WN 97W0000095

WORKING NOTE

User's Manual for the SCPS Reference Implementation Software

September 1997

Robert C. Durst
Patrick D. Feighery
Mary Jo Zukoski

Sponsor: NASA
Dept. No.: W15F
Contract No.: F-19628-94-C-0001
Project No.: 03970639

Washington C^3 Center
McLean, Virginia

Abstract

This document provides the information necessary for individuals receiving the Space Communications Protocol Standards (SCPS) Reference Implementation Software Release Distribution version 1.0 to install and use the software.

KEYWORDS: SCPS, SCPS-FP, SCPS-TP, SCPS-SP, SCPS-NP, Reference Implementation Supporting Documentation

Table of Contents

Section 1

Introduction

This section provides a basic background on the SCPS project and the SCPS software release.

1.1 What is SCPS?

The Space Communications Protocol Standards (SCPS) project has been jointly established by NASA and DOD to recommend and develop standard space data communications protocols. It is the task of the SCPS project to specify integrated layered stacks of standards (and their options) that can satisfy common requirements of both the military and civil space user communities of the United States. Every effort is being made to converge on common solutions to common problems so that a wide range of opportunities for future standardization and interoperability both within and between these communities may rapidly emerge. Reduced life cycle costs as well as increased reliability, maintainability, portability, supportability and securability are expected to result from these common standards.

The primary scope of the SCPS project is to recommend and develop data communications protocol standards for the transfer of information between space mission end systems. The ends of a data flow may be a computer located on the ground which is communicating with a remote computer located in space, or two communicating computers that are located in space and separated by one or more in-space communications links, or combinations of both. The primary focus of the SCPS is therefore on the transfer of data through space-to-ground, ground-to-space, and space-to-space communications subnetworks.

While satisfying space mission requirements constitutes the primary focus, the project has kept in mind the needs of similar communication environments:

- *Stressed communications environments*, such as in battlefield situations characterized by long propagation delays, noisy and bandwidth-constrained communications links, intermittent connectivity and resource-constrained end systems.
- *Commercial wireless environments*, such those provided by satellite or radio systems supporting Internet or Intranet traffic.

The SCPS project is a joint activity between the National Aeronautics and Space Administration (NASA) and the Department of Defense (DOD) that is chartered with producing a set of specifications for standardized end-to-end space mission data transfer covering the following technical areas:

1. An efficient file handling protocol (the SCPS File Protocol, or SCPS-FP), optimized towards the up-loading of spacecraft commands and software, and the downloading of collections of observational telemetry data;
2. Various flavors of underlying retransmission control protocol (the SCPS Transport Protocol, or SCPS-TP), optimized to provide reliable end-to-end delivery of spacecraft command and telemetry messages between computers that are communicating over a network containing one or more unreliable space data transmission paths;
3. A data protection mechanism (the SCPS Security Protocol, or SCPS-SP) which assures the end-to-end security and integrity of such message exchange;
4. A scaleable networking protocol (the SCPS Network Protocol, or SCPS-NP) that supports both connectionless and connection oriented routing of these messages through networks containing multiple space data links.

1.2 The SCPS Reference Implementation

As part of the standards development process, implementations of each protocol have been developed. These protocol implementations have been integrated together for distribution as the Reference Implementation of the SCPS protocols. This Reference Implementation is being made available for the purpose of evaluation and experimentation with the protocols by potential users of the SCPS capabilities. The SCPS Reference Implementation software also contains SCPS utilities, sample client/server applications and a benchmarking tool.

The protocols implemented in this distribution of the SCPS Reference Implementation are based on the versions of the standards released for review within the international space community, under the auspices of the Consultative Committee for Space Data Systems, and within the United States Department of Defense space community, under the auspices of the Space and Missile Systems Command (SMC) and the Defense Information Services Agency (DISA). The specific versions of the standards are listed in the reference's section of this document. It is anticipated that subsequent releases of this Reference Implementation will track changes to the protocols resulting from national and international review.

SAIC, Inc., in McLean, VA developed the SCPS File Protocol. SPARTA, Inc., in Columbia, MD developed the SCPS Security Protocol. The MITRE Corporation in Reston, VA developed the SCPS Transport and Network Protocols. The MITRE Corporation also integrated the four SCPS protocols with the assistance of the other developers.

The SCPS Reference Implementation has been mainly developed on a Sun SPARCstation running the Sun OS 4.1.3 operating system. However, it has been ported to many other environments, as described later in this section. For Unix-based environments to which the Reference Implementation has not already been ported, some effort may be required. However, the Reference Implementation has been developed in a *portable* manner, without depending on capabilities that are not typically found on Unix-based platforms.

1.3 Philosophy of the SCPS Reference Implementation

The following section describes the philosophy behind the development of the SCPS Reference Implementation.

1.3.1 Portability

The SCPS Reference Implementation was developed in the C programming language, and executes in application space on UNIX-based machines. It uses system calls common to most UNIX-based machines, and is therefore portable to many types of execution environments

The transport layer and below of the SCPS Reference Implementation has been ported to the following environments:

- Sun Microsystems Sun 3/60 hardware platform running the Sun OS 4.1.1 operating system
- Sun Microsystems SPARCstations hardware platform running either the SunOS 4.1.3 or SunOS 4.1.4 or Solaris 2.5 or Solaris 2.6 operating system
- Intel 80x86 hardware platform running the FreeBSD operating system.
- Intel 80x86 hardware platform running the NetBSD operating system.
- Intel 80x86 hardware platform running the Linux operating system.
- Silicon Graphics Indigo hardware platform running the IRIX operating system.

The SCPS File Protocol (SCPS-FP) has been ported to all of the above environments except Solaris, NetBSD, and the Silicon Graphics IRIX operating environments.

The SCPS Reference Implementation uses its own thread scheduler to achieve a limited multitasking capability within a single Unix process. This particular approach was selected because POSIX threads were not (and still are not) as widely supported as necessary, and because some operating platforms (e.g. spacecraft) had no real operating system services at all. The impact of this decision on portability is that there is some embedded assembly language code in the file “thread.c”. This assembly language code saves and restores the stack frame and the machine registers during task context switches. The assembly language code is embedded using the GNU C-compiler style for embedding assembly code, and therefore the SCPS Reference Implementation depends on the availability of the GNU C compiler. In addition, users wishing to port the SCPS Reference Implementation beyond the following hardware architectures will need to add a small amount of assembly language to the function “threadHandoffCPU” in the file “thread.c”:

- Sun Sparc instruction set architecture
- Motorola M680X0 instruction set architecture

- Intel 80X86 instruction set architecture
- MIPS 3000 instruction set architecture

(Note that this assembly language has also been generated for the MIL-STD 1750 processor, but is not distributed with the SCPS Reference Implementation. Users interested in this code should contact the authors.)

The SCPS Reference Implementation has been developed to be portable across most Unix-type operating systems. There are some operating system specific portions of the software, and users porting to other operating system platforms than those mentioned above may need to make modifications to the code. Users should also note that the Reference Implementation has not been ported to the Microsoft Disk Operating System (DOS) or Windows environments. Users wishing to undertake such a portation task are encouraged to contact the developers via the feedback mechanisms listed later in this document.

1.3.2 User Interface

The SCPS Reference Implementation has adopted a policy to capitalize on established interfaces for network applications to minimize SCPS software development efforts. The Berkeley socket interface is the most popular network interface to the Internet protocols for network applications development. The SCPS lower layer application interface mirrors the Berkeley socket paradigm. This approach also mitigates technical risks by exploiting operational experience that the Internet protocols have accrued.

The SCPS File Protocol presents the same user interface at the client as the Internet File Transfer Protocol. Again, the purpose of this decision was to maximize user familiarity with the interface and to minimize development costs.

1.3.3 Implementation Organization

The SCPS Reference Implementation is distributed in a compressed tar (tape archive) file that, when uncompressed and extracted from the archive, creates a “SCPS” directory, and several subdirectories. The subdirectories are as follows:

Directory	Contents
FP	SCPS File Protocol source code
apps	Example applications that operate directly over SCPS-TP via the scps socket interface
bin	Directory into which the executable code will be placed after compilation. Also, the initial location for storage of various initialization files for the SCPS Security Protocol and the SCPS Network Protocol
include	The directory that contains header files that are included (directly or indirectly) by application programs using the SCPS protocols. Refer to the manual pages of the SCPS socket interface calls for specifics.
lib	The directory that contains the SCPS library file, which is comprised of the object files that support the configuration built by the user
source	The directory that contains source code for the SCPS-TP, SCPS-SP, and SCPS-NP, plus supporting software. The supporting software includes the interface code to the lower layers, code to perform buffer management, and a simple thread scheduler for multitasking external to the operating system kernel.
docs	The directory that contains adobe pdf versions of this file and the SCPS TP and SCPS NP user's guide.
tcpdump	The directory that contains files to patch tcpdump-3.4a5 allowing it to understand the SCPS lower layer protocols.

1.3.4 Implementation of Protocol Features

The SCPS Reference Implementation has been developed with the philosophy that only the specific SCPS lower layer services required by a particular application should be incorporated the build that supports that application. To realize this, the SCPS Reference Implementation allows users to customize the SCPS library to include only the protocols and options within those protocols which are required.

The SCPS-TP forms the basis of the SCPS library, the SCPS-SP may be included or not, and the network layer protocol may either be SCPS-NP or IP. (When operating over IP, the SCPS-SP uses the Internet protocol number reserved for private encryption protocols.) Refer to section 2.5 (Customizing the SCPS Environment) for information on how to incorporate specific protocols into the C library.

Within the SCPS-TP, users may customize the specific protocol capabilities that are included in the build. The flags to control this customization are also described in section 2.5.

1.3.5 Lower Layer Services

The SCPS Reference Implementation operates outside the Unix kernel, as an application process. It uses the kernel's socket interface as a substitute for direct access to lower layer devices. The SCPS Reference Implementation may operate over many different types of lower layer services, to allow flexibility in the development and execution of SCPS-based applications. Users may configure the SCPS protocol suite to operate directly over a Raw IP interface, or over a UDP/IP interface. To use the Raw IP interface, the SCPS Reference Implementation process must be run at superuser privilege.

Users have a choice of two network layer protocols when using the SCPS Reference Implementation: IP or the SCPS NP. When using the SCPS-NP, the Reference Implementation always encapsulates the SCPS-NP packets in either Raw IP or in UDP/IP. This permits routing of the SCPS-NP packets through the Internet. (In an operational environment, users may choose to retain this encapsulation in ground networks, and strip the encapsulating IP or UDP/IP headers at the point of entry into the wireless network segment.) When using IP as the network layer protocol, the IP packets may either be routed directly, or, like SCPS-NP packets, may be encapsulated in (another) IP header. (This could be useful when the IP address of the point of entry into the wireless network segment is known. The "outer" packet would carry the IP address of the wireless entry point, and the "inner" packet would carry the IP address of the destination.) As with SCPS-NP packets, IP packets may also be encapsulated in UDP/IP headers. For both IP and the SCPS-NP, the UDP/IP approach is necessary if the user wishes to test in loopback mode on a single machine.

The following table summarizes the alternatives for configuring lower layer services:

Network Layer Protocol In Use	Lower Layer Service Options
IP	Direct operation over Ethernet IP-in-IP encapsulation IP in UDP/IP encapsulation IP Loopback to same machine via UDP/IP
SCPS-NP	NP-in-IP encapsulation NP in UDP/IP encapsulation NP Loopback to same machine via UDP/IP

1.3.6 Execution Speed

The SCPS Reference Implementation was developed to be portable to many environments. To make the SCPS protocols sufficiently generic to run under many different environments, execution speed has been sacrificed. We have attempted to minimize the performance impacts, but some are unavoidable. Of particular note is the fact that the protocols execute outside the operating system kernel, and will be adversely affected by writes across the kernel-to-application interface. This effect will become more significant as packet sizes decrease (because the per-packet overhead increases in relation to the amount of user data being transmitted). However, under certain circumstances the SCPS Reference Implementation has been able to operate at a sustained rate of over 27 million bits per second (on a SPARCstation 20 using an ATM interface to a DS3 link).

1.4 Software Included in the SCPS Reference Implementation Software Distribution Release

The SCPS Reference Implementation software distribution release contains C language source files to create the SCPS library and the SCPS file protocol (both client and server). Also provided with the software distribution is an example of a client/server application using the services of the SCPS lower layer protocols, a SCPS benchmarking tool, and diffs to tcpdump that allow it to understand the SCPS lower layer protocols.

In addition to the source and header files, there are some configuration files that are used at run-time to support the SCPS Security Protocol and the SCPS Network Protocol. These files must be adapted to your particular network environment, and are discussed in section 3.1.

1.5 Supporting Documentation

The MITRE Working Note WN 97W0000018 discusses the SCPS Transport Protocol, providing background material and manual pages describing the options that may be invoked by applications. The MITRE Working Note WN 97W0000026 provides similar information for the SCPS Network Layer protocol. Pdf versions of these documents and of this document are located in the *docs* directory of the SCPS distribution.

Section 2

Building SCPS

This section provides information necessary to build a distribution of the SCPS protocol suite. We first discuss the operating environment, noting those to which the SCPS Reference Implementation has been ported and tested. Next, we describe how to extract the SCPS Reference Implementation from its distribution file. Third, we discuss the steps that each user must perform before building the SCPS Reference Implementation for the first time. Finally, we discuss the "make" procedure and the results from that procedure that the user should expect.

2.1 Environment

The SCPS Reference Implementation has been developed to execute on various hardware platforms and operating system environments. The lower layers (transport and below) of the current SCPS Reference Implementation supports the following environments:

- Sun Microsystems Sun 3/60 hardware platform running the Sun OS 4.1.1 operating system
- Sun Microsystems SPARCstations hardware platform running either the SunOS 4.1.3 or SunOS 4.1.4 or Solaris 2.5 operating system
- Intel 80x86 hardware platform running the FreeBSD operating system.
- Intel 80x86 hardware platform running the NetBSD operating system.
- Intel 80x86 hardware platform running the Linux operating system.
- Silicon Graphics hardware platform running the IRIX operating system.

It should be noted that the SCPS File Protocol has been developed for the SunOS, FreeBSD and Linux operating system. It has not been ported to the Solaris, NetBSD, or IRIX operating systems.

The SCPS Reference Implementation must be built using the GNU/gcc compiler. This is a freely available C compiler that has been ported to many environments. The SCPS Reference Implementation software has been developed using GNU/gcc versions 2.5.8, 2.6.3, and 2.7.2. GNU/gcc versions 2.6.1, 2.6.2, 2.7.0, and 2.7.1 have known problems and should not be used when building the SCPS Reference Implementation.

2.2 Extracting the SCPS Reference Implementation Software Distribution

The SCPS Reference Implementation has been distributed in a compressed tar format. Before the software can be built for a particular environment it must be uncompressed and unarchived.

To uncompress the software distribution enter the following command:

uncompress SCPS_RI_1.1.1.tar.Z

This will result in the file SCPS_RI.1.1.1.tar archive file to be created. To unarchive this file enter the following command:

tar -xf SCPS_RI_1.1.1.tar

The result of this command will be the creation of a directory, named 'SCPS,' that contains the files and directories described in Appendix D. The SCPS directory will be created as a subdirectory of the current working directory from which the tar command was executed, and will be referred to as the "top-level" directory in subsequent sections.

2.3 Pre-build Configuration Actions

Before one builds the SCPS Reference Implementation for the first time, the user must perform some configuration actions. These actions involve executing configuration scripts and providing parameters to those scripts to convey the user's wishes. Three separate configuration actions must be taken. The following paragraphs describe these configuration steps. In addition to these mandatory actions, the user may wish to reconfigure some of the internal options within the SCPS-TP. This activity will be discussed after the discussion of the configuration steps.

2.3.1 Configuring the SCPS Lower Layers

The configuration process for the SCPS lower layer protocol software involves two actions that are combined into a single configuration command.

The first of the two actions is to determine the specifics about the host computer (in terms of CPU type and operating system). This is performed in an automated manner, with no user input required beyond starting the configuration command script.

The second of the two actions is for the user to inform the command script about how to configure the SCPS lower layer protocol software. There are three issues to be addressed:

1. Should the SCPS stack include the SCPS Security Protocol?
2. Should the network layer protocol be IP or the SCPS Network Protocol?
3. What form of lower layer encapsulation should be used? (Refer to section 1.3.5 for a discussion of the alternatives available.)

The result of each of these decisions is passed to the configuration command script as parameters. The configuration command script will use a default value for any parameter that is not supplied. The name of the configuration command script is 'configure.' To invoke the configuration command script and accept all of the default values for parameters, one would change directory to the 'source' subdirectory under the SCPS directory and execute the command

./configure

The '*./*' preceding the command name is to ensure that the configure command *in the 'source' directory* is executed, rather than any others that may be resident on the system.

To configure the SCPS lower layers to include the SCPS Security Protocol the following option must be included on the configure command line:

--scpssp=yes

The default is to not include the SCPS Security protocol (i.e., --scpssp=no)

To configure the SCPS lower layers to include the SCPS Network Protocol the following option must be included on the configure command line:

--scpsnp=yes

The default is to not include the SCPS Network protocol (i.e., --scpsnp=no), but rather use IP.

The following option is used to configure the encapsulation protocol for the SCPS lower layer protocols which are not using the SCPS Network Protocol

To configure the encapsulation protocol to be IP the following option must be included on the configure command line:

--encap=raw

To configure the encapsulation protocol to be IP in IP the following option must be included on the configure command line:

--encap=raw_encap

To configure the encapsulation protocol to be UDP/IP the following option must be included on the configure command line:

--encap=udp

To configure the encapsulation protocol to be UDP/IP in loopback mode the following option must be included on the configure command line:

--encap=loop

The following option is used to configure the encapsulation protocol for the SCPS lower layer protocols using the SCPS Network Protocol

To configure the encapsulation protocol to be NP in IP the following option must be included on the configure command line:

--encap=raw

To configure the encapsulation protocol to be UDP/IP the following option must be included on the configure command line:

--encap=udp

To configure the encapsulation protocol to be UDP/IP in loopback mode the following option must be included on the configure command line:

--encap=loop

For example, if the SCPS Lower layer were configured to include the SCPS Network Protocol, the SCPS File Protocol, and run over UDP/IP (i.e. TP/SP/NP/UDP/IP) the following is how it would be configured:

./configure --scpssp=yes --scpsnp=yes --encap=udp

The configuration command script uses the information supplied by the user and the information that it determines about the host machine to create a "Makefile" that controls the software build operation. Upon execution, the configuration command script will echo the configuration decisions supplied by the user (or default values) and information regarding the host configuration. The following results from executing the above example on a Sun Sparc workstation using the SunOS 4.1.3 operating system:

./configure --scpssp=yes --scpsnp=yes --encap=udp

SCPS-SP set to yes

SCPS-NP set to yes

Encapsulation set to udp

Configuring make file for sparc-sun-sunos4.1.3_U1

This completes the pre-build configuration of the SCPS Lower Layers, with the possible exception of user-modifications to the default SCPS-TP configuration. These possible modifications are described in section 2.3.4.

2.3.2 Configuring the SCPS demonstration applications

There are no user-configurable options for the SCPS demonstration applications. They need only to determine the type of CPU and operating system. From the SCPS directory, change directory to the 'apps' subdirectory, and execute the configuration command script in the following manner:

./configure

If executed on the same system as the example above, one should see the following output:

Configuring make file for sparc-sun-sunos4.1.3_U1

This completes the configuration of the SCPS demonstration applications.

2.3.3 Configuring the SCPS File Protocol

The configuration process for the SCPS File Protocol is similar to that for the SCPS Lower Layers, in that the software must determine the type of the local CPU and operating system, and it must receive information from the user regarding configuration options. Three major configuration decisions must be made:

1. Should the SCPS-FP implementation size be "small" (roughly corresponding to the "SCPS-FP Conforming Minimum Implementation" from the SCPS-FP specification) or "medium" (roughly corresponding to the "SCPS Conforming Full Implementation") or "large" (roughly corresponding to the "FTP Interoperable Implementation")? Refer to the SCPS-FP specification (CCSDS 717.0, MIL STD 2045-47000) for more details on these capabilities.
2. Should the SCPS-FP operate over SCPS-TP or over the system's in-kernel TCP implementation?
3. Should debug information be printed to the screen during operation?

As with the SCPS Lower Layer configuration, the result of each of these decisions is passed to the configuration command script as parameters. The configuration command script will use a default value for any parameter that is not supplied. To invoke the configuration command script and accept all of the default values for parameters, one would change directory to the 'FP' subdirectory under the SCPS directory and execute the command

./configure

Continuing our example of using a Sun Sparc workstation, the response to the 'configure' command (with default values for all parameters) is as follows:

./configure

Configuring make file for sparc-sun-sunos4.1.3_U1

Created "Makefile"

system=sunos size=small debug=off Using TP sockets

In order to specify a non-default value for the size of the SCPS-FP implementation, the user should specify the following command-line parameter to the 'configure' utility, where *specified_size* is replace by either *small*, *medium*, or *large*:

--size=specified_size

To explicitly specify the transport protocol over which the SCPS-FP should operate, the user should supply the *–tp* parameter to the 'configure' utility, specifying 'yes' to use the SCPS stack as the lower-layer service, and specifying 'no' to use the in-kernel TCP/IP stack as the lower-layer service.

To specify whether debugging information is enabled or disabled, the user should supply the *–debug* parameter, with 'yes' or 'no', respectively.

Here is an example of the 'configure' command with non-default values for all parameters:

./configure --size=medium --tp=no --debug=yes

Configuring make file for sparc-sun-sunos4.1.3_U1

Created "Makefile"

system=sunos size=medium debug=on Using Operating System sockets

Successful execution of the 'configure' utility completes the configuration of the SCPS-FP software.

2.3.4 Configuring SCPS-TP Compile-time Options

The SCPS-TP is typically configured to allow using applications to request optional capabilities at run-time. By default, these optional capabilities are included in the software so that they may be invoked as necessary. Some users may wish to remove capabilities that are not currently required. In the Makefile and the template for the Makefile (Makefile.in), there is a variable called "SCPS_TP," which is specified as a sequence of define constants. Each of these define constants causes certain SCPS-TP capabilities to be included in the compilation. These constants are *defined* by being preceded by "-D." The possible define constants (with their "-D" directives) are described below:

Define Constant	Meaning
-DCONGEST	Include congestion control software (TCP Vegas style)
-DVJ_CONGEST	(Note: -DCONGEST MUST be specified also.) Enable standard TCP congestion control in addition to TCP Vegas style congestion control.
-DOPT_SCALE	Enable the window scaling capabilities defined in RFC 1323
-DOPT_TSTMP	Enable the TCP timestamps option defined in RFC 1323
-DINIT_CWND_INCR	Enable initial congestion window of up to 4380 bytes (refer to J. C. Hoe, ACM SIGCOMM '96, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP")
-DOPT_SCPS	Enable combined negotiation of SNACK, header compression, record boundaries, and BETS via a single connection-time option.
-DOPT_SNACK1	Enable SCPS selective negative acknowledgment option
-DOPT_COMPRESS	Enable SCPS header compression
-DOPT_RECORD_BOUNDARY	Enable record boundaries on SOCK_SEQPACKET sockets
-DOPT_BETS	Enable Best Effort Transmission Service (a partial-reliability TCP service)
-DMFX_TRANS	Enable multiple forward transmissions (mutually exclusive with *all* congestion control schemes).

By default, the SCPS_TP constant is defined in the Makefile as the following string:

```
SCPS_TP =  -DOPT_SCALE -DOPT_SNACK1 -DOPT_BETS -DOPT_TSTMP -DOPT_COMPRESS \
-DCONGEST -DVJ_CONGEST -DOPT_SCPS -DINIT_CWND_INCR -DOPT_RECORD_BOUNDARY
```

Note that the back slash ("\") at the end of the first line indicates a continuation of that line, and should not be deleted.

Users should bear in mind that specifying these options only makes them *available* for use. If a particular capability described above is included in the build of SCPS-TP, the *use* of that capability may be enabled or disabled on a per-connection basis. The following table describes

the default behavior when that capability is included in the build of SCPS-TP. Users may issue socket options to change the behavior from its default. Refer to the SCPS-TP User's Guide for more details.

Capability	Default behavior
TCP Vegas style congestion control	Enabled
TCP standard congestion control	Disabled
RFC 1323 window scaling	Enabled
RFC 1323 TCP timestamps	Enabled
Increased initial congestion window	Enabled
Combined SCPS negotiation option	Enabled
Selective Negative Acknowledgment	Enabled
SCPS Header Compression	Disabled
Record Boundaries	Disabled
Best Effort Transport Service	Disabled
Multiple Forward Transmissions	Disabled

Note that the Makefile is overwritten by an execution of the 'configure' utility. If changes to the SCPS_TP variable in the Makefile are made, but not also made in the Makefile template (Makefile.in), these changes will be lost when the 'configure' utility is next used in the source directory.

2.4 Building the Reference Implementation

The UNIX command 'make' is a commonly-used method of simplifying the maintenance of programs. After the individual directories have been properly configured as described in section 2.3, return to the SCPS top level directory and type:

make

The SCPS Reference Implementation will build the SCPS library, the SCPS-FP, and all of the example SCPS applications. The SCPS-FP executable files and the example applications will be copied to the *bin* directory. The SCPS library will be copied to the *lib* directory.

2.5 Tcpdump

The SCPS Reference Implementation provides a simple method of extending tcpdump to understand the SCPS lower layer protocols. A copy of tcpdump version 3.4a5 (available via anonymous ftp from ftp.ee.lbl.gov) must be obtained. To successfully build tcpdump-3.4a5, the library libpcap-0.4a5 must be acquired and is also available from the above ftp size. To extend tcpdump to understand the SCPS lower layer protocols copy these two files from the SCPS *tcpdump* directory into the tcpdump-3.4a5 directory and execute the *apply_scps_tcpdump_patch* script. After the patch has been applied, reexecute the tcpdump 'configure' utility and make tcpdump according to its directions.

Section 3

Using SCPS

This section provides information necessary to configure the SCPS protocol suite and develop network application programs using SCPS protocol services. This section also provides information on the sample SCPS application examples, tools and utilities distributed with the SCPS Reference Implementation software.

3.1 Configuring SCPS

After the SCPS Reference Implementation Software Release Distribution has been build for the particular environment, certain tables must be configured before SCPS application may be executed. These tables are the SCPS routing table, the SCPS address mapping table and the SCPS security requirements.

3.1.1 Configuring the SCPS Routing Table

When using the SCPS Network Protocol, the SCPS routing table must be configured to create routes between the SCPS entities. The SCPS routing table is staticly configured by reading in the file *npNextHopFile*. Routing entries within SCPS are interpreted in the following manner, "To reach a End System whose IP address is X forward the packet to the next hop system whose IP address is Y." The format of the file is the following. The first line contains the IP address of the machine. The following lines contain a pair of IP addresses where the first address is the destination IP address and the second address is the next hop IP address. It should be noted that all addresses in this file must be in their hexadecimal form.

3.1.2 Configuring the SCPS Address Mapping Table

When using the SCPS Network Protocol, the SCPS address mapping table must be configured to map IP address to a corresponding SCPS NP address. This is required in the application are using any SCPS NP address families. Reading in the file npIP_NP_File staticly configures the SCPS routing table. The format of the file is the following: A pair of addresses separated by a space where the first address is the Extended form of the address and the second address is its corresponding basic form. It should be noted that all addresses in this file must be in their hexadecimal form.

3.1.3 Configuring the SCPS Security Requirements

When using the SCPS Security Protocol, the protocol must be configured before any of the security services may be used. Reading in the file SA_table statically configures the SCPS

security protocol. The format of the file is the following. Each line contains 10 fields separated by a space. Figure 1 is an example for a *SA_table* file.

- Source IP address - In hexadecimal format
- Destination IP address - In hexadecimal format
- crypt key - ASCII key
- crypt block size - Length of the crypt key in octets
- crypt algorithm - 0 for DES, 1 for XOR
- Integrity Check Value key - ASCII key for Integrity check algorithm
- Integrity Check Value size - Length of the Integrity Check Value key in octets
- Integrity Check Value algorithm - 0 for MD5
- Security Label Length - Length of the security
- Security Label - Security label
- Security QoS - Security capabilities allowed for this connection. Simply and the following values together (0x1 for Confidentiality, 0x2 for Authentication, 0x4 for Security Label, and 0x8 for Integrity)

```
c0307280  c0307283  KEYKEYE  8  1  MD5PKRY!  16  0  0  0  F
c0307283  c0307280  KEYKEYE  8  1  MD5PKRY!  16  0  0  0  F
c0307280  c0307299  KEYKEYE  8  1  MD5PKRY!  16  0  0  0  F
c0307299  c0307280  KEYKEYE  8  1  MD5PKRY!  16  0  0  0  F
```

Figure 1. Example of Security Configuration File

3.2 Using SCPS Protocol Services

The SCPS Reference Implementation provides a library based method allows a single application to communicate over a SCPS based communication stack. The SCPS library must be preconfigured with all necessary information.

3.3 Developing Applications using the SCPS Library

To develop applications using the SCPS library, the SCPS header file “scps.h” must be included in the application and it must be linked in with the SCPS library “libscps.a” created for library based applications.

In this version of the Reference Implementation, users must incorporate the thread scheduler into their applications, and explicitly pass control to the scheduler and on to the SCPS stack (to watch for incoming packets) on a frequent basis during execution. The exact frequency depends upon packet arrival rates. Users explicitly pass control by inserting calls to the function *sched()* in their code. The flow of control in the user's application is unaffected by the presence of these calls.

In addition to placing calls to *sched()* in the body of the user's application, the user must perform the start-up actions of initializing the thread scheduler and the SCPS stack in the main routine, and then pass control to the scheduler. An example of this appears in the *scps_initiator.c* file in the *apps* directory. Examine the *main()* function, from the comment

```
/* Boilerplate code */
```

to the call to *exit(0)*.

3.4 SCPS Application Examples

The SCPS Reference Implementation software distribution release provides two examples to aid users in understanding how to develop application using the SCSP protocol suite.

3.4.1 SCPS Server

The program scps_resp is an example of implementing a network server application using the services of the SCPS protocols. This program along with its peer program scps_init show the interaction of application using the services of the SCPS protocol suite. This program allows users to set various SCPS TP options. Since scps_resp is the server application is must be executed before the client application.

The following is an example of using scps_resp:

To receive a stream of data from remote hosts at a rate of 1,000,000 bps using SCPS compression, type the following command on the host:

scps_resp -R 1000000 -C

This will initialize the server and wait for clients to connect.

Developers of application servers using the services of the SCPS protocol suite are encouraged to examine the source code as a *possible* method of implementing SCPS based servers.

Appendix A provides a UNIX manual page description of the scps_resp application.

3.4.2 SCPS Client

The program scps_init is an example of implementing a network client application using the services of the SCPS protocols. This program along with its peer program scps_resp show the interaction of application using the services of the SCPS protocol suite. This program allows users to set various SCPS TP options. Since scps_init is the client application is must be executed after the server application.

The following is an example of using scps_init:

To send a stream of data from to a remove server 'hostb' at a rate of 1,000,000 bps using SCPS compression, type the following command on the local host:

scps_init -R 1000000 -C hostb

This will initialize the client, connect to the waiting server, send a stream of data, and close the connection.

Developers of application clients using the services of the SCPS protocol suite are encouraged to examine the source code as a *possible* method of implementing SCPS based clients.

Appendix B provides a UNIX manual page description of the scps_init application.

3.5 SCPS Utilities

The SCPS Reference Implementation software distribution release provides five utilities to aid users in using and configuring SCPS.

3.5.1 SCPS ttcp

Scps_ttcp is a benchmarking tool for determining SCPS performance between two systems. Scps_ttcp is based on the commonly used performance test program ttcp. Ttcp was originally developed at the US Army Ballistics Research Lab (BRL) and has been released into the public domain. It was then modified by the MITRE Corporation to execute over a SCPS protocol stack.

The following are some examples of using scps_ttcp:

To send a stream of data from one host to another host at a rate of 1,000,000 bps using SCPS compression, type the following command on the remote (receiving) host;

scps_ttcp -r -s -R 1000000

And type the following command on the local (sending) host:

scps_ttcp -t -s -R 1000000 remote_host

After executing scps_ttcp will display performance information about the connection upon its completion. (Note that, as with regular *ttcp*, the sending host will display its performance when the last application data is *enqueued for transmission* rather than when it receives an acknowledgment from the remote host. In long-delay and/or high bit-error-rate environments, this will result in an artificially-low measure of connection delay. However, the performance numbers reported by the receiving side will not display this bias.)

Section 4

Feedback

The SCPS team encourages feedback from users concerning their experiences using the SCPS Reference Implementation software distribution release. This allows the SCPS project to develop a greater insight into the types of applications and programs considering the SCPS protocols, and to identify those modifications to the Reference Implementation that would make it more useful.

The SCPS team is also working on developing an automated problem reporting system. This system will probably be based on GNU/gnats software. Until this system in completed, it is requested that all problems be written and sent to the MITRE Corporation. Appendix E provides a sample form for reporting problems concerning the SCPS Reference Implementation.

Please direct all comments and suggestions to:

Mr. Robert C. Durst
MS W650
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean, Virginia 22102

List of References

1. 1. *Space Communications Protocol Specification (SCPS)—Network Protocol (SCPS-NP)*. Draft Recommendation for Space Data System Standards, CCSDS 713.0-R-3. Washington, D.C.: CCSDS, September 1997.
2. *Space Communications Protocol Specification (SCPS)—Security Protocol (SCPS-SP)*. Draft Recommendation for Space Data System Standards, CCSDS 713.5-R-3. Washington, D.C.: CCSDS, September 1997.
3. *Space Communications Protocol Specification (SCPS)—Transport Protocol (SCPS-TP)*. Draft Recommendation for Space Data System Standards, CCSDS 714.0-R-3. Washington, D.C.: CCSDS, September 1997.
4. *Space Communications Protocol Specification (SCPS)—File Protocol (SCPS-FP)*. Draft Recommendation for Space Data System Standards, CCSDS 717.0-R-3. Washington, D.C.: CCSDS, September 1997.
5. MIL-STD-2045-43000, Network Protocol for High-stress, Resource-constrained Environments (Draft), 30 September 1997, Standardization Document Order Desk, 700 Robbins Avenue, Building 4D, Philadelphia, PA 19111-5094.
6. MIL-STD-2045-43001, Security Protocol for High-stress, Resource-constrained Environments (Draft), 30 September 1997, Standardization Document Order Desk, 700 Robbins Avenue, Building 4D, Philadelphia, PA 19111-5094.
7. MIL-STD-2045-44000, Transport Protocol for High-stress, Resource-constrained Environments (Draft), 30 September 1997, Standardization Document Order Desk, 700 Robbins Avenue, Building 4D, Philadelphia, PA 19111-5094.
8. MIL-STD-2045-47000, File Protocol for High-stress, Resource-constrained Environments (Draft), 30 September 1997, Standardization Document Order Desk, 700 Robbins Avenue, Building 4D, Philadelphia, PA 19111-5094.
9. Durst, R. C. et al, User's Manual for the SCPS Transport Protocol, MITRE Working Note WN 97 W0000018, March 1997.
10. Durst, R. C. and M. J. Zukoski, User's Manual for the SCPS Network Protocol, MITRE Working Note WN 97W0000026, April 1997.

Appendix A

Manual Page for scps_init Application

SCPS_INIT(1)

NAME

scps_init - sample SCPS based client program

SYNOPSIS

scps_init [-A] [-b size] [-B] [-C] [-D] [-G] [-L] [-M] [-R rate] [-S] [-T] [-Z] host

DESCRIPTION

Scps_init is a sample of a SCPS based client program. Scps_ttcp allows the user to set various SCPS options via the command line.

OPTIONS

-A ackdelay Set the value of the maximum time in milliseconds a SCPS TP entity must send a TP acknowledgment.

-b size Set size of socket buffer in bytes.

-B Enable SCPS Best Effort Transport Service (BETS) operation if supported by the peer.

-C Enable SCPS compression operation is supported by the peer.

-D Don’t buffer SCPS TP writes (sets SCPS NODELAY option).

-G Enable TCP Vegas Congestion control algorithm.

-L Set SCPS socket to non-blocking mode.

-M mtu Sets the Maximum Transmission Unit (MTU) in bytes for this connection.

-R rate Sets the SCPS rate control value in bps.

-S Disables SCPS Selective Negative Acknowledgment (SNACK)

-T Disables SCPS TimeStamp operation.

-Z Enable SCPS BETS, SCPS compression, SCPS SNACK, and SCPS TimeStamp options.

Appendix B

Manual Page for scps_resp Application

SCPS_RESP(1)

NAME

scps_resp - Sample SCPS based server program

SYNOPSIS

scps_resp [-A] [-b size] [-B] [-C] [-D] [-G] [-L] [-M] [-R rate] [-S] [-T] [-Z]

DESCRIPTION

Scps_resp is a sample of a SCPS based client program. Scps_ttcp allows the user to set various SCPS options via the command line.

OPTIONS

-A ackdelay Set the value of the maximum time in milliseconds a SCPS TP entity must send a TP acknowledgment.

-b size Set size of socket buffer in bytes.

-B Enable SCPS Best Effort Transport Service (BETS) operation if supported by the peer.

-C Enable SCPS compression operation is supported by the peer.

-D Don’t buffer SCPS TP writes (sets SCPS NODELAY option).

-G Enable TCP Vegas Congestion control algorithm.

-L Set SCPS socket to non-blocking mode.

-M mtu Sets the Maximum Transmission Unit (MTU) in bytes for this connection.

-R rate Sets the SCPS rate control value in bps.

-S Disables SCPS Selective Negative Acknowledgment (SNACK)

-T Disables SCPS TimeStamp operation.

-Z Enable SCPS BETS, SCPS compression, SCPS SNACK, and SCPS TimeStamp options.

Appendix C

Manual Page for scps_ttcp Application

SCPS_TTCP(1)

NAME

scps_ttcp - test SCPS TP performance

SYNOPSIS

scps_ttcp -t [-u] [-s] [-p port] [-l buflen] [-b size] [-n num-bufs] [-A align] [-O offset] [-f format] [-a] [-D] [-v] [-E] [-C] [-G] [-D] [-M] [-m mtu] [-R rate] [-L priority] [-e] [host] [<in]

scps_ttcp -r [-u] [-s] [-p port] [-l buflen] [-b size] [-A align][-O offset] [-f format] [-B] [-T] [-v] [-E] [-C] [-G] [-a] [-D] [-M] [-m mtu] [-R rate] [-L priority] [-e] [host] [<in] [host] [>out]

DESCRIPTION

Scps_ttcp times the transmission and reception of data between two systems using the SCPS TP protocols. For testing, the transmitter should be started with -t and-s after the receiver has been started with -r and -s. Scps_ttcp also provides the capability of determining round trip throughput. Tests lasting at least tens of seconds should be used to obtain accurate measurements.

Scps_ttcp allows the user to set various SCPS options via the command line.

OPTIONS

-t Transmit mode.

-r Receive mode.

-u Use UDP instead of TCP.

-s If transmitting, source a data pattern to network; if receiving, sink (discard) the data. Without the -s option, the default is to transmit data from stdin or print the received data to stdout.

-l length Length of buffers in bytes (default 8192). For UDP, this value is the number of data bytes in each packet. The system limits the maximum UDP packet length. This limit can be changed with the -b option.

When testing UDP performance, it is important to set the packet size to be less than or equal to the maximum transmission unit of the media. Otherwise, IP fragmentation will distort the test. For Ethernet, set the length to 1508 bytes.

-b size Set size of socket buffer. The default varies from system to system. This parameter affects the maximum UDP packet length. It may not be possible to set this parameter on some systems

-n numbufs Number of source buffers transmitted (default 2048).

-p port Port number to send to or listen on (default 2000). On some systems, this port may be allocated to another network daemon.

-a ackdelay Set the value of the maximum time in milliseconds a SCPS TP entity must send a TP acknowledgment.

-D If transmitting using TCP, do not buffer data when sending (sets the TCP_NODELAY socket option.

-B When receiving data, output only full blocks, using the block size specified by -l. This option is useful for programs, such as tar(1), that require complete blocks.

-A align Align the start of buffers to this modulus (default 16384).

-O offset Align the start of buffers to this offset (default 0). For example, ``-A8192 -O1" causes buffers to start at the second byte of an 8192-byte page.

-f format Specify, using one of the following characters, the format of the throughput rates as kilobits/sec ('k'), kilobytes/sec ('K'), megabits/sec ('m'), megabytes/sec 'M'), gigabits/sec ('g'), or gigabytes/sec ('G'). The default is 'K'.

-T ``Touch" the data as they are read in order to measure cache effects.

-v Verbose: print more statistics.

-d Debug: set the SO_DEBUG socket option.

-E Enable SCPS Best Effort Transport Service (BETS) operation if supported by the peer.

-C Enable SCPS compression operation is supported by the peer.

-G ## Override the default Congestion Control Algorithm
0 = Disable Congestion Control
1 = Van Jacobson Congestion Control
2 = TCP-Vegas Congestion Control

-F ## Set default ACK behavior
0 = Strictly delayed ACKs
1 = ACK every segment
2 = ACK every other segment

-S Disable SCPS Selective Negative Acknowledgment (SNACK)

-M Disable RFC 1323 TimeStamp operation if supported by peer

-m mtu Sets the Maximum Transmission Unit (MTU) in bytes for this connection.

-R rate Sets the SCPS rate control value in bps.

- L Sets the SCPS precedence level.

- e Perform a ping pong (command/response) test.

- c Enable Record Boundaries on SOCK_SEQPACKET sockets.

Appendix D

List of Files in Reference Implementation Distribution

SCPS directory: Top level directory:

File name	Description
Makefile	Makefile generated by running ‘configure’ script
FP	Directory containing the SCPS File Protocol source code
README	README file about SCPS
apps	Directory containing the SCPS application source code
bin	Directory containing SCPS executables
docs	Directory containing SCPS documentation in abobe pdf format
include	Directory containing SCPS include files required by SCPS applications
lib	Directory containing SCPS library required to be linked by SCPS applications.
source	Directory containing SCPS source code for SCPS-TP, SCPS-SP, SCSP-NP, and IP.
tcpdump	Directory containing patch file for tcpdump version 3.4a5 allowing it to understand SCPS lower layer protocols

SCPS *source* directory: SCPS-TP, SCPS-SP, SCPS-NP, IP, and support files:

File name	Description
buffer.c	Buffer management routines
buffer.h	Header file for buffer management routines
compress.c	SCPS-TP header compression and decompression routines
compress.h	Header file for SCPS-TP header compression
errors.c	Error reporting routines
get_local_internet_addr.c	Routines for acquiring IP addresses (remote and local)
ip.c	Routines for building and parsing IP headers
ip.h	Header files to support building and parsing IP headers
ll.h	Header files to support lower-layer interfaces
ll_support.c	Routines for interfacing with lower layers
Makefile	Makefile for SCPS-TP, SCPS-SP, and SCPS-NP
md5.c	Message Digest 5 routines
md5.h	Header file for Message Digest 5 routines

File name	Description
mib.h	SCPS-NP management information base header file
net_types.h	Routines defining lower layer interface out of SCPS-TP
np_scmp.h	Header file for SCPS Control Message Protocol
q.c	Precedence-queuing routines
q.h	Header file for precedence queuing routines
scmp.c	SCPS Control Message Protocol routines
scps.h	General header file for SCPS Reference Implementation
scps_config.c	Configuration parameters for scps initiator application
scps_constants.h	Various constant definitions
scps_defines.h	Internal use constant definitions
scps_globali.c	Global variable definitions for scps initiator application
scps_globalr.c	Global variable definitions for scps responder application
scps_initiator.c	SCPS initiator (client) example application
scps_ip.h	Header file for IP header parsing and formatting routines
scps_np.c	SCPS Network Protocol implementation
scps_np.h	Header file for SCPS Network Protocol
scps_responder.c	SCPS responder (server) example application
scps_sadb.c	SCPS security association database
scps_sadb.h	Header file for SCPS security association database
scps_sp.c	SCPS security protocol
scps_sp.h	Header file for SCPS security protocol
scps_spc.c	Cryptographic functions for SCPS-SP
scpserrno.h	Error number definitions
scpsnp_protos.h	Function prototypes for SCPS-NP
scpstp.h	Header file for SCPS-TP (TCP)
scpsudp.h	Header file for SCPS-TP (UDP)
systype.h	System type information for error routines
thread.c	Thread scheduler routines
thread.h	Header file for thread scheduler routines
tp.c	Busy wait loop for SCPS-TP implementation
tp.h	Header file for SCPS-TP internal use
tp_checksum.c	Internet checksum routines for SCPS-TP
tp_debug.c	Buffer integrity checking routines for SCPS-TP
tp_handler.c	Inbound packet demultiplexing routines for SCPS-TP (TCP)
tp_output.c	Output routines for SCPS-TP (TCP)
tp_outseq.c	Out-of-sequence queue routines for SCPS-TP (TCP)
tp_process.c	Receive processing routines for SCPS-TP (TCP)

File name	Description
tp_socket.c	Socket interface routines for SCPS-TP (TCP)
tp_sockopt.c	Socket option routines for SCPS-TP (TCP) and related layers
tp_timers.c	SCPS-TP (TCP) timer routines
tp_utility.c	General support routines for SCPS-TP (TCP)
udp_handler.c	UDP demultiplexing routines
udp_output.c	UDP output processing routines
udp_socket.c	UDP socket interface routines
udp_utility.c	UDP general support routines
wheel_timers.c	Timer management routines

SCPS *FP* directory: SCPS-FP files:

File name	Description
Makefile	Makefile generated by running ‘configure’ script
Makefile.in	Initialization file for ‘configure’ script
autopa.c	Placeholder for user and password routines for autorestart
bincmp.c	Binary compare routines
bincmp.h	Header file for binary compare routines
cmdca.c	Client commands for base implementation
cmdsa.c	Server commands for base implementation
configure	Shell script to generate Makefile
crc.c	IEEE 802.3 Cyclic redundancy check computation routine
crc.h	Header file for CRC routine
crcchk.c	Routine to calculate and report CRC
crcs.c	Routine to calculate the SCPS CRC over a file’s contents
date.c	Routine to provide compile time, date, and build size
edit.c	Applies record updates to file, writes result to output file
edit.h	Header file for record update
ftp.h	Header file for ftp
libc.c	Library routines for client
libc.h	Header file for client library routines
libs.c	Library routines for server
logc.c	Routine to write client logfile
logs.c	Routine to write server logfile

File name	Description
mib.h	Header file for Management Information Base
prtstat.c	Performance benchmarking routines
prtstat.h	Header file for performance benchmarking routines
rx_avail.c	Routine to determine whether there is readable data on a socket
rx_avail.h	Header file to support rx_avail
scpsdiff.c	Routine to create list of records for updating a file
server.c	SCPS FP server
sfp.c	SCPS-FP client
sortupdt.c	Routine to sort update records
tpif.h	Header file for transport layer interface

SCPS *apps* directory: Application Files:

File name	Description
Makefile	Makefile generated by running 'configure' script
Makefile.in	Initialization file for 'configure' script
Configure	Configure script used to create Makefile
scps_config.c	Configuration parameters for SCPS client/server example.
scps_initiator.c	Example of SCPS client application
scps_responder.c	Example of SCPS server application
ttcp.c	Benchmarking application source code

SCPS *bin* directory: Executable Files:

File name	Description
Makefile	Makefile supporting the "make clean" operation
SA_table	Sample initialization file for Security Association database - must be modified to suit your network
npIP_NP_File	Sample address translation specifications for SCPS-NP - must be modified to suit your network
npNextHopFile	Sample static routing table for SCPS-NP - must be modified to suit your network
SAMPLE.SA_table SAMPLE.npIP_NP_File SAMPLE.npNextHopFile	Write-protected versions of the above, for reference.

SCPS *docs* directory: Documentation about SCPS:

File name	Description
NP_Users_Manual.pdf	User's manual of the SCPS Network Protocol in abobe pdf format
TP_Users_Manual.pdf	User's manual of the SCPS Transport Protocol in abobe pdf format
FP_Readme.pdf	Readme file for the SCPS File Protocol in adobe pdf format
RI.pdf	User's manual of the SCPS Reference Implementation in abobe pdf format (this document)

SCPS *include* directory: Include Files:

File name	Description
scps.h	SCPS header file to be included by all applications
scpserrno.h	SCPS header file to be included by all applications

SCPS *lib* directory: Application Files:

File name	Description
Makefile	Makefile supporting the "make clean" operation

SCPS *tcpdump* directory: Tcpdump patches:

File name	Description
README	README describing how to apply the patch to tcpdump
apply_scps_tcpdump_patch	Bourne shell script to patch tcpdump
tcpdump-3.4a5.scps.diff	Patch file containing diffs for tcpdump allowing it to understand the SCPS lower layer protocols

Appendix E

Problem Report Form

Name of Organization:
Address of Organization:
Point of Contact:
E-mail Address:
Date:
Classification of Problem Found:
Summary of Problem Found:
Version of SCPS Code:
Configuration:
Environment:
Platforms:
Scenarios:
Detailed Description of Problem: (include copy of Makefile and packet traces, if applicable)

Glossary

BETS	Best Effort Transport Protocol
DISA	Defense Information Services Agency
DOD	Department of Defense
GCC	GNU C Compiler
GNU	GNU Not Unix
IP	Internet Protocol
NASA	National Aeronautics and Space Administration
NP	Network Protocol
SCMP	SCPS Control Message Protocol
SCPS	Space Communication Protocol Standards
SCPS-FP	SCPS File Protocol
SCPS-NP	SCPS Network Protocol
SCPS-SP	SCPS Security Protocol
SCPS-TP	SCPS File Protocol
SMC	Space and Missile Systems Command Defense Information Services Agency
SNACK	Selective Negative Acknowledgment
TCP	Transmission Control Protocol
TP	Transport Protocol
UDP	User Datagram Protocol