICATION

NUMAUTOR <decimal-integer>
where <decimal-integer> is greater than 0.

5.1.16.2 EXAMPLES

Example 1: Set maximum number of automatic restarts
sfp> numautor 5 <enter>
“Max restart count: 5 (when autorestart enabled).”

5.1.16.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

– user command;

– maximum number of automatic restarts specified by user.

In response to the user command NUMAUTOR, the User-PI updates and saves the maximum number of automatic restarts for access by a file transfer command later and notifies the user, via the User-Interface process, that the maximum restart count has been changed. The User-Interface process waits for the next user command.

The maximum number of autorestart tries will remain as specified by the user until the FP session is terminated or the NUMAUTOR command is issue again.

5.1.17 DISPLAY CONFIGURATION STATUS OF REMOTE/LOCAL SYSTEM (STATUS)

With the STATUS configuration commands, the user may display the current status of the configuration options and the connections.

For the local status, the user does NOT need a connection to the server to use this command. For the remote, the user must have established a connection to the server to use the command.

5.1.17.1 USER INTERFACE SPECIFICATION

For client (local) process: STATUS

For server (remote) process: RSTATUS

5.1.17.2 EXAMPLES

Example 1: Display configuration status of the client

```
sfp> status <enter>
" Connected.
  Mode: stream
  Type: Image
  Structure: file
  Autorestart: enabled; Max Number of Automatic Restarts: 3
  Use PORT command: on"
```

Example 2: Display configuration status of the server
sfp> rstatus <enter>
" 211-System Status.
211-Connected.
211-Mode: stream
211-Type: Image
211-Structure: file
211-Reply Suppression: enabled
211-BETS: disabled
211 Autorestart: enabled"

5.1.17.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

user command.

5.1.17.3.1 Local System

In response to the user command STATUS, the User-PI forwards the User-FP's current configuration and connection settings to the User-Interface process. The User-Interface process forwards the response to the user and then waits for the next user command.

5.1.17.3.2 Remote System

In response to the user command RSTATUS, the User-PI sends the protocol command STAT to the Server-PI via the control connection.

In turn the Server-PI sends the Server-FP's configuration and connection settings to the User-PI as part of the server reply. An appropriate reply is sent via the control connection, as specified in RFC 959, section 4. The User-PI forwards the server's reply with current configuration and connection settings to the User-Interface process. The User-Interface process waits for the next user command.

5.1.18 CONFIGURE IDLE TIMEOUT PARAMETER (TIMEOUT)

With the TIMEOUT configuration option, the user can specify how long the server be idle before timing out. The idle timeout remains as set by the user until the FP session is terminated or another TIMEOUT command is issued.

The user must have established a connection to the server to use this command.

5.1.18.1 USER INTERFACE SPECIFICATION

TIMEOUT <decimal-integer>

where <decimal-integer> is greater than 0 and represents the number of seconds that should pass before a timeout occurs.

5.1.18.2 EXAMPLES

Example 1: Set server's idle timeout to 5 mins (5*60 = 300 seconds)
sfp> timeout 300 <enter>
"200 IDLE Command Successful"

5.1.18.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- idle timeout specified by user.

In response to the user command TIMEOUT, the User-PI sends the protocol command IDLE to the Server-PI via the control connection.

In turn the Server-PI set the idle timeout locally. An appropriate reply is sent via the control connection, as specified in 3.3 of the SCPS File Protocol specification. The User-PI forwards the server's reply to the User-Interface process. The User-Interface process waits for the next user command.

5.1.19 REQUEST COMMAND HELP FROM REMOTE/LOCAL SYSTEM (HELP)

With the HELP command, the user may request help on the FP commands. Help will list those commands that are active.

For the local help, the user does NOT need a connection to the server to use this command. For the remote help, the user must have established a connection to the server to use the command.

5.1.19.1 USER INTERFACE SPECIFICATION

For client (local) process: HELP [<string>]

For server (remote) process: RHELP [<string]

5.1.19.2 EXAMPLES

Example 1: Request help from client.
sfp> help <enter>
"GET OPEN
PUT QUIT
TIMEOUT TYPE
HELP STRUCT"

Example 2: Request help from the server
sfp> rhelp <enter>
"214- RETR STOR IDLE HELP
QUIT TYPE STRUCT PORT
214"

Example 3: Request help from the server for specific command.
sfp> rhelp RETR <enter>
"214 Syntax RETR: <sp> file-name"

5.1.19.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

- user command;
- argument string (usually a specific command), if present.

5.1.19.3.1 Local System

In response to the user command HELP, the list of active FP user commands is forwarded to the user if no argument is provided. If an argument is provided, any help on the argument is forwarded to the user. The User-Interface process then waits for the next user command.

5.1.19.3.2 Remote System

In response to the user command RHELP, the User-PI sends the protocol command HELP to the Server-PI via the control connection.

In turn the Server-PI sends the Server-FP's help as part of the server reply. The help response may be a list of all active server FP commands or help on a specific FP command, if the user provided a command name with RHELP. An appropriate reply is sent via the control connection, as specified in RFC 959, section 4. The User-PI forwards the server's reply to the User-Interface process which then waits for the next user command.

5.1.20 ENABLE BEST EFFORT TRANSPORT SERVICE (BETS)

The Best Effort Transport Service (BETS), an optional service of the SCPS-TP that is not available in Commercial-Off-The-Shelf TCP, provides a data transfer service that guarantees correct and in-sequence data delivery, but possibly with gaps. With this configuration option, the user enables the BETS option and optionally specifies a fill code to fill any gaps in the data. BETS is enabled for the FP session but only applies to Get and Put. If BETS is enabled during a file transfer and a gap is encountered in the data, the file protocol will fill the gaps with the user-specified fill code or the default fill code if the user does not specify one. On re-login or re-connect, the BETS/NOBETS state reverts to the default configuration.

The user must have established a connection to the server to use this command.

This option is not interoperable with Commercial-Off-The-Shelf FTP/TCP/IP clients or servers.

5.1.20.1 USER INTERFACE SPECIFICATION

BETS {<BETS-fill-code>}

where <BETS-fill-code> ::= a decimal integer in the range of 0 to 255
<BETS-fill-code> is an optional argument representing an ASCII character.

5.1.20.2 EXAMPLES

Example 1: Enable BETS:
sfp> bets <enter>
"211 Best Effort Transport Service Enabled with Fill Code 255"

Example 2: Enable BETS with a fill character:
sfp> bets 0 <enter>
"211 Best Effort Transport Service Enabled with Fill Code 0"

5.1.20.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

- user command;
- BETS fill code.

In response to the user command BETS, the User-PI enables the BETS state flag and sets the BETS fill code as indicated, saves this information for access by a file transfer command later, and sends the protocol command BETS to the Server-PI via the control connection.

In turn the Server-PI enables its BETS flag and saves the BETS fill code if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS File Protocol specification) via the control connection.

The User-Interface process waits for the next user command.

5.1.21 DISABLE BEST EFFORT TRANSPORT SERVICE OPTION (NOBETS)

With the NOBETS configuration option, the user specifies that gaps in the data are not allowed. On re-login or re-connect, the BETS/NOBETS state reverts to the default configuration.

The user must have established a connection to the server to use this command.

5.1.21.1 USER INTERFACE SPECIFICATION

NOBETS

5.1.21.2 EXAMPLES

Example 1: Disable BETS:
sfp> nobets <enter>
"211 Best Effort Transport Service Disabled"

5.1.21.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI:

user command.

In response to the user command NOBETS, the User-PI disables the BETS state flag, saves the flag for access by a file transfer command later, and sends the protocol command NBES to the Server-PI via the control connection.

In turn the Server-PI disables its BETS flag if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS File Protocol specification) via the control connection.

5.2 ACCESS SERVICES

This subsection describes the service interfaces for the following access services:

- start up the SCPS-FP application;
- open connection to server;
- log user into server;
- close connection to server;
- quit file protocol.

5.2.1 START UP SCPS-FP APPLICATION (SFTP)

With the SFTP access command, the user can start up the SCPS-FP application and if a server is specified, open a control connection to the server.

5.2.1.1 USER INTERFACE SPECIFICATION

SFTP
or
SFTP <server name>

where <server name> is the name of the server with which to establish a connection.

5.2.1.2 EXAMPLES

Example 1: Start up the SCPS-FP application
SFTP <enter>
"sfp>"

Example 2: Start up the SCPS-FP application and open a control connection to the server SERVER1.
SFTP SERVER1 <enter>
"Connected to SERVER1.
220 SERVER1 SFTP Server Ready.
sfp>"

5.2.1.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If no arguments are present, the User-Interface displays the FP application prompt and waits for the next user command.

If an argument is present, the User-Interface process provides the following information to the User-PI:

server name.

In response to the user application SFTP being initiated with a server name, the User-PI initiates a connect request to the designated server.

After the connection has been established and the user has been notified of its establishment, the User-Interface process displays the FP application prompt and waits for the next user command.

5.2.2 OPEN CONNECTION TO SERVER (OPEN)

With the OPEN access command, the user can open a control connection to a specified server. The Server-PI must not open a new non-proxy connection if an old one is still open.

5.2.2.1 USER INTERFACE SPECIFICATION

OPEN <server name>

where <server name> is the name of the server with which to establish a connection.

5.2.2.2 EXAMPLES

Example 1: Start up the SCPS-FP application and open a control connection to the server SERVER1.
sfp> OPEN SERVER1 <enter>
"Connected to SERVER1.
220 SERVER1 SFTP Server Ready.
sfp>"

5.2.2.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- server name.

In response to the user command OPEN, the User-PI issues a connect request for the designated server via SCPS-TP.

Upon receipt of the request, the Server-PI responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS File Protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. At this time the User-Interface process may automatically prompt the user for a username and password.

5.2.3 LOG USER INTO SERVER (USER)

With the USER access command, the user can log into a server for which there is already a connection. Typically, the user need not use this command because the user is automatically prompted for a username and password after a control connection is established with the server. This command may also be used to change the user login to another user.

5.2.3.1 USER INTERFACE SPECIFICATION

USER

or

USER <login name at server>

5.2.3.2 EXAMPLES

Example 1: Log in as 'Smith' on the currently open server.
sfp> USER smith <enter>
"331 Password required for smith"
"Password: " <pass word is entered>
"230 User smith logged in."

5.2.3.3 INTERACTION WITH PROTOCOL

If there are no arguments to USER, then the User-Interface process prompts the user for a username. The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- username.

In response to the user command USER, the User-PI sends the protocol command USER to the Server-PI via the control connection.

In turn the Server-PI validates the username against access list and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS File Protocol specification) via the control connection.

The User-PI forwards the server reply to the User-Interface process which forwards it to the user. The User-Interface process queries the user for password and gives the password to the User-PI.

The User-PI sends the protocol command PASS to the Server-PI via the control connection.

The Server-PI determines if the password is correct, logs user into the server if the password is correct, and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS File Protocol specification) via the control connection.

The User-PI forwards the server reply to the User-Interface process which forwards it to the user. The User-Interface process waits for next user command.

5.2.4 CLOSE CONNECTION TO SERVER (CLOSE)

With the CLOSE user command, the user can close the control connection to a currently open server, thereby logging the user out. The SCPS-FP application remains up and ready for the next command. This is an optional command.

5.2.4.1 USER INTERFACE SPECIFICATION

CLOSE

or

CLOSE <server-name>

5.2.4.2 EXAMPLES

Example 1: Close connection.
sfp> CLOSE <enter>

5.2.4.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- server to close (either specified by user or defaults to current connection).

In response to the user command CLOSE, the User-PI sends the protocol command QUIT to the designated Server-PI via the control connection.

In turn the Server-PI responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS File Protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-PI issues a disconnect request for the server to SCPS-TP. The FP application is NOT terminated.

5.2.5 QUIT FILE PROTOCOL (QUIT)

With the QUIT access command, the user can close all open connections and quit the SCPS-FP application.

5.2.5.1 USER INTERFACE SPECIFICATION

QUIT

5.2.5.2 EXAMPLES

Example 1: Quit SCPS-FP
sfp> QUIT <enter>

5.2.5.3 INTERACTION WITH PROTOCOL

The User-Interface receives the user command and provides the following information to the User-PI: user command.

In response to the user command QUIT, the User-PI sends the protocol command QUIT to the Server-PI via the control connection.

In turn the Server-PI responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS File Protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-PI issues a disconnect request for the server to SCPS-TP. The FP application is terminated.

5.3 FILE TRANSFER SERVICES

This subsection describes the service interfaces for the following file transfer services:

- retrieve a file;
- store a file;
- initiate proxy file transfer (retrieve or store).

5.3.1 RETRIEVE FILE (GET)

With the GET file transfer command, the user can request and receive a file that is stored at a remote system (server). The GET user command takes one or two pathname arguments. If the second pathname is not supplied, the GET user command uses the given pathname for both arguments.

An implementation may trigger special processing based on the second argument. For example, if the second argument is a dash "-", an implementation might send the file to the standard output device (usually the screen). If the second argument is a percent symbol "%", an implementation might send the file to the default printer.

If the server supports login access control, the user must be logged in to use this command.

5.3.1.1 USER INTERFACE SPECIFICATION

GET <remote_src_file> <local_dest_file>

where <remote_src_file> ::= <pathname>
<local_dest_file> ::= <pathname>
<pathname> ::= <string>

<pathname> represents the location of the file data. For a file system-based storage system, <pathname> would be a filename with possibly a directory specification. For a memory-based storage system (i.e., no file system), <pathname> would take the form of one of the following:

<address>:<file-length> /for source file

(e.g., 1:500 = at location 1,
length of 500)
<address> /for destination file

The <address> and <file-length> are still specified in ASCII form.

The specification of <local_src_file> is only needed if the user wants to store the file locally with a different name or display the data to the screen.

5.3.1.2 EXAMPLES

Example 1: Retrieve yourfile.txt and store locally with name same
sfp> GET yourfile.txt <enter>

Example 2: Retrieve yourfile.txt and store locally as myfile.txt.
sfp> GET yourfile.txt myfile.txt <enter>

Example 3: Retrieve yourfile.txt and display contents.
sfp> GET yourfile.txt - <enter>

Example 4: Retrieve file a byte 1 for length 500 and store local at byte 300.
sfp> GET 1:500 300 <enter>

5.3.1.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- name of remote file;
- name of local file (if not specified by user, use remote file name);
- whether or not data should be displayed.

The User-PI forwards the following information to the User-DTP:

- name of local file (if not specified by user, use remote file name);
- whether or not data should be displayed.

If the 'send port' state flag is enabled, the User-PI computes a data PORT for the user and sends the protocol command PORT to the Server-PI via the control connection. The Server-PI saves the PORT information and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS File Protocol specification) via the control connection. The User-PI forwards the server reply to the User-Interface process which forwards it to the user.

If the BETS state flag is enabled, the User-DTP and the Server-DTP request the BETS option from the transport service.

The User-DTP listens on data port (the default data port, if SENDPORT was not issued and PORT was not specifically issued by the user). The User-PI sends the protocol command RETR to the Server-PI via the control connection.

The Server-DTP opens a data connection to user's data port and responds to the User-PI via the control connection with a preliminary reply as specified in 3.3 of the SCPS File Protocol specification.

Data connection establishment and management for the User-DTP and the Server-DTP is performed as specified in 3.2 of the SCPS File Protocol specification.

The Server-DTP accesses the local data file and sends its contents to User-DTP via the data connection.

Upon receipt of the server's preliminary reply, the User-PI forwards the server reply to the User-Interface process which forwards it to the user. The User-DTP then begins accepting the data sent by the Server-DTP (concurrent to the server sending data). If the BETS state flag is enabled and a gap in the data is detected as the data is received, the User-DTP fills in the gap with the BETS fill code for the length of the gap. Because the sending Server-FP may not be aware of any gaps in the data that may appear at the User-FP, it is the responsibility of the User-FP to notify the user of any gaps in the data and whether the gaps were filled. All data is stored locally or written to the screen as specified by the user. If the data is to be stored locally and the file already exists, the file is overwritten with the new data.

If an error is detected on either the data connection or the control connection and the 'autorestart' state flag is enabled, the file transfer will be restarted automatically as outlined in 3.2.2 of the SCPS-FP protocol specification.

Upon completion of the transfer or on the occurrence of an error, the Server-PI closes the data connection and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS File Protocol specification) via the control connection.

The User-PI forwards the server's final reply to the User-Interface process which forwards it to the user. The User-Interface process waits for next user command.

NOTE – A representative scenario of a GET with autorestart is provided in 5.8 below.

5.3.2 STORE FILE (PUT)

With the PUT file transfer command, the user can store a file at a remote system (server).

If the server supports login access control, the user must be logged in to use this command.

5.3.2.1 USER INTERFACE SPECIFICATION

PUT <local_src_file> <remote_dest_file>

where

<remote_dest_file> ::= <pathname>
<local_src_file> ::= <pathname>
<pathname> ::= <string>
<pathname> represents the location of the file data. For a file system-based storage system, <pathname> would be a filename with possibly a directory specification. For a memory-based storage system (i.e., no file system), <pathname> would take the form of one of the following:

<address>:<file-length> /for source file
(e.g., 1:500 = at location 1,
length of 500)
<address> /for destination file

The <address> and <file-length> are still specified in ASCII form.

The specification of <remote-dest-file> is needed only if the user wants to store the file with a different name (or location) at the server.

5.3.2.2 EXAMPLES

Example 1: Send myfile.txt and store at the server with name same.
sfp> PUT myfile.txt <enter>

Example 2: Send myfile.txt and store at the server as yourfile.txt.
sfp> PUT myfile.txt yourfile.txt <enter>

Example 3: Send file at byte 100 for length 200 and store remote at byte 300.
sfp> PUT 100:200 300 <enter>

5.3.2.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- local file name;
- remote file name (same as local filename, if none specified by user).

The User-PI forwards the following information to the User-DTP:

name of local file.

If the 'send port' state flag is enabled, the User-PI computes a data PORT for the user and sends the protocol command PORT to the Server-PI via the control connection. The Server-PI saves the PORT information and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection. The User-PI forwards the server reply to the User-Interface process which forwards it to the user.

If the BETS state flag is enabled, the User-DTP and the Server-DTP request the BETS option from the transport service.

The User-DTP listens on data port (the default data port if SENDPORT was not issued and PORT was not specifically issued by the user). The User-PI sends the protocol command STOR to the Server-PI via the control connection.

The Server-DTP opens a data connection to user's data port and responds to the User-PI via the control connection with a preliminary reply as specified in 3.3 of the SCPS-FP protocol specification.

Data connection establishment and management for the User-DTP and the Server-DTP is performed as specified in 3.2 of the SCPS-FP protocol specification.

Upon receipt of the server's preliminary reply, the User-PI forwards the server reply to the User-Interface process which forwards it to the user. The User-DTP then begins sending the local file contents to the Server-DTP via the data connection. If the file already exists, the file is overwritten with the new data.

If an error is detected on either the data connection or the control connection and the 'autorestart' state flag is enabled, the file transfer is restarted automatically as outlined in 3.2.2 of the SCPS-FP protocol specification.

The Server-DTP then begins accepting the data sent by the User-DTP (concurrent to the user-DTP sending data. If the BETS state flag is enabled and a gap in the data is detected as the data is received, the Server-DTP fills in the gap with the BETS fill code for the length of the gap. All data is stored locally.

Upon completion of the transfer or on the occurrence of an error, the Server-PI closes the data connection and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's final reply to the User-Interface process which forwards it to the user. The User-Interface process waits for next user command.

NOTE – A representative scenario of a PUT with autorestart is provided in 5.8 below.

5.3.3 INITIATE PROXY FILE TRANSFER (PROXY)

With the PROXY transfer command, the user can initiate a data transfer between two remote systems via the control of one local system. Control connections are established between the local system and remote system A and between the local system and remote system B. The data connection is established between remote system A and remote system B.

In order to use this command, the user must have opened connections to two servers. The last connection opened is the active server. The first one is the passive server. If a server supports login access control, the user must be logged in to use this command.

5.3.3.1 USER INTERFACE SPECIFICATION

PROXY GET <remote-src-file> <local-dest-file>
or
PROXY PUT <local-src-file> <remote-dest-file>

<remote-src-file>, <local-dest-file>, <local-src-file>, and <remote-dest-file> are as defined for the GET and PUT commands.

5.3.3.2 EXAMPLES

Example 1:
Send a file from Server2 to Server1 via proxy transfer.
sfp> open Server1 <enter>
sfp> open Server2 <enter>
sfp> proxy put myfile.txt <enter>

Example 1:
Retrieve a file from Server1 to Server2 via proxy transfer.
sfp> open Server1 <enter>
sfp> open Server2 <enter>
sfp> proxy get myfile.txt <enter>

5.3.3.3 INTERACTION WITH PROTOCOL

In order for this command to work, the user must have opened control connections between the client and remote server “a” and between the client and remote server “b”. For the purposes of this discussion, it is assumed that the connection to remote server “a” was made first and the connection to remote server “b” second. This makes server “a” the passive server and server “b” the active (current) server.

The User-Interface process parses the user command and its arguments. If no arguments are specified, the User-Interface process should prompt the user for the command (and its arguments) that should be performed in proxy. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

user command to issue in proxy and its parameters.

The User-PI sends the protocol command PASV to the Server-“a”-PI via the control connection.

The Server-“a”-PI determines its data port, includes it in the server reply, and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server reply to the User-Interface process which forwards it to the user. The User-PI extracts the port information from the PASV reply and uses it as the argument to the PORT command. The User-PI sends the protocol command PORT to the Server-“b”-PI via the control connection.

The Server-“b”-PI saves Server-“a’s” data port and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI sends the protocol command RETR (on a GET) and STOR (on a PUT) to the Server-“a”-PI via the control connection.

The User-PI sends the protocol command STOR (on a GET) and RETR (on a PUT) to the Server-“b”-PI via the control connection.

The following Server-“a” and Server-“b” actions occur concurrently:

The Server-“a”-DTP listens on its data port. The Server-“a”-DTP responds to the User-PI via the control connection with a preliminary reply as specified in RFC 959, section 4.

Upon receipt of server “a’s” preliminary reply, the User-PI forwards the server reply to the User-Interface process which forwards it to the user.

The Server-“b”-DTP opens a data connection to Server-“a”-DTP data port (which it got via the PASV/PORT command sequences) and responds to the User-PI via the control connection with a preliminary reply as specified in 3.3 of the SCPS-FP protocol specification.

Data connection establishment and management for the Server-“a”-DTP and the Server-“b”-DTP is performed throughout the section as specified 3.2 of the SCPS-FP protocol specification.

Upon receipt of server “b’s” preliminary reply, the User-PI forwards the server reply to the User-Interface process which forwards it to the user.

For a PUT command, the Server-“b”-DTP sends it file contents to the Server-“a”-DTP via the data connection and the Server-“a”-DTP concurrently receives the file data and stores it locally.

For a GET command, the Server-“a”-DTP sends its file contents to the Server-“b”-DTP via the data connection and the Server-“b”-DTP concurrently receives the file data and stores it locally.

Upon completion of the transfer or on the occurrence of an error, the Server-“b”-PI closes the data connection and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection. The Server-“a”-PI also responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the servers’ final replies to the User-Interface process which forwards it to the user. The User-Interface process waits for next user command.

5.4 RECORD ACCESS SERVICES

This subsection describes the service interfaces for the following record access services:

- retrieve file records;
- update file records.

5.4.1 RETRIEVE FILE "RECORDS" (RAREAD)

The RAREAD user command is used to transfer records of a remote file, rather than a complete file, to the user. The record read user command includes the following user-specified options.

- change directory option
- display option
- forced read option

Each of these options is discussed below.

The change directory option is used to automatically change to a remote working directory if a directory path is specified in the remote filename. A directory path shall be deemed to be present if either a backslash "\" or forward slash "/" appear in the filename. If a directory path is specified in the remote filename of the record read *user command*, the User-PI shall issue a change directory SCPS-FP command (CWD) to the Server-FTP to effect a directory change at the Server-FTP before issuing a record read SCPS-FP command (READ) to the Server-FTP. The current working directory remains as set here for the rest of the FP session until another CWD command is executed or the user logs out.

The display option is used by the user to write the specified records to a computer screen rather than store the records in a local file. If a dash "-" is specified for the local file name, the record data shall be displayed to the screen.

The forced read option is used to ignore recoverable errors and then to continue the record read request. The user is willing to except potentially corrupt or incomplete data. If this option is selected, the user and server processes continue until the read transfer is deemed completed or an irrecoverable error occurs.

If the server supports login access control, the user must be logged in to use this command.

5.4.1.1 USER INTERFACE SPECIFICATION

raread <SP> <remote-path-src> <SP> <local-path-dest> <SP> <record-id-spec> <SP> <forced-read-option>

where

<pathname> ::= <string>
<pathname> represents the location of the file data. For a file system-based storage system, <pathname> would be a filename with possibly a directory specification. For a memory-based storage system (i.e., no file system), <pathname> would take the form of one of the following:

<address>:<file-length> /for source file
(e.g., 1:500 = at location 1,
length of 500)
<address> /for destination file

The <address> and <file-length> are still specified in ASCII form.

<remote-path-src> ::= name of file (with or without full pathname) to read from remote system.
::= <pathname>
<remote-path-src-contents> ::= 1{<record>}many
<local-path-dest> ::= name of file (with or without full pathname) at the local system in which to store the records read from the remote system OR a dash to signal that data should be written to screen.
::= <pathname>
<local-path-dest-contents> ::= 1{<record>}many
<record> ::= <octet> | <CCSDS-Pkt-Record>
<CCSDS-Pkt-record> ::= CCSDS Packet

<record-id-spec> ::= <record-id-range> | <record-id> 1{,<record-id-range> | <record-id>} 9
** no spaces allowed
** record id's do not have to be in numerical order
** <record-id-end> must be greater than <record-id-start>

<record-id-range> ::= <record-id-start>-<record-id-end>
<record-id-start> ::= <record-id>
<record-id-end> ::= <record-id> | <eof>

<eof> ::= string 'EOF', upper and/or lower case permitted

<record-id> ::= Numeric ID of the "record" to access; corresponds an octet number for file structure and to the CCSDS Packet Sequence Count field of a CCSDS Packet for CCSDS Packet structures
::= <decimal-integer>

** <record-id> of zero (0) for IMAGE type indicates the first octet of the file

<forced-read-option>::= option to continue read retrieval event if an error is encountered
::= Y | y | N | n ; Default = N; Optional

5.4.1.2 EXAMPLES

Example 1:
Extract records 1, 2, 5, 10, 11 from “myfile.xyz” and write the records to “localfile” at the client:

sfp> raread myfile.xyz localfile 1,10,11,5,2 <enter>

Example 2:
Extract record 4 from “myfile.xyz” (after changing to directory /directory1/directory2) and display data directly to user rather than write to local file:

sfp> raread /directory1/directory2/myfile.xyz - 4 <enter>

Example 3:
Extract records 4-10, 16-22, and 32 from “myfile.xyz”, write the records to “localfile” at the client and ignore recoverable errors on read.

sfp> raread myfile.xyz localfile 16-22,32,4-10 y <enter>

Example 4:
Extract records 4-10 and all records from 60 through the end of the file, write the records to “localfile” at the client.

sfp> raread myfile.xyz localfile 4-10,60-eof <enter>

Example 5:
Extract records 4-10 and all records from 60 through the end of the file, write the records locally at the client. Use addresses for the filenames.

sfp> raread 50:2000 500 4-10,60-eof <enter>

(Remote file starts at memory 50 and goes for length 2000. Local file storage should start at memory location 500.)

5.4.1.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. This includes checking the validity of the record ranges. If an EOF string appears in the ranges, the string should be converted to the number 4294967295. (Note: Nearly all C environments have a macro ULONG_MAX which yields that number.) The server will trigger on that number and read to the end of the file.

If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

– user command;

- name or pathname of remote file;
- name or pathname of local file in which to store records;
- list of valid ranges of records (with EOF range converted);
- flag for forced-read-option;
- whether or not records should be displayed.

The User-PI forwards the following information to the User-DTP:

- name of local file in which to store data;
- whether or not records should be displayed.

If the 'send port' state flag is enabled, the User-PI computes a data PORT for the user and sends the protocol command PORT to the Server-PI via the control connection. The Server-PI saves the PORT information and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection. The User-PI forwards the server reply to the User-Interface process which forwards it to the user.

If the user specified a pathname from which to access the remote file, the User-PI extracts the pathname from the filename and sends the protocol command CWD to the Server-PI via the control connection. The Server-PI changes to the current working directory to that specified and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection. The User-PI forwards the server reply to the User-Interface process which forwards it to the user.

The User-DTP listens on data port (the default data port if SENDPORT was not issued and PORT was not specifically issued by the user).

The User-PI prepares the control data. The User-PI sends the protocol command READ to the Server-PI via the control connection.

The Server-DTP opens a data connection to user's data port and responds to the User-PI via the control connection with a preliminary reply as specified in 3.3 of the SCPS-FP protocol specification.

Data connection establishment and management for the User-DTP and the Server-DTP is performed as specified in 3.2 of the SCPS-FP protocol specification.

Upon receipt of the server's preliminary reply, the User-PI forwards the server reply to the User-Interface process which forwards it to the user. The User-DTP then begins sending the control data to the server-DTP.

The Server-DTP then begins accepting the control data sent by the User-DTP (concurrent to the user-DTP sending control data) and stores the data locally. After the Server-DTP has received all of the control data, it keeps the data connection open as specified in 3.2.1.3 of the

SCPS-FP protocol specification. The Server-DTP reads the remote file and extracts the selected ranges of records. Then it starts sending the record data to the User-DTP.

The User-DTP begins accepting the data sent by the Server-DTP (concurrent to the server sending data) and stores the data locally or displays it to a computer screen, as specified by the user.

Upon completion of the record read or on the occurrence of an error, the Server-PI closes the data connection and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's final reply to the User-Interface process which forwards it to the user. The User-Interface process waits for next user command.

NOTE – A representative scenario of a RAREAD transaction is provided in 5.8 below.

5.4.2 UPDATE FILE "RECORDS" (RAUPDT)

The RAUPDT user command is used to update octets of a remote file, rather than retransfer a complete updated file, to the server. This command relies on the user to provide the record update data which consists of a series of signals that indicate which octets to delete and modify in the remote file and where to add new data. The updates are made against a copy of the original remote file and the resulting modified data is written to a new remote file.

The record update user command includes one user-specified option. The change directory option is used to automatically change to a remote working directory if a directory path is specified in the remote filename. A directory path shall be deemed to be present if either a backslash "\" or forward slash "/" appear in the filename. If a directory path is specified in the remote filename of the record read *user command*, the User-PI shall issue a change directory SCPS-FP command (CWD) to the Server-FTP to effect a directory change at the Server-FTP before issuing a record update SCPS-FP command (UPDT) to the Server-FTP. The current working directory remains as set here for the rest of the FP session until another CWD command is executed or the user logs out.

If the server supports login access control, the user must be logged in to use this command.

5.4.2.1 USER INTERFACE SPECIFICATION

raupdt <remote-path-src> <SP> <remote-path-dest> <SP> <local-path-src> <SP> <local-path-dest> <SP><update-file-path-src>

where <pathname> ::= <string>

<pathname> represents the location of the file data. For a file system-based storage system, <pathname> would be a filename with possibly a directory specification. For a memory-based storage system (i.e., no file system), <pathname> would take the form of one of the following:

<address>:<file-length> /for source file
(e.g., 1:500 = at location 1,
length of 500)
<address> /for destination file

The <address> and <file-length> are still specified in ASCII form.

<remote-path-src> ::= name of file (with or without full pathname) to use as source file for updates on remote system
::= <pathname>
<remote-path-dest> ::= name of file (with or without full pathname) in which to store the updated source file on remote system
::= <pathname>
<local-path-src> ::= name of file (with or without full pathname) to use as test source file for update (theoretically it should be a file with the same contents as <remote-path-src>)
::= <pathname>
<local-path-dest> ::= name of file (with or without full pathname) in which to store the locally updated source file.
::= <pathname>
<update-file-path-src> ::= name of file (with or without full pathname) in which update signals are stored locally.
::= <pathname>
<remote-path-src-contents>::= 1{<binary-record>}many |
1{<CCSDS-Pkt-record>}many

<remote-path-dest-contents> ::= 1<binary-record>}many |
1{<CCSDS-Pkt-record>}many

<local-path-src-contents> ::= 1{<binary-record>}many |
1{<CCSDS-Pkt-record>}many

<local-path-dest-contents> ::= 1{<binary-record>}many |
1{<CCSDS-Pkt-record>}many

<octet> ::= 1 8-bit byte
<binary-record> ::= <octet>
<CCSDS-Pkt-record> ::= CCSDS Packet

<data> ::= 1{<binary-record>}<record-count> |
1{<CCSDS-Pkt-record>}<record-count>

<record-count> ::= Corresponds to the number of octets for file structures and to the number of CCSDS Packets for CCSDS Packet structures

::= <decimal-integer>

<record-id> ::= Numeric ID of the "record" to access; corresponds an octet number for file structure and to the CCSDS Packet Sequence Count field of a CCSDS Packet for CCSDS Packet structures

::= <decimal-integer>

NOTE – <record-id> zero (0) when in IMAGE type indicates the first octet of the file.

<update-file-path-src-contents> ::= <update-data>
<update-data> ::= 1{<update-signal>}many
<update-signal> ::= <delete-signal><record-id><record-count> |
<change-signal-id><record-id><record-count><data>
<delete-signal> ::= signal to delete record(s) whose start is specified by <record-id>
::= d
<change-signal-id> ::= <addafter-signal> |
<insertbefore-signal> |
<writeover-signal>
<addafter-signal> ::= signal to add (insert) records after the record specified by <record-id>
::= a
<insertbefore-signal> ::= signal to add (insert) records before the record specified by <record-id>
::= i
<writeover-signal> ::= signal to write over (replace) records starting at the record specified by <record-id>
::= w

NOTE – The record range associated with a update signal shall not overlap with the record range of any another update signal.

5.4.2.2 EXAMPLES

Example 1:
Update "myfile" with updates in "diff_file" and store in "myfile.new" on the server. Update is first tested against "cfile" and stored in "cfile_new" at the local machine:
sfp> raupdt myfile myfile.new cfile cfile_new diff_file <enter>

Example 2:

After changing to the directory “dir1”, update “myfile” with updates in “diff_file” and store in “myfile.new” on the server. Update is first tested against “cfile” and stored in “cfile_new” at the local machine:

sfp> raupdt /dir1/myfile /dir2/myfile.new cfile cfile_new diff_file <enter>

Example 3:

An example of update data for a record update request follows. The target file happens to be an ASCII text file.

Original Data:

This is line 1<LF>
This is line 2<LF>
This is line 3<LF>
This is line 4<LF>
This is line 5<LF>
This is line 6<LF>
This is line 7<LF>
Maryx had a little<LF>
This is line 9<LF>
This is line 10<LF>

Desired Data:

this is the first line to add<LF>
this is the second line to add<LF>
This is line 1<LF>
This is line 2<LF>
This is line 3<LF>
This is line 4<LF>
Mary had a little lamb.<LF>
This is line 9<LF>
This is line 10<LF>

Update Data:

**Note in this example the 32-bit numbers (file offset) is designated by a 8 digits in the number, the 16-bit numbers (length) are designated by 4 digit numbers. File offset always precedes length. e.g. 000000000061 designates file offset = 0, length = 61.

i000000000061this is the first line to add<LF>this is the second line to add<LF>d000000600045w000001050018Mary had a little<SP>i000001230005lamb.

Example 4:

Update “myfile” with updates in “diff_file” and store in “myfile.new” on the server. Update is first tested against “cfile” and stored in “cfile_new” at the local machine. For this example we assume that both client and server have no file system and therefore use memory addresses to locate and update data.
sfp> raupdt 300:1000 200 1300:1000 500 5400:200 <enter>
In this example,
300:1000 is “myfile” at memory location 300 for length 1000
200 is “myfile.new” to be stored at memory location 200
1300:1000 is “cfile” at memory location 1300 for length 1000
500 is “cfile_new” to be stored memory location 500
5400:200 is “diff_file” at memory location 5400 for length 200

5.4.2.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- name or pathname of original remote file;
- name or pathname of new remote file;
- name or pathname of local original remote file;
- name or pathname of new local file;
- name or pathname of local file containing update data.

The User-PI forwards the following information to the User-DTP:

name of local file containing update data.

The user-PI ensures that the update data are sorted by offset in ascending order.

The User-PI tests the update against the local original file using the local update data. If an error occurs during the test update, the User-PI notifies the User-Interface process of the error. The User-Interface process forwards the error to the user.

If the ‘send port’ state flag is enabled, the User-PI computes a data PORT for the user and sends the protocol command PORT to the Server-PI via the control connection. The Server-PI saves the PORT information and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection. The User-PI forwards the server reply to the User-Interface process which forwards it to the user.

If the user specified a pathname from which to access the remote file, the User-PI extracts the pathname from the filename and sends the protocol command CWD to the Server-PI via the control connection. The Server-PI changes to the current working directory to that specified and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection. The User-PI forwards the server reply to the User-Interface process which forwards it to the user.

The User-DTP listens on data port (the default data port if SENDPORT was not issued and PORT was not specifically issued by the user).

The User-PI prepares the control data. The User-PI sends the protocol command UPDT to the Server-PI via the control connection.

The Server-DTP opens a data connection to user's data port and responds to the User-PI via the control connection with a preliminary reply as specified in 3.3 of the SCPS-FP protocol specification.

Data connection establishment and management for the User-DTP and the Server-DTP is performed as specified in 3.2 of the SCPS-FP protocol specification.

Upon receipt of the server's preliminary reply, the User-PI forwards the server reply to the User-Interface process which forwards it to the user. The User-DTP then begins sending the control data to the server-DTP.

The Server-DTP then begins accepting the data sent by the User-DTP (concurrent to the user-DTP sending data) and stores the data locally. After the Server-DTP has received all of the control data, it performs the update against a copy of the designated file using the update data provided in the control data. After sending the control data, the user-DTP closes the data connection to indicate the end of the control data to the server-DTP. Upon completion of the update or on the occurrence of an error, the server-PI responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's final reply to the User-Interface process which forwards it to the user. The User-Interface process waits for next user command.

NOTE – A representative scenario of a RAUPDT transaction is provided in 5.8 below.

5.5 INTERRUPT, ABORT, RESTART SERVICES

This subsection describes the service interfaces for the following interrupt, abort, and restart services:

- manually interrupt file transfer;

– manually abort file transfer;
– manually restart file transfer.

5.5.1 MANUAL INTERRUPT (Ctrl-Y)

With the Ctrl-Y command, the user can interrupt a file transfer in such a manner that the user may restart the file transfer by issuing RESTART followed by the point of restart. Manual interrupt will be recognized only if a file transfer is in progress.

5.5.1.1 USER INTERFACE SPECIFICATION

Ctrl-Y

5.5.1.2 EXAMPLES

When a file transfer is in-progress and the user would like to interrupt the operation, the user should press the Ctrl and Y keys down to interrupt the transfer.

5.5.1.3 INTERACTION WITH PROTOCOL

In response to the Ctrl-Y key sequence, the User-PI stops the User-DTP processing and sends the protocol command INTR to the Server-PI via the control connection.

In turn the Server-PI stops the Server-DTP processing, determines the interrupt point of the file transfer if the request is error free, and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

If the transfer that was in progress was a user-to-server transfer, the User-PI forwards the server's reply to the User-Interface process which forwards it to the user. If the transfer that was in progress was a server-to-user transfer, the User-PI will replace the interrupt point in the reply with that stored at the User-FP and then forward the server's modified reply to the User-Interface process which forwards it to the user.

The User-Interface process waits for the next user command.

NOTE – A representative scenario of an interrupted file transfer is provided in 5.8 below.

5.5.2 MANUAL ABORT (Ctrl-C)

With the Ctrl-C command, the user can terminate a file transfer before completion. A manually abort file transfer cannot be restarted by the user. Manual abort will be recognized only if a file transfer is in progress.

5.5.2.1 USER INTERFACE SPECIFICATION

Ctrl-C

5.5.2.2 EXAMPLES

When a file transfer is in-progress and the user would like to cancel the operation, the user should press the Ctrl and C keys down to abort the transfer.

5.5.2.3 INTERACTION WITH PROTOCOL

In response to the Ctrl-C key sequence, the User-PI stops the User-DTP processing and sends the protocol command ABOR to the Server-PI via the control connection.

In turn the Server-PI stops the Server-DTP processing and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

NOTE – A representative scenario of an aborted file transfer is provided in 5.8 below.

5.5.3 MANUALLY RESTART FILE TRANSFER (RESTART)

With the RESTART command, the user can restart a previously interrupted file transfer. To begin a restart, the user specifies the restart command with a restart marker and then follows the restart command with the original transfer command (e.g., get, put) that was interrupted.

If the server supports login access control, the user must be logged in to use this command.

5.5.3.1 USER INTERFACE SPECIFICATION

RESTART <restart marker>

where <restart marker> is the point as which to restart the file transfer.

When prompted the user follows up with the GET or PUT command that was interrupted.

5.5.3.2 EXAMPLES

Example 1: A GET on file myfile.txt was interrupted at byte 54.
Restart the file transfer at this point.
sfp> RESTART 54 <enter>
"350 Send GET command"
GET myfile.txt

5.5.3.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- restart point.

In response to the user command RESTART, the User-PI saves the restart point locally for use during the file transfer and sends the protocol command REST to the Server-PI via the control connection.

In turn the Server-PI saves the restart point locally for use during the file transfer if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

NOTE – The file transfer is not restarted until the file transfer command (e.g., GET or PUT) is resent.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.6 FILE OPERATION SERVICES

This subsection describes the service interfaces for the following file operation services:

- rename a file;
- delete a file;
- list directory contents;
- create directory;
- delete directory;
- change working directory;
- get size of file.

5.6.1 RENAME A FILE (RENAME)

With the RENAME file operation command, the user can rename a file stored on a remote system.

If the server supports login access control, the user must be logged in to use this command.

5.6.1.1 USER INTERFACE SPECIFICATION

RENAME <from-name> <to-name>

where <from-name> is the current name of the file
<to-name> is the new name of the file.

5.6.1.2 EXAMPLES

Example 1: Rename file myfile.txt to myfile.doc
sfp> RENAME myfile.txt myfile.doc <enter>

5.6.1.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- the current remote filename;
- the new remote filename.

In response to the user command RENAME, the User-PI sends the protocol command RNFR to the Server-PI via the control connection.

In turn the Server-PI saves the current name of the file locally if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-PI sends the protocol command RNTO to the Server-PI via the control connection.

In turn the Server-PI renames the file specified in RNFR to the name specified in RNTO if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.6.2 DELETE A FILE (DELETE)

With the DELETE file operation command, the user can delete a file from a remote system.

If the server supports login access control, the user must be logged in to use this command.

5.6.2.1 USER INTERFACE SPECIFICATION

DELETE <filename>

where <filename> is the name of the file to delete

5.6.2.2 EXAMPLES

Example 1: Delete file myfile.txt
sfp> delete myfile.txt <enter>

5.6.2.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- name of file to delete.

In response to the user command DELETE, the User-PI sends the protocol command DELE to the Server-PI via the control connection.

In turn the Server-PI deletes the specified file if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.6.3 LIST DIRECTORY CONTENTS (LS)

With the LS file operation command, the user can list the contents of a directory on the remote system, if the remote system supports directory structures.

If the server supports login access control, the user must be logged in to use this command.

5.6.3.1 USER INTERFACE SPECIFICATION

LS <directory path>

where <directory path> is either a directory path or a filename.

5.6.3.2 EXAMPLES

Example 1: List all files in the current working directory.
sfp> LS <enter>

Example 2: List all files in the current working directory matching f*.c
sfp> LS f*.c <enter>

Example 3: List all files in the directory /home/jones
sfp> LS /home/jones <enter>

5.6.3.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- directory to list (use current directory if none specified by user).

The User-PI forwards the following information to the User-DTP:

write output to screen.

If the 'send port' state flag is enabled, the User-PI computes a data PORT for the user and sends the protocol command PORT to the Server-PI via the control connection. The Server-PI saves the PORT information and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection. The User-PI forwards the server reply to the User-Interface process which forwards it to the user.

The User-DTP listens on data port (the default data port, if SENDPORT was not issued and PORT was not specifically issued by the user). The User-PI sends the protocol command LIST to the Server-PI via the control connection.

The Server-DTP opens a data connection to user's data port and responds to the User-PI via the control connection with a preliminary reply as specified 3.3 of the SCPS-FP protocol specification.

Data connection establishment and management for the User-DTP and the Server-DTP is performed throughout the section as specified in 3.2 of the SCPS-FP protocol specification.

The Server-DTP performs a directory listing on the specified directory and sends the list of files to User-DTP via the data connection.

Upon receipt of the server's preliminary reply, the User-PI forwards the server reply to the User-Interface process which forwards it to the user. The User-DTP then begins accepting the data sent by the Server-DTP (concurrent to the server sending data) and writes it to the screen.

Upon completion of the transfer or on the occurrence of an error, the Server-PI closes the data connection and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's final reply to the User-Interface process which forwards it to the user. The User-Interface process waits for next user command.

5.6.4 CREATE DIRECTORY (MKDIR)

With the MKDIR file operation command, the user can create a directory on the remote system, if the remote system supports directory structures.

If the server supports login access control, the user must be logged in to use this command.

5.6.4.1 USER INTERFACE SPECIFICATION

MKDIR <directory path>

where <directory path> is a directory path.

5.6.4.2 EXAMPLES

Example 1: Create directory XYZ in current working directory
sfp> MKDIR XYZ <enter>

5.6.4.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- name of directory to create.

In response to the user command MKDIR, the User-PI sends the protocol command MKD to the Server-PI via the control connection.

In turn the Server-PI creates the specified directory if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.6.5 DELETE DIRECTORY (RMDIR)

With the RMDIR file operation command, the user can delete a directory on the remote system, if the remote system supports directory structures.

If the server supports login access control, the user must be logged in to use this command.

5.6.5.1 USER INTERFACE SPECIFICATION

RMDIR <directory path>

where <directory path> is a directory path.

5.6.5.2 EXAMPLES

Example 1: Delete directory XYZ from current working directory
sfp> RMDIR XYZ <enter>

5.6.5.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- name of directory to delete.

In response to the user command RMDIR, the User-PI sends the protocol command RMD to the Server-PI via the control connection.

In turn the Server-PI deletes the specified directory if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.6.6 CHANGE WORKING DIRECTORY (CD)

With the CD file operation command, the user can change the current working directory on the remote system to a different directory, if the remote system supports directory structures.

If the server supports login access control, the user must be logged in to use this command.

5.6.6.1 USER INTERFACE SPECIFICATION

CD <directory path>

where <directory path> is a directory path.

5.6.6.2 EXAMPLES

Example 1: Change current working directory to XYZ
sfp> CD XYZ <enter>

5.6.6.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- name of directory to which to change.

In response to the user command CD, the User-PI sends the protocol command CWD to the Server-PI via the control connection.

In turn the Server-PI changes the current working directory to the specified directory if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.6.7 GET FILE SIZE (SIZE)

With the SIZE file operation command, the user request the size of a remote file.

If the server supports login access control, the user must be logged in to use this command.

5.6.7.1 USER INTERFACE SPECIFICATION

SIZE <pathname>

where <pathname> is a string indicating the filename or full pathname with filename of the file for which to obtain the size.

5.6.7.2 EXAMPLES

Example 1: Get size of ASCII file myfile.xyz:
sfp> size myfile.xyz <enter>
"213 myfile.xyz SIZE 30 bytes, 10 lines"

Example 2: Get size of binary file yourfile.xyz:
sfp> size yourfile.xyz <enter>
"213 yourfile.xyz SIZE 500 bytes"

5.6.7.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- name of file to size.

In response to the user command SIZE, the User-PI sends the protocol command SIZE to the Server-PI via the control connection.

In turn the Server-PI obtains the size of the file as stored locally if the request is error free and responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.7 MISCELLANEOUS SERVICES

This subsection describes the service interfaces for the following miscellaneous services:

execute commands specific to remote system.

5.7.1 EXECUTE SITE SPECIFIC COMMANDS (SITE)

With the SITE file operation command, the user can execute a command that is site specific (i.e., specific to the remote system).

If the server supports login access control, the user must be logged in to use this command.

5.7.1.1 USER INTERFACE SPECIFICATION

SITE <string>

where <string> is command to execute

5.7.1.2 EXAMPLES

Example 1: Execute the 'chmod' UNIX command.
sfp> SITE "chmod +x file1" <enter>

5.7.1.3 INTERACTION WITH PROTOCOL

The User-Interface process parses the user command and its arguments. If syntax errors are encountered, the user is notified of the errors and the user command is terminated. The User-Interface process waits for the next user command.

If the user command parses correctly, the User-Interface process provides the following information to the User-PI:

- user command;
- the string (command) to execute at the server.

In response to the user command SITE, the User-PI sends the protocol command SITE to the Server-PI via the control connection.

In turn the Server-PI sends the string to the remote system's operating system for execution. The Server-PI responds to the User-PI with an appropriate reply (as specified in 3.3 of the SCPS-FP protocol specification) via the control connection.

The User-PI forwards the server's reply to the User-Interface process which forwards it to the user. The User-Interface process waits for the next user command.

5.8 TYPICAL SCPS-FP SCENARIOS

This subsection provides typical scenarios for a standard file transfer (5.8.1, as taken from RFC 959, section 7), stream mode autorestart for a User-FP to Server-FP file transfer (5.8.2), stream mode autorestart for a Server-FP to User-FP file transfer (5.8.3), a record update (5.8.4), a record read (5.8.5), a file transfer manual interrupt (5.8.6), and a file transfer manual abort (5.8.7). All of these scenarios ignore the PORT command which may be used prior to the file transfer.

The following Legend applies to the figures in 5.8.2 through 5.8.6:

italicized text	- option-related
[ ]	- conditional response
" "	- user entered

5.8.1 SCENARIO: USER AT HOST U WANTING TO TRANSFER FILES TO/FROM HOSTS

In general, the user will communicate to the server via a mediating User-FP process. The following may be a typical scenario. The User-FP prompts are shown in parentheses, '---->' represents commands from host U to host S, and '<----' represents replies from host S to host U.

```
LOCAL COMMANDS BY USER              ACTION INVOLVED

sfp (host) multics<CR>        Connect to host S, port L,
                                     establishing control connections.
                                     <---- 220 Service ready <CRLF>.
      username Doe <CR>              USER Doe<CRLF>---->
                                     <---- 331 User name ok,
                                               need password<CRLF>.
      password mumble <CR>           PASS mumble<CRLF>---->
                                     <---- 230 User logged in<CRLF>.
```

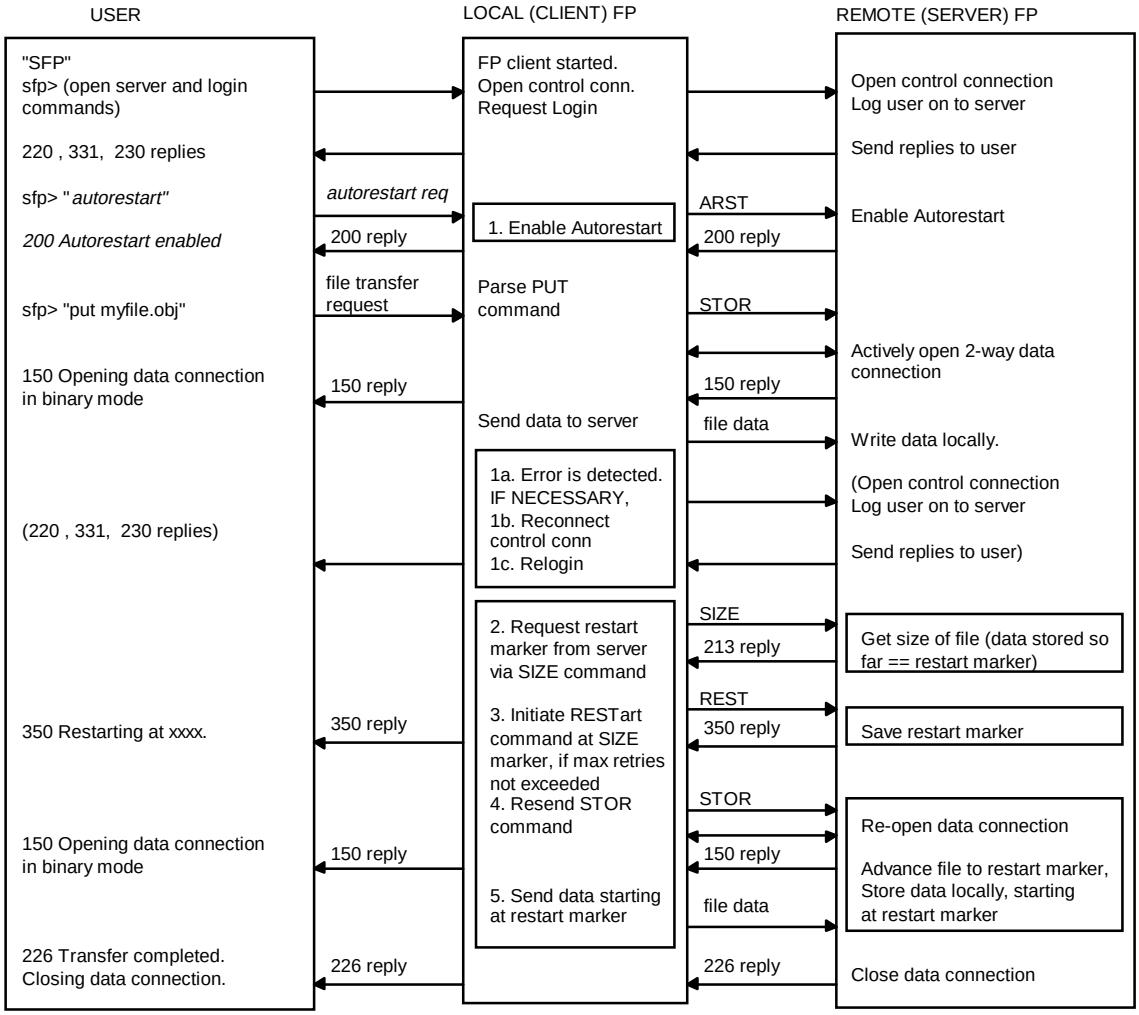
```
retrieve (local type) ASCII<CR>
(local pathname) test 1 <CR>   User-FP opens local file in ASCII.
(for. pathname) test.pl1<CR>   RETR test.pl1<CRLF> ---->
                               <---- 150 File status okay;
                                     about to open data
                                     connection<CRLF>.
                               Server makes data connection
                               to port U.

                               <---- 226 Closing data connection,
                                   file transfer successful<CRLF>.
type Image<CR>                 TYPE I<CRLF> ---->
                               <---- 200 Command OK<CRLF>

store (local type) image<CR>
(local pathname) file dump<CR> User-FP opens local file in Image.
(for.pathname) >udd>cn>fd<CR>  STOR >udd>cn>fd<CRLF> ---->
                               <---- 550 Access denied<CRLF>
terminate                      QUIT <CRLF> ---->
                               Server closes all
                               connections.
```

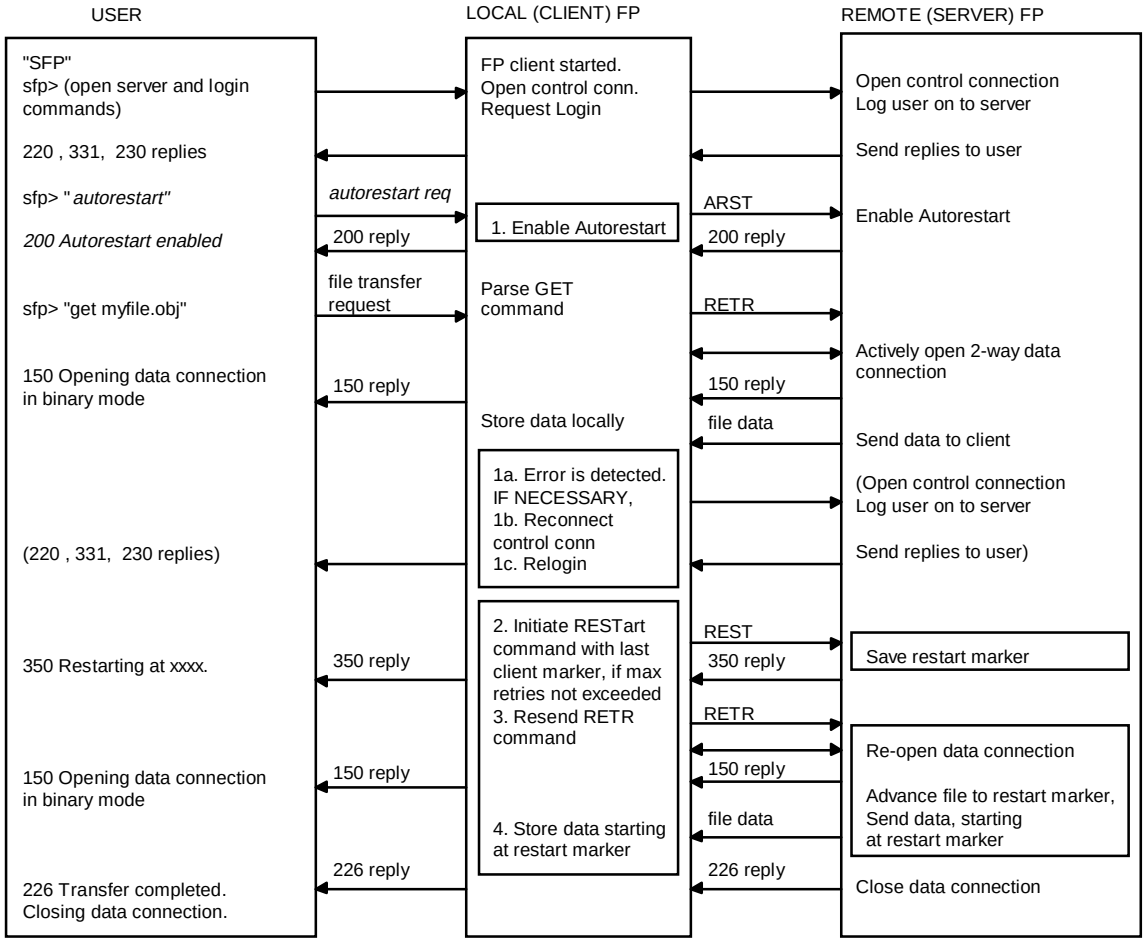
5.8.2 SCENARIO: AUTORESTART OF USER TO SERVER FILE TRANSFER (IN STREAM MODE)

This scenario illustrates the series of user and SCPS-FP commands that are transferred between the user, the User-FP (local/client SCPS-FP), and the Server-FP (remote/server SCPS-FP) when autorestarting a user-to-server transfer. This example assumes that autorestart is disabled prior to the start of the scenario.



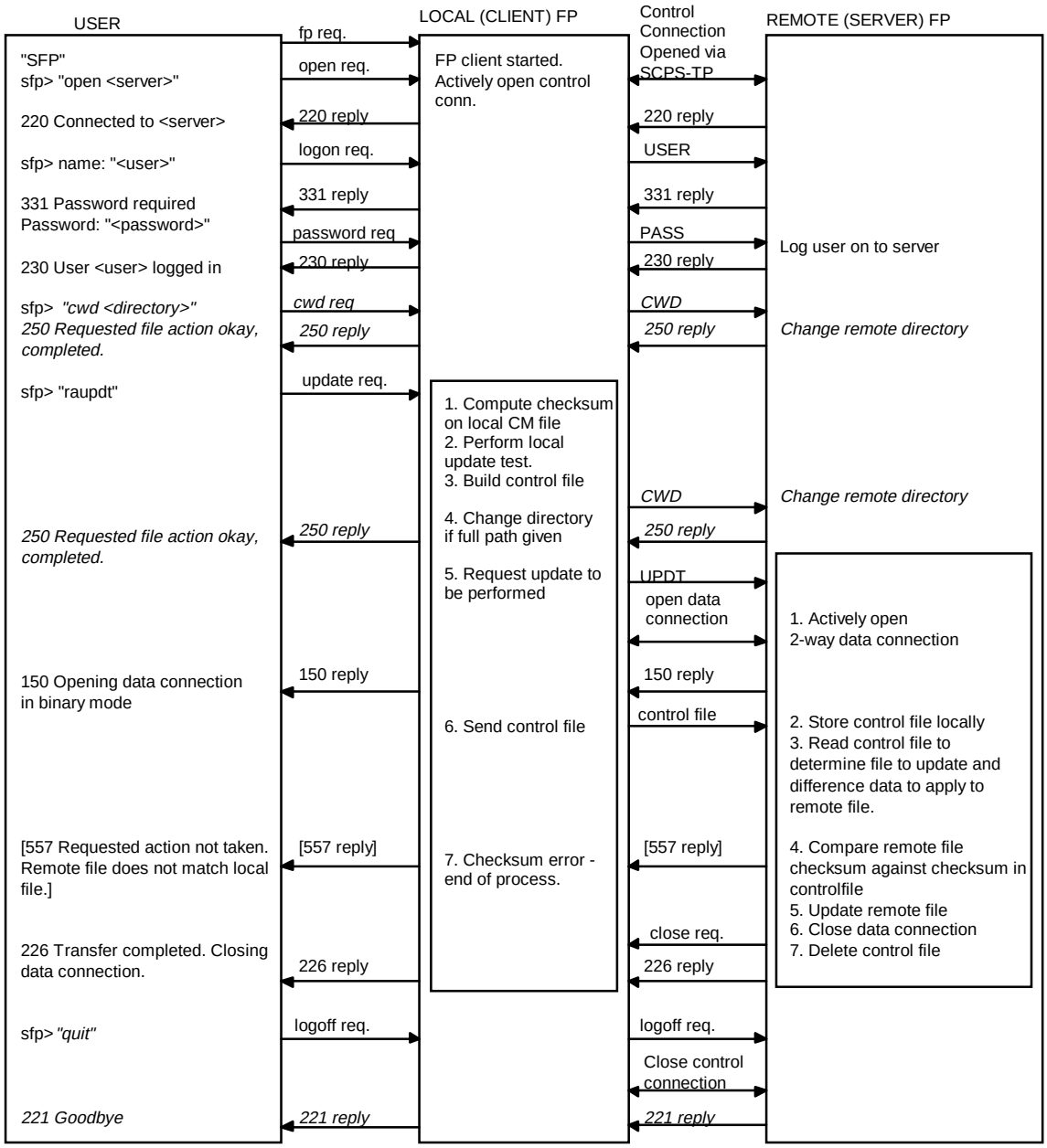
5.8.3 SCENARIO: AUTORESTART OF SERVER TO USER FILE TRANSFER (IN STREAM MODE)

This scenario illustrates the series of user and SCPS-FP commands that are transferred between the user, the User-FP (local/client SCPS-FP), and the Server-FP (remote/server SCPS-FP) when autorestarting a server-to-user transfer. This example assumes that autorestart is disabled prior to the start of the scenario.



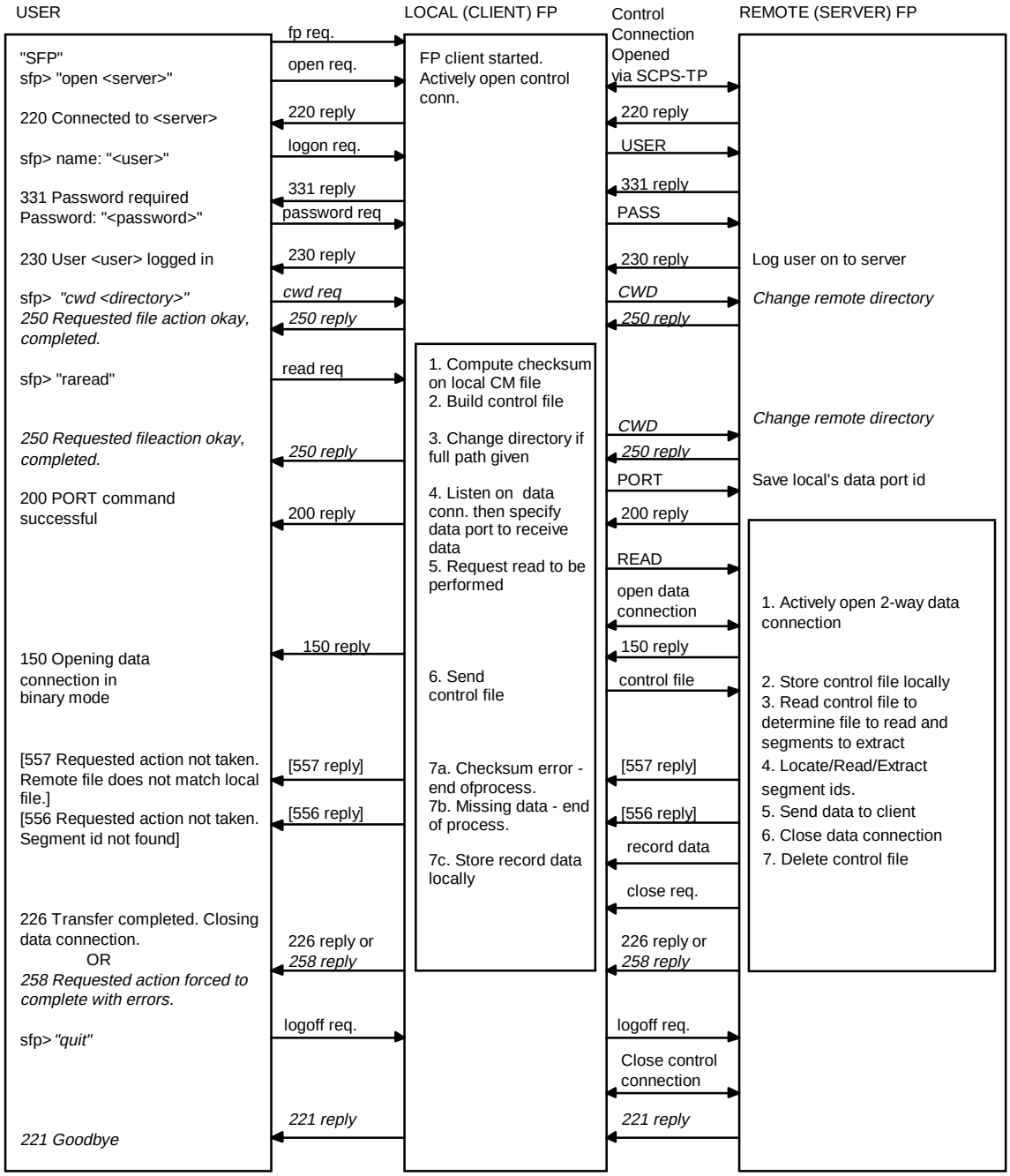
5.8.4 SCENARIO: RECORD UPDATE

This scenario illustrates the series of user and SCPS-FP commands that are transferred between the user, the User-FP (local/client SCPS-FP), and the Server-FP (remote/server SCPS-FP) for a record update.



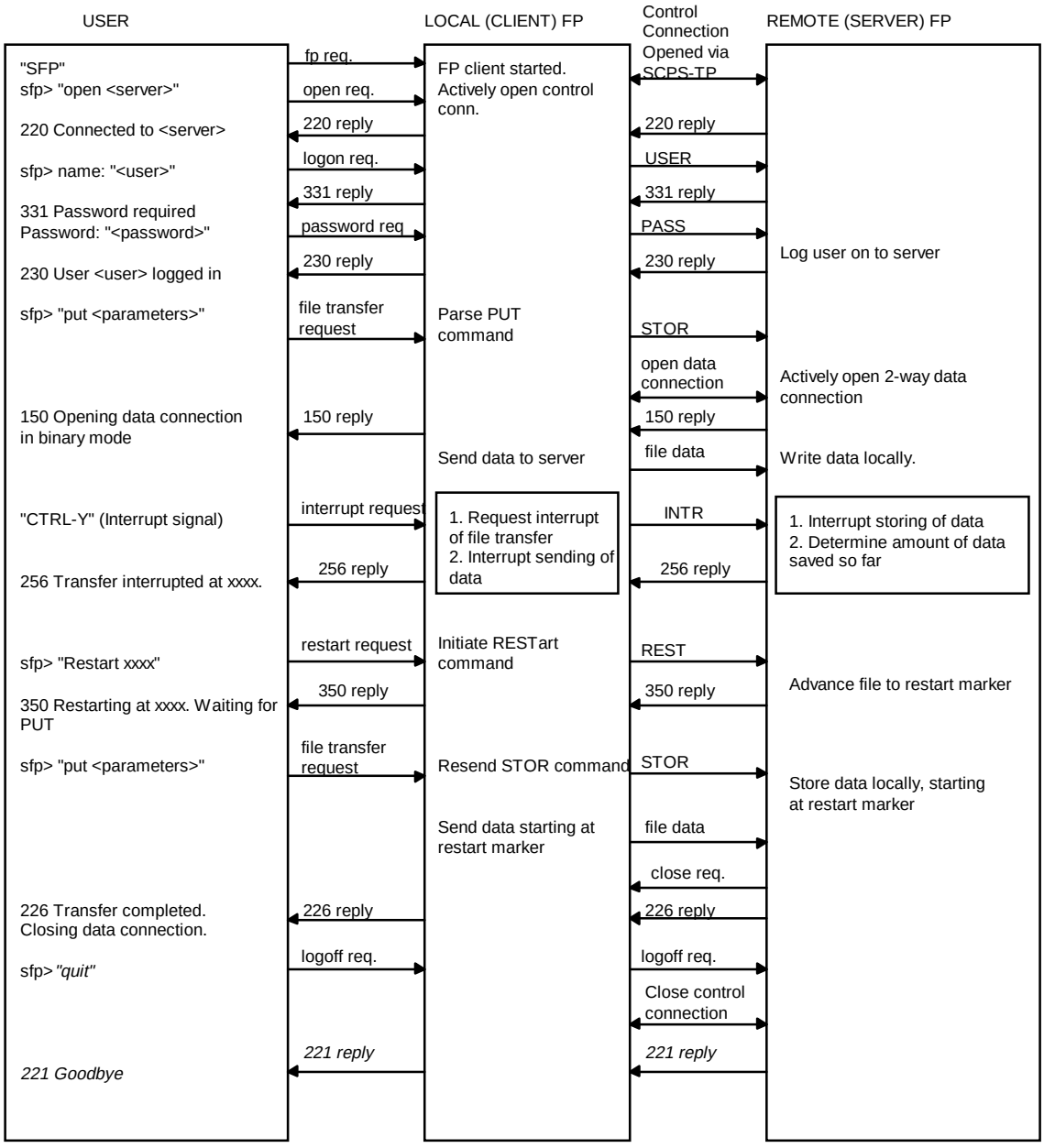
5.8.5 SCENARIO: RECORD READ

This scenario illustrates the series of user and SCPS-FP commands that are transferred between the user, the User-FP (local/client SCPS-FP), and the Server-FP (remote/server SCPS-FP) for a record read.



5.8.6 SCENARIO: MANUAL INTERRUPT OF FILE TRANSFER

This scenario illustrates the series of user and SCPS-FP commands that are transferred between the user, the User-FP (local/client SCPS-FP), and the Server-FP (remote/server SCPS-FP) for a manual interrupt of a user-to-server file transfer.



5.8.7 SCENARIO: MANUAL ABORT OF FILE TRANSFER

This scenario illustrates the series of user and SCPS-FP commands that are transferred between the user, the User-FP (local/client SCPS-FP), and the Server-FP (remote/server SCPS-FP) for a manual abort of a user-to-server file transfer.