1 Overview[1][2][3]

This document describes the SCPS transparent transport layer gateway shipped with the SCPS Reference Implementation. This is written to help users better understand and configure the SCPS gateway. The following describes the sections within this document. Section 2 provides an overview of the SCPS protocols. Section 3 provides background on the transparent gateways. Sections 4 through 7 provide detail instructions on how to compile the SCPS gateway. Section 8 describes how to configure the SCPS gateway to work best in its deployed environment. Sections 9 through 13 describe advanced features provided with the SCPS gateway. Appendix A describes the directives that can be used to configure the gateway. Appendix B describes the interaction between TCP and satellite links. Finally, Appendix C describes some network analysis tools that have been used at MITRE.

[1] Approved for Public Release; Distribution Unlimited

[2] ©2005 The MITRE Corporation. All Rights Reserved

[3] The research described in this paper was carried out for the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration

2 Background on SCPS

The Space Communication Protocol Standards (SCPS) originated in 1992, when elements of NASA and the US Air Force jointly commissioned the development of a specialized suite of data transfer protocols expressly designed for the stressed environment of satellite communications. A paramount design goal of SCPS is that it had to be based on current Internet protocols and fully interoperable with IETF standards. SCPS was designed to meet the following goals:

- Best possible use of limited bandwidth
- High link utilization
- Conservation of power
- Prioritization of traffic
- Tolerant of intermittent connectivity
- High forward/return link asymmetry

In addition to the performance objectives, the SCPS consortium was looking to establish these protocols as an international standard for the space networking community. The SCPS suite[SCPS FP] [SCPS TP] [SCPS SP] [SCPS NP] has been adopted as standards by the International Standards Organization (ISO), the Consultative Committee on Space Data Systems (CCSDS) and the US Department of Defense. Figure 1 describes the standards.

Protocol	CCSDS	MIL	ISO
	Standards	Standards	Standards
SCPS Network Protocol (SCPS-NP)	713.0-B-1	2045-43000	15891
SCPS Security Protocol (SCPS-SP)	713.5-B-1	2045-43001	15892
SCPS Transport Protocol (SCPS-TP)	714.0-B-1	2045-44000	15893
SCPS File Protocol (SCPS-FP)	717.0-B-1	2045-47000	15894

Figure 1 SCPS Standards Numbering Scheme for Various Standards Bodies

3 Background on TCP Gateways

It is widely known that the Internet's TCP protocol, which forms the basis for many popular applications (e-mail, file transfer, web transfer) has performance degradation issues when operating over satellite and other wireless technologies as compared to wired technologies. This performance degradation can be quite severe. Appendix B provides a detailed explanation of these issues.

TCP gateways, operating at the transport layer, increase the performance of TCP based applications where native performance suffers due to characteristics of a link of subnetwork on the path. These gateways do not modify the application protocol, thus applications operate end-to-end and are totally transparent to the applications. Technically, these gateways perform a technique called spoofing in which they intercept the TCP connection in the middle and terminate that connection as if the gateway is the intended destination. Gateways, typically bracketing the satellite network, split a single TCP connection into three separate connections. These gateways communicate via standard TCP when talking to each of the end-systems and a third connection using an optimized rate based protocol is used when transferring the data between them. This technique allows essentially one to isolate the properties of satellite networks that degrade performance from manifesting itself to TCP. For example, corruption loss on the satellite network does not cause the transmission rate to be cut in half, or congestion loss on the terrestrial network does not cause data to be retransmission consuming satellite bandwidth that may be a scarce resource. This also allows the deployment of protocols that can be tuned to match the various characteristics of the satellite link without effective the end-systems. Now the default TCP parameters on most end-systems are now applicable for the terrestrial environment in which they operate.

The SCPS Transport Protocol was not originally designed for as a PEP solution. It was originally developed by the U.S. Department of Defense (DoD) and international space agencies to meet the challenges of their unique environments. Meanwhile as the wide spread adoption of the Internet continued, satellite and other wireless technologies extended the Internet in environments not possible or feasible with wired technologies. Performance issues with the Internet's TCP protocol when operating in a non-terrestrial environment surfaced. The SCPS Transport Protocol could be useful in solving these challenges. The simplest method to introduce this new technology was via transport layer gateways. Therefore, applications without modifications could benefit from the SCPS protocols.

3.1 Sample Topology

Figure 1 depicts a typical scenario involving split gateway to improve applications performance. The gateways implements two transport layer protocols; TCP to communication with end-systems ensuring the users of the system do not need to modify their systems; and the SCPS TP protocol to communicate between gateways. Once a gateway receives and acknowledges the data sent via TCP, it is responsible to reliably transmit the data to its remote peer. Once the remote gateway receives and acknowledges the data sent via protocol SCPS TP, it is responsible to reliably transmit it to the remote end-systems.

It should be noted that if security is required, it typically must be applied above the transport layer which allows the TCP headers to remain in the clear. Thus is disables the end-to-end use of IPSEC.

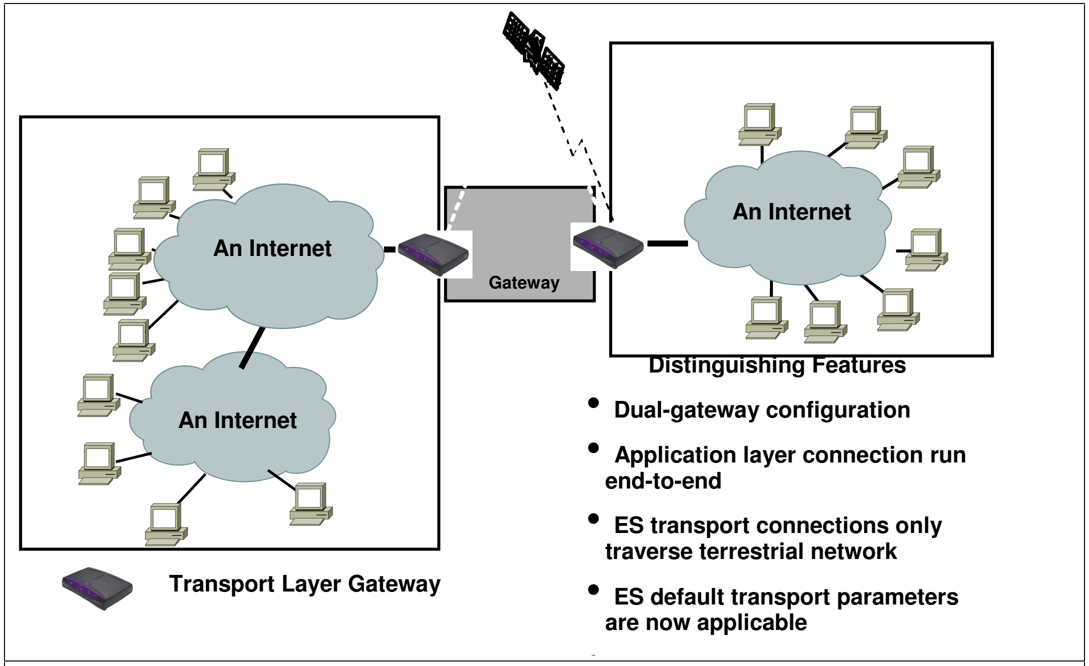
An Internet
An Internet
Gateway
An Internet
Distinguishing Features
• Dual-gateway configuration
• Application layer connection run end-to-end
• ES transport connections only traverse terrestrial network
• ES default transport parameters are now applicable
Transport Layer Gateway

Figure 1 SCPS Gateway Deployment Scenario

4 Overview of SCPS Gateway Configuration

The SCPS transparent gateway will operate on either the FreeBSD or the Linux operating system. It was originally developed for the FreeBSD operating system. However due to the popularity of the Linux operating systems, the SCPS gateway was ported to this operating system.

To run on the FreeBSD operating system, the divert functionality must be enabled. See section 5 on configuring and compiling the SCPS gateway under the FreeBSD operating system.

To run on the Linux operating system, three separate methods may be used. Initially the SCPS gateway used a ported divert mechanism; however this method is not recommend and hence will not be discussed. The gateway may either use the TUN method as described in section 6 or the TAP method as described in section 7. The TUN method is similar to the divert method in FreeBSD. Both operate at the network layer and can use the kernel routing interface to send and receive traffic. Hence they require the use of external routing table entries to forward the data flow. The TAP method operates at the link layer, bridging between to Ethernet interfaces. Thus this method is more easily used to insert the technology into an already existing network.

The SCPS RI gateway has experienced problems compiling with newer versions of gcc. If this occurs add the additional configuration option to the ./configure script in the source directory

```
--gateway_single_thread=yes
```

5 Compilation for FreeBSD – Divert

Before compiling the SCPS code, the underlying kernel support number must be enabled. In order to run the SCPS gateway under the FreeBSD operation system, the following must be modified in the FreeBSD kernel

- The maximum socket buffer size must be increased.
- The default socket buffer size for divert sockets must be increased.
- The kernel must be built to include the FreeBSD divert and firewall options.

These changes along with the rest of the steps in this section must be performed by a system administrator or someone with root privileges.

To change the maximum socket buffer size, edit the file /usr/src/sys/sys/socketvar.h and make the following change at approximately line 93. Change

```
        #define SB_MAX    (256*1024)     /* default for max chars in sockbuf */
        to
        #define SB_MAX     (1024*1024)    /* default for max chars in sockbuf */
```

Next the default buffer sizes for the divert socket must be changed by editing the file /usr/src/sys/netinet/ip_divert.c and make the following change at approximately line 64. Change

```
        #define DIVSNDQ       ((65536) + 100)
        #define DIVRCVQ       ((65536) + 100)
        to
        #define DIVSNDQ       (240000)
        #define DIVRCVQ       (240000)
```

Finally, the kernel configuration file in the directory /usr/src/sys/i386/conf must be modified to incorporate the divert utility. The following options must be added to your kernel configuration file.

```
options       IPDIVERT             #Divert Socket
options       IPFIREWALL             #Firewall code
```

Now the kernel should be built and installed as normal. It should be noted that the default configuration of the IPFIREWALL utility is to disallow all network access to the machined. Before rebooting the machine the following line needs to be added to the end of the file /etc/rc right before the statement "exit 0" to again allow network access to the machine.

ipfw add 30000 allow all from any to any

Once these are compiled into the kernel, the next step is to configure the makefiles in the application directory followed by the source directory. From the main SCPS RI directory enter the following commands.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes
cd ..
make clean;make
```

If all compiles cleanly, the executable ‘gateway’ will be installed in the bin directory. Please note that there are other applications in the bin directory. Do not execute these binaries. These are for the standalone system and must be configured differently.

6 Compiling for Linux 2.4 Based Kernels- TUN

Before compiling the SCPS code, the underlying kernel support number must be enabled. The following supporting software must be obtained and installed from other sources if not already present on the system.

- tun/tap modules from http://vtun.sourceforge.net/tun
- iproute2 from ftp://ftp.sunet.se/pub/Linux/ip-routing
- iptables from http://netfilter.samba.org

In order to run the SCPS gateway under the Linux operation system via the TUN method, the following kernel configuration options must be enabled before configuring the SCPS software.

```
UNDER NETWORKING OPTIONS
        Kernel/User network socket
        Netlink packet filtering
        Advanced Router
                IP policy routing
                IP use netfilter MARK value as routing key
        IP Netfilter Configuration
                IP tables support
                netfilter MARK match support
        Packet mangling
                MARK target support
UNDER NETWORKING DEVICE
        Universal TUN/TAP device driver support.
```

Once these are compiled into the kernel, the next step is to configure the makefiles for the SCPS gateway. First configure the makefiles in the application directory followed by the source directory. From the main SCPS RI directory enter the following commands.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes --tun=yes
cd ..
make clean;make
```

If all compiles cleanly, the executable 'gateway' will be installed in the bin directory. Please note that there are other applications in the bin directory. Do not execute these binaries. These are for the standalone system and must be configured differently.

The following directives must be added to the resource file.

```
AIF_TUN_NAME     tun0
BIF_TUN_NAME     tun1
C_TUN_NAME       tun2
```

7 Compiling for Linux 2.4 Based Kernels - TAP

Before compiling the SCPS code, the underlying kernel support number must be enabled. The following supporting software must be obtained and installed from other sources if not already present on the system.

- tun/tap modules from http://vtun.sourceforge.net/tun
- brctl from http://bridge.sourceforge.net

In order to run the SCPS gateway under the Linux operation system via the TAP method, the following kernel configuration options must be enabled before configuring the SCPS software.

```
UNDER NETWORKING OPTIONS
        802.1d Ethernet Bridging
UNDER NETWORKING DEVICE
        Universal TUN/TAP device driver support
```

Once these are compiled into the kernel, the next step is to configure the makefiles for the SCPS gateway. First configure the makefiles in the application directory followed by the source directory. From the main SCPS RI directory enter the following commands.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes --tap=yes
cd ..
make clean;make
```

If all compiles cleanly, the executable ‘gateway’ will be installed in the bin directory. Please note that there are other applications in the bin directory. Do not execute these binaries. These are for the standalone system and must be configured differently.

The following directives must be added to the resource file.

```
AIF_TAP_NAME    tap0
BIF_TAP_NAME    tap1
```

By default you can not access the gateway via either of the two interfaces when using the tap interface. To change this behavior, add the following to the resource file.

```
C_TAP_REMOTE_ACCESS        1
```

8 Configuring the SCPS Gateway

Now that the SCPS gateway has properly compiled, the gateway needs to understand the networks it connects. This is done via the directives in the gateway's resource file or 'rfile'. This rfile contains all the information about the network enabling the SCPS gateway increase the performance of TCP based applications. The rfile is defined into three sections, the first section describes the 'A interface' the second describes the 'B interface', and the third contains information common to both interfaces. By convention the 'A interface' is associated with the terrestrial side while the 'B interface' is associated with the RF side. Appendix A contains a complete list of all directives. Within that section the prefixes AIF and BIF are replaced by XIF. Simply substitute the X with either A for the 'A interface' and B for the 'B interface'.

8.1 LAN Network

The LAN network is the easiest to configure since a default configuration should work in most environments. The directives involved are: AIF_NAME, AIF_BUF, AIF_CC, AIF_RATE, and AIF_TCPONLY

- AIF_NAME is the name of the interface that connects the gateway to the LAN side. For example in FreeBSD, it may be sis1, while for Linux it may be eth1.
- AIF_BUF is the size of the transport layer's send and receive buffers. A common value for a LAN network is 32768
- AIF_CC is the congestion control algorithm the gateway will run on the LAN side. Unless situation dictates otherwise, the VJ congestion control algorithm should be run, therefore the value should be 1.
- AIF_RATE, is the minimum of the maximum rates on the LAN side. Typically, Ethernet is the common network, therefore either 10000000 or 100000000 should be used. Note that if a T1 or other link is involved then this directive should be set to 1536000.
- AIF_TCPONLY indicates the SCPS TP extensions should not be used on this network, even though the SCPS uses techniques registered with IANA to properly negotiate the SCPS options. The TCP standard states that if that option is well formed and not implemented that option should be ignored. Some operating systems unfortunately improperly handle TCP options not understood by their implementation. Therefore to accommodate these broken implementations of TCP, this value should be set to 1 to not negotiate SCPS on the LAN network.

8.2 WAN Network

The WAN network is more difficult to configure properly. As for the LAN side, I will present the most common directives and their values. Following this, I will discuss the use of the other directives. The common directives involved are: BIF_NAME, BIF_BUF, BIF_CC, BIF_RATE, and BIF_MINRTO.

- BIF_NAME is the name of the interface that connects the gateway to the WAN side. For example in FreeBSD, it may be sis2, while for Linux it may be eth2.
- BIF_BUF is the size of the transport layer's send and receive buffers. Typically, this value is set to twice a value known as the Bandwidth Delay Product (BDP). To obtain the BDP, take the value of the bandwidth and multiply it by the round trip delay then divide by 8 to get bytes. For example if the bandwidth of the satellite is 2 Mbps and the round trip time is 600 milliseconds, then the BDP would be 150,000 bytes. Multiply this by 2 and the value of BIF_BUF is 300000.
- BIF_CC is the congestion control algorithm the gateway will run on the WAN side. If the entire bandwidth of the satellite is dedicated to the gateway process, then set this to pure rate control – a value of 0. If this bandwidth is shared by sources other than the gateway, then a congestion control algorithm should be used. Either value of 1 for the Vegas congestion control algorithm or a value of 2 for the standard VJ congestion control algorithm should be used.
- BIF_RATE is the minimum of the maximum rates on the WAN side. Most commonly it is the bandwidth of the satellite link which the gateway has access to. In the example, this value should be set to 20000000
- BIF_MINRTO is the minimum value of the retransmission timer in microseconds. It should be set to the round trip time of the satellite link. In the example, this value should be set to 600000.

Figure X shows a sample rfile based on the above description for the FreeBSD operating system.

```
# Gateway Resource File - comments are allowed, starting with '#'.
# Comments can be anywhere EXCEPT between directives and their values.

# A-interface information for the terrestrial side.
AIF_NAME        sis1
AIF_BUF         32768
AIF_CC          1
AIF_RATE        10000000
AIF_TCPONLY     1

# B-interface information for the satellite side.
BIF_NAME        sis2
BIF_BUF         300000
BIF_RATE        2000000
BIF_CC          0
BIF_MINRTO      600000
```

Sample Resource File

8.3 Complete List of Resource File Directives

As previously indicated, there are many other directives that can be used to configure the gateway. The following is a list of the directives and guides to possible values.

- XIF_TS controls the negotiation and use of transport layer timestamps. Unless bandwidth is constraining, this should always be enabled, which is the default. To disable transport layer timestamps set the value of this directive to 0.
- XIF_SNACK controls the negotiation and use of the transport layer Selective Negative ACKnowledgement (SNACK). This should always be enabled, which is default. To disable SNACK set the value of the directive to 0.
- XIF_TP_COMPRESS enables the SCPS transport layer header compression algorithm. This should be used on low bandwidth links. This algorithm is disabled by default. To enable transport layer header compression set the value of this directive to 1.
- XIF_MTU defines the interface's Maximum Transmission Unit (MTU). By default the gateway reads the MTU associated with the interface XIF_NAME. To explicitly set the MTU set the value of this directive to the desired MTU (in bytes)
- XIF_SMTU defines the maximum packet side that the gateway will emit. This differs from the XIF_MTU in that XIF_MTU is the maximum packet side the interface can receive. By default the gateway sets the directive to the value of the XIF_MTU directive. To explicitly set the SMTU set the value of this directive to the desired MTU (in bytes)
- XIF_RBUF is the transport layer's receive buffer size. This directive is used when bandwidth on the forward and return channel are different. By default the receive buffer size the same as the send buffer size. To explicitly set the receive buffer size set the value of this directive to the desired size (in bytes).
- XIF_ACK_DELAY changes the value of the delayed acknowledgement timer from its default of (200 milliseconds). This directive in conjunction with the XIF_ACK_BEHAVE directive is typically only changed when throughput is restricted because of being 'receive window limited'.
- XIF_ACK_BEHAVE changes the behavior of sending TCP acknowledgements. This directive in conjunction with the XIF_ACK_DELAY directive is typically only changed when throughput is restricted because of being 'receive window limited'. If throughput is receive window limited you may enable strictly delayed acknowledgements set this parameter to 0. To acknowledge every segment set this parameter to 1. To acknowledge every other segment (the default) set this parameter to 2.
- XIF_IRTO changes the initial value of the retransmission timer from the default of 6 seconds. The value of this parameter is in microseconds.
- XIF_MAXRTO changes the maximum value of the retransmission timer from the default of 60 seconds. The value of this parameter is in microseconds.
- XIF_SNACK_DELAY adds a delay in milliseconds from reception of SNACK and corresponding action. The default is 0, (i.e., react immediately to the

reception of a SNACK.) but may be changed for a variety of reasons including striping/duplicating packets from a single connection across multiple links.

- XIF_VEGAS_ALPHA is the threshold from the Vegas congestion control algorithm to increase its rate during the congestion avoidance phase.
- XIF_VEGAS_BETA is the threshold from the Vegas congestion control algorithm to decrease its rate during the congestion avoidance phase.
- XIF_VEGAS_GAMMA is the threshold from the Vegas congestion control algorithm to terminate the slow start phase and enter the congestion avoidance phase.
- XIF_VEGAS_SS changes the behavior during the slow start phase of the Vegas congestion control algorithm. A value of 0 (default) double the congestion window every round trip time. A value of 1 double the congestion window every other round trip time (as defined by Brakmo's Paper)
- XIF_NL determines the network layer used to emit packets. A value of 1 (default) for the IP protocol, while a value of 2 is for the NP protocol. Note that if the NP protocol is used, there is no native support for NP hence UDP encapsulation must also be used
- XIF_SCPS_SECURITY configures the use of the SCPS security protocol. Note that the SCPS security must be compiled in into the gateway before use. The value of 0 (default) mean don't use the security protocol. A value of 1, used mainly for debugging, indicated the security protocol should only be used if an entry in the security association data base is present. A value of 2 indicated the security protocol must be used.
- XIF_MPF controls the use of Multiple Path Forwarding (MPF.) MFP is a technique to duplicate packets across multiple disjoint subnetwork to ensure a timely and reliable delivery of messages. A value of 0 (default) disables MFP while a value of 1 enables it. MPF must be enabled at compile time.
- XIF_MPF_SRC is used multiple times to list the source addresses that are to duplicate the packet, encapsulate it, and route packets through different subnetworks.
- XIF_MPF_DST is used multiple times to list the source addresses that are to duplicate the packet, encapsulate it, and route packets through different subnetworks.
- XIF_LAYERING enables the use of UDP encapsulation for forward packets between the two SCPS Gateways. A value of 1 enables UDP encapsulation while a value of 0 (default) disables UDP encapsulation.
- XIF_LOCAL_IP contains the source IP address of the outer UDP encapsulation packet if UDP encapsulation is enabled
- XIF_REMOTE_IP contains the destination IP address of the outer UDP encapsulation packet if UDP encapsulation is enabled
- C_REMOTE_UDP_PORT contains the source UDP port number of the outer UDP encapsulation packet if UDP encapsulation is enabled
- C_LOCAL_UDP_PORT contains the destination UDP port number of the outer UDP encapsulation packet if UDP encapsulation is enabled
- XIF_OVERHEAD contains the number of bytes if UDP encapsulation is used. This parameter should be set to 28.

- XIF_MSS_FF contains the number of additional bytes if UDP encapsulation is used that will be consumed by the rate control This parameter should be set to 28.
- XIF_DIVPORT changes the default divert port numbers which identify incoming connection requests. The default for AIF_DIVPORT is 53000 and the default for BIF_DIVPORT if 53001.
- C_DIVPORT changes the default port numbers which are used to gather all traffic but the initial connection request. The default for C_DIVPORT is 53002.
- C_DIVERT_START_RULE is only used by the FreeBSD operating system to identify the starting rule number. This directive is used to change the divert rule numbers if there is a conflict with other firewall utilities. The default is 10000.
- C_DIVERT_INSERT_RULE is only used by the FreeBSD operating system to identify the rule number that the gateway should start searching after it reinserts the packets back into the kernel.. This directive is used to change the divert rule numbers if there is a conflict with other firewall utilities. The default is 10008.
- XIF_TUN_NAME is the name of the TUN interface. The default for AIF_TUN_NAME is tun0 while the default for BIF_TUN_NAME is tun1.
- C_TUN_NAME is the name of the TUN interface. The default for C_TUN_NAME is tun2
- XIF_TAP_NAME is the name of the TAP interface. The default for AIF_TAP_NAME is tap0 while the default for BIF_TAP_NAME is tap1.
- C_OTHER_PROTO_QLEN controls the length of queues for non-TCP based traffic and it only applicable for TAP based access. The default packet length is 5. After the queue length is exceeded, incoming packets will be dropped. This mechanism is used to control the flow of non-TCP based traffic though the WAN side.
- C_OTHER_PROTO_XRATE_DROP controls the policy for non-TCP based traffic if rate is not available and is only applicable for TAP based access. A value of 0 (default) indicated that packets should not be dropped if rate does not exist. The queue will slowly grow and packet will be dropped at the tail of the queue. A value of 1 indicated that packets will be dropped if rate does not exist.
- C_NETSTAT_INTERVAL periodically logs the number of clusters and sockets currently in use syslog. The default is to disable logging. This directive can be used to determine the number of active connections through the gateway and memory usage over time.
- C_CLUST_THRESH and C_CLUST_FILENAME are used to prohibit new connections to be established through the gateway. If the number of clusters used is greater than the value of C_CLUST_THRESH and the filename specified in C_CLUST_FILENAME exists then new connection are reset. There are no default values for these directives, by default this functionality is disabled.
- C_PKT_IO_FILENAME traces a packet as it goes through the gateway. Traces are logged to syslog only if the filename specified in C_CLUST_FILENAME exists.

9 Advanced Configuration Options – Memory Pool

The SCPS gateway has a predefined pool of memory to accommodate the transport layer buffers, which is are source of most of the memory requirements of the gateway. A cluster (32768 bytes of memory) is the basic building block for memory allocation. Currently the gateway allocates 10 Megabytes of memory for clusters; therefore the default is to have 307 clusters for transport layer buffers. Clusters are allocated/deallocated dynamically based on current need. Each socket, the basic transport layer connection unit in the gateway, has two buffers; a send and receive buffer. The sender buffer is a retransmission buffer (i.e., to store data for possible retransmission) while the receive buffers as an out-of-sequence queue (i.e., to save packets received by the sender that are not in sequence to avoid - the possible need for the sender to retransmit them.) Each connection through the gateway requires two sockets, one for the LAN side connection and the other for the WAN side connections. Therefore at a minimum, each connection requires 4 clusters.

You will need to size the transport layer buffers based on the characteristics of the two networks the gateway is connecting. For example if you are operating over a 2 Mbps link with a round trip time of 600 milliseconds, then you may set the buffer sizes to 300,000 bytes on the WAN side. This would require 20 clusters for both the send and receive buffer. On the LAN side, you would probably set the buffer sizes to 32768. This would require 2 clusters, 1 for the send buffer and 1 for the receive buffer. Therefore the maximum total number of clusters for a single gateway connection would be 22.

Depending on the type of traffic and environment, new connections may not be established because of memory (cluster) limitations. During the compilation of the gateway you are able to change the amount of memory allocated to the cluster pool via the directive *–memory*.

For example, if you are running on FreeBSD and decide you need to set the buffers to 20 Mbps, then you would configure and compile the code like the following.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes --memory=20000000
cd ..
make clean;make
```

Remember, the system must have enough physical memory to support the gateway needs along with the needs of the other processes on the system. The gateway does not check if swapping or other system memory management techniques are required because of a lack of enough physical memory to support all of the processes running. If swapping occurs, performance will degrade.

10 Advanced Configuration Options – Maximum Number of Sockets Supported

The default of the SCPS gateway is to support about 126 sockets. This results in at most 63 connections through the gateway. During the compilation of the gateway you may to double the number of sockets via the directive *–gateway_larger*.

For example, if you are running on a Linux 2.4 kernel with the TUN interface and decide to double the number of sockets that can be allocated, you would configure and compile the code like the following.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes –tun=yes –gateway_larger-=yes
cd ..
make clean;make
```

Remember, you may need to change the size of the cluster pool to use the sockets.

If still more sockets are required, you may modify the source code directly. The three files need to be changed are source/thread.h, source/scps_defines/h. and include/scps.h. Look for the C define statement *#ifdef GATEWAY_LARGER*. It should be obvious which arrays you need to change.

11 Advanced Configuration Options – Multiple Routes

The SCPS gateway has the ability to allocate bandwidth dedicated to certain types of traffic. For example, let's assume the SCPS has been deployed in a hub and spoke environment, where a single SCPS gateway at the hub communicates with two sites – each via a SCPS gateway. Site A's prefix is 10.10.1.X/24 and is connected via a 2 Mbps satellite channel. Site B's prefix is 10.10.2.X/24 and is connected via a 512 Kbps satellite channel. The SCPS gateway has the ability to communicate to Site A, with rate control set to 2 Mbps, while communicating to Site B, with the rate control set to 512 Kbps. This is done by creating new routes. Each route is characterized by a *key* to match traffic flow and the corresponding routing parameters. Initially there are two default routes that are created 'interface' (i.e., terrestrial or satellite) by reading the gateway's resource file.

Implementing this function requires: additional compilation options to the SCPS gateway and the use of external 'gw_route_cmdr' utility to create these new routes. The gw_route_cmdr application allows you to add/delete/modify additional routes. Technically you can delete the default route, although I've never done this before and would not recommend it. However you can modify the default routes. For example, something happened and your bandwidth was temporary reduced by half.

During the configuration of the gateway the compilation flag--*gateway_router* and --*gateway_ri_console* enables this functionality. For example, if you are running on a FreeBSD kernel and want to enable this function, configure and compile the code like the following.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes --gateway_router=yes --gateway_ri_console =yes
cd ..
make clean;make
```

To compile the gw_route_cmdr application you need to change directories to utils/gw_route_cmdr. Before compiling the software, if you are running on a Linux operating system, you need to exit the Makefile and change the line from:
SYMTAB = -g -Wall
to
SYMTAB = -g –Wall –DLINUX

To compile the software type in the following command.

```
Make
```

To add a route you must specify the key that will be used during the route lookup process. Unlike standard routing tables that only use the destination IP address to lookup a route, the SCPS routing table is extended to also use the source IP address as well as the source and destination transport layer port numbers. The following ordered list describes fields that will be used to find the best matching route.

1. Src/Dst IP address and Src/Dst TCP port number
2. Dst IP address and Dst TCP port number
3. Src/Dst IP address
4. Dst IP address
5. default

In other words a route whose key is specified by the source and destination IP address is a better match than a route whose key is only specified by is destination IP address.

Each route contains a few parameters that characterize the traffic flow through the interface. The following list describes these attributes.

- Maximum rate control in bps (default is 2 Mbps)
- Maximum Transmission Unit in bytes (default in 1500 bytes)
- Sending Maximum Transmission Unit in bytes (default is 1500 bytes)
- Congestion control method
    1. pure rate control
    2. VJ congestion control (default)
    3. Vegas congestion control

Each route within gw_route_cmdr is identified to the user by a route id. This will be returned to the user when adding routes; shown to the user for listing routes; identifies specific routes for modifying and deleting.

Now, we will present the syntax of gw_route_cmdr. It may appear a bit kludgy at first glance. However, I developed the interface in this manner, so it could be used by graphical user interfaces.

```
Usage: gw_route_cmdr
        -A ##   Command: (A)dd (D)elete (G)et (M)odify (L)ist
        -G ##   Set the IP address of the gateway
        -S ##   Set the source IP address
        -s ##   Set the soiurce netmask-
        -D ##   Set the destination IP address
        -d ##   Set the destination netmask
        -X ##   Set the source transport low port number
        -x ##   Set the source transport high port number
        -C ##   Set the destination transport low port number
        -c ##   Set the destination transport high port number
        -R ##   Set the rate
        -M ##   Set the receiving MTU
        -m ##   Set the sending MTU
        -r ##   Set the route id
        -p ##   Set the protocol id
        -T ##   Set the congestion control algorithm
```

Example

If we go back to the previous example, traffic to Site A (10.10.1.X/24) is rated out at 2 Mbps, while traffic to Site B (10.10.2.X/24) is rated out at 512 Kbps. I will assume that the IP address of the hub' gateway is 10.10.3.1. Type in the following

```
gw_route_cmdr -A A -G 10.10.3.1 -D 10.10.1.1 –d 255.255.255.0 -R 2000000 -T 0
gw_route_cmdr -A A -G 10.10.3.1 -D 10.10.2.1 –d 255.255.255.0 -R 512000 -T 0
```

To list the results of the routing table type in the following.

```
gw_route_cmdr -A L -G 10.10.3.1
```

To delete the two routed tables we just entered type in the following

```
gw_route_cmdr -A d -G 10.10.3.1 -r 4
gw_route_cmdr -A d -G 10.10.3.1 -r 3
```

It should be noted that since gw_route_cmdr required the IP address of the router, it allows for each remote control of different gateways from the same *gw_route_cmdr*.

12 Advanced Configuration Options – SCPS Security Protocol – Beta Version

The SCPS gateway allows the optional use of the SCPS Security Protocol. The SCPS SP is smaller non-interoperable cousin of IPSEC with less overhead and a little less functionality. Some of the key features of the SCPS SP are:

- Confidentiality, integrity, authentication, access control all present
- Algorithm independent
- One security association per host pair
- Replay attacks are prevented via transport layer sequence numbers

During the compilation of the gateway you are able to enable the option use of the SCPS security protocol via the directive --secure_gateway

For example, if you are running on the FreeBSD operating system and decide to enable the SCPS security protocol, you would configure and compile the code like the following.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes --secure_gateway=yes
cd ..
make clean;make
```

The resource file directive XIF_SCPS_SECURITY determines the use of the SP protocol on each side of the gateway one enabled via compile time.

Need to talk about eh SADB LATER XXX ?

13 Advanced Configuration Options – UDP Encapsulation – Beta Version

The SCPS gateway may encapsulate the flow of data between the two gateways via the UDP protocol. For example, this may allow users to direct the gateway traffic to a third machine.

During the compilation of the gateway you are able to enable the UDP encapsulation mechanism via the directive -- gateway_dual_interface

For example, if you are running on the FreeBSD operating system and decide to enable the UDP encapsulation mechanism, you would configure and compile the code like the following.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes -- gateway_dual_interface =yes
cd ..
make clean;make
```

Directives are required in the resource file to support this operation. First you need to enable the encapsulation of the interface via the XIF_LAYERING option. Setting it to 1 enables UDP encapsulation while setting to 0 (the default) disable UDP encapsulations. Next you need to specify the source and destination IP address via the XIF_LOCAL_IP and XIF_REMOTE_IP directives respectively. The source address is the address associated with the gateway – from which the packet will be emitted, and the destination address is the address to which the packet will be addressed. In addition, you need to specify the source and destination port number via the directives C_LOCAL_UDP_PORT and C_REMOTE_UDP_PORT respectively.

14 Advanced Configuration Options – Packet Classification Based on DSCP – Beta Version

The SCPS gateway has the ability to classify flows and provide different emission policies based on IPv4 Differentiated Services Code Points (DSCPs). For example you can assign particular bandwidth allocations to various DSCPs. This is implemented by extending the multiple router extension and the gw_route_cmdr application described in section 11

During the configuration of the gateway the compilation flag--*gateway_router_dscp* enables this functionality. For example, if you are running on the Linux operating system and decide to enable the DSCP based packet classification, you would configure and compile the code like the following.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes --tap=yes --gateway_router_dscp=yes --
gateway_ri_console=yes
cd ..
make clean;make
```

To compile the gw_route_cmdr application you need to change directories to utils/gw_route_cmdr. Before compiling the software, if you are running on a Linux operating you need to exit the Makefile and change the line from:

```
SYMTAB = -g -Wall
```

to

```
SYMTAB = -g –Wall –DLINUX
```

To compile the software type in the following command

```
Make
```

To add a route you must specify the key that will be used during the route lookup process. When routing based on DSCP the lookup procedure is different than described in section 11. The routing process is now based in interface (LAN or WAN) and DSCP. Neither network layer addresses nor transport layer port numbers are used.

Before adding routes, additional directions must be added to the gateway's *rfile* to indicate which interface is associated with the LAN side and which interface is associated with the WAN side.

The following directives must be added to the resource file.

AIF_LAN_OR_WAN
BIF_LAN_OR_WAN

A value of 1 is indicates this interface is the LAN side, while a value of 2 indicated this interface is the WAN side.
The gw_route_cmdr application has been augmented to include the following parameters

-L	indicates this route is associated with the LAN side
-W	indicates this route is associated with the WAN side
-t XX	indicates that the DSCP XX is associated with the route

For example let assume that we want to rate traffic with DSCP of 0x10 at 1 Mbps while rate traffic with DSCP of 0x41 at .512 Mbps. Therefore the following commands are needed.

```
/usr/sbin/gw_route_cmdr -A A -G localhost -W -t 0x10 -R 1000000 -T 0
/usr/sbin/gw_route_cmdr -A A -G localhost -W -t 0x41 -R 512000 –T 0
```

15 Advanced Configuration Options – Minimum Packet Emission Guarantee Plus Vegas – Beta Version

SCPS proxy can be statically configured to emit data at a predefined rate across the RF. This rate is typically equal to the bandwidth of the link itself. This option implements a slightly new packet emission strategy that combines the notion of pure rate control and the Vegas congestion control algorithm. In this strategy a minimum data rate a PEP will emitted is specified. If the offered load from can consume more than the minimum, the Vegas Congestion control Algorithm will be employed to determine how much more bandwidth a site may use. Vegas use the variation in round trip time to indicate queueing in the network.

During the configuration of the gateway the compilation flag*--gateway_router and –min_rate* enables this functionality. For example, if you are running on the Linux operating system and decide to enable the DSCP based packet classification, you would configure and compile the code like the following.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes --tap=yes --gateway_router =yes –min_rate=yes
cd ..
make clean;make
```

To compile the gw_route_cmdr application you need to change directories to utils/gw_route_cmdr. Before compiling the software, if you are running on a Linux operating you need to exit the Makefile and change the line from:
SYMTAB = -g -Wall
to
SYMTAB = -g –Wall –DLINUX

To compile the software type in the following command

```
Make
```

There are two methods of defining the minimum guaranteed route. This information can either be identified by adding new directives in the resource file or it van be identified by the gw_route_cmdr applications.

To specify the guaranteed minimum route via the gateway' resource file, the following directives may be used. The value is bits per second.

AIF_MIN_RATE
BIF_MIN_RATE

The other method is to add additional routes via gw_route_cmdr. The gw_route_cmdr application has been augmented to include the following parameters

-Z	indicates minimum bandwidth guarantee

16 Advanced Configuration Options – Lower CPU Utilization – Beta Version

To obtain the maximum throughput possible the SCPS Reference Implementation constantly polls the internal sockets to read data from the kernel as soon as it is available. This unfortunately means the SCPS reference implementation will consume all available CPU resources that it can. Therefore the SCPS RI will consume about 98 percent of the CPU, even when there is no traffic traversing the RI. This has caused some adverse interactions with other processes trying to consume the CPU's resources.

Two additional configuration options are available to alleviate this concern. There will be a performance hit because the interfaces will be serviced as quick as they possibly could. However the given the data rates involved and the speed of the processor, this may be an issue. The two compilation flags that can be included are –low_idle and –low_cpu. The configuration option –low_idle will keep the RI from consuming CPU resources during period of idle traffic. The configuration option –low_cpu can also be included to further reduce CPU resources when traffic is being processed. If the –low_cpu option is included then the –low_idle option must also be include.

For example, if you are running on the Linux operating system using the TAP interface and want to significantly reduce the CPU consumption, you would configure and compile the code like the following.

```
cd ../apps
./configure
cd ../source
./configure --gateway=yes --tap=yes –low_idle=yes –low_cpu=yes
cd ..
make clean;make
```

APPENDIX A Overview of the Gateway's Configuration

Before running the SCPS gateway, its resource file (rfile) must be configured to match the various characteristics of the networks it is attached to. The following table describes all of the rfile directives to configure the gateway.

	Resource File Directive	Description
Basic Transport Layer Configuration Directives		
	XIF_NAME	Name of the interface (i.e., sis1, eth1)
	XIF_BUF	Size in bytes of the transport layers send buffers
	XIF_RATE	Maximum rate associated with network
	XIF_CC	Congestion Control Technique: 0 – Pure Rate Control 1 – Vegas Congestion Control Algorithm 2 – VJ Congestion Control Algorithm
Advanced Transport Layer Configuration Directives		
	XIF_TS	Enable Transport Layer Timestamps 0 – Disable 1 – Enable (Default)
	XIF_SNACK	Enable Transport Layer SNACK 0 – Disable 1 – Enable (Default)
	XIF_MTU	Receive Maximum Transmission Unit in bytes (Default is MTU on actual interface.
	XIF_SMTU	Send Maximum Transmission Unit in bytes (Default is XIF_MTU directive value)
	XIF_TCPONLY	Disable SCPS TP Protocol 0 – Enable (Default) 1 – Disable
	XIF_ACK_DELAY	Delayed Acknowledgement Timer Value in msec (Default is 200 msecs)
	XIF_ACK_BEHAVE	TCP Acknowledgement Behavior 0 – Strictly delayed acks 1 – Acknowledge every segment 2 - Acknowledge every other segment (Default) Both 1 and 2 are still subject to the delayed acknowledgement timer
	XIF_NODELAY	Enable Transport Layer No Delay 0 – Disable (Default) 1 – Enable
	XIF_RBUF	Size in bytes of the transport layers receive buffers (Default is XIF_BUF)
	XIF_TP_COMPRESS	Enable Transport Layer Compression 0 – Disable(Default) 1 – Enable
	XIF_MSS_FF	Decrement MSS X bytes (default is 0)
Timer Configuration Directives		

	XIF_IRTO	Initial value of Retransmission Timer in microseconds (default is 6000000)
	XIF_MINRTO	Minimum value of Retransmission Timer in microseconds (default is 250000)
	XIF_MAXRTO	Maximum value of Retransmission Timer in microseconds (default is 64000000)
	XIF_SNACK_DELAY	Delay from Reception of SNACK and corresponding action in msec (default is 0)
Vegas Congestion Control Configuration Directives		
	XIF_VEGAS_ALPHA	Value of Vegas ALPHA parameter for congestion avoidance
	XIF_VEGAS_BETA	Value of Vegas BETA parameter for congestion avoidance
	XIF_VEGAS_GAMMA	Value of Vegas GAMMA parameter for slow start
	XIF_VEGAS_SS	Vegas Slow Start Behavior 0 – Standard Vegas SS Behavior (Default) 1 – Standard VJ SS Behavior
Option Protocol Configuration Directives		
	XIF_NL	Network Layer Protocol 1 – IP Network Layer (Default) 2 – NP Network Layer
	XIF_SCPS_SECURITY	Control use of SCPS SP protocol 0 – Don’t use SP (Default) 1 – Use SP only if entry in SADB exists 2 – Always use SP
Multiple Path Forward Function Configuration Directives		
	XIF_MPF	Enable Multiple Path Forwarding 0 – Disable (Default) 1 - Enable
	XIF_MPF_SRC	Source Address for MPF encapsulation (will be used many times)
	XIF_MPF_DST	Destination Address for MPF encapsulation (will be used many times)
Additional Protocol Layering Techniques Configuration Directives		
	XIF_LAYERING	Enable UDP encapsulation 0 – Disable (Default) 1 - Enable
	XIF_LOCAL_IP	Source address if UDP encapsulation enabled
	XIF_REMOTE_IP	Destination address if UDP encapsulation enabled
	C_LOCAL_UDP_PORT	Source UDP port if UDP encapsulation enabled
	C_REMOTE_UDP_PORT	Destination UDP port if UDP encapsulation enabled
	XIF_NEXT_HOP	Next hop packets should be forward to
	XIF_DIV_ADDR	
	XIF_DIV_PORT	
	XIF_OVERHEAD	Additional overhead in bytes via external encapsulation which must be accounted (Default is 0)
Operating Systems Support Configuration Directives.		
	XIF_DIVPORT	Divert port number for this interface
	C_DIVPORT	Divert port number for this interface

	C_DIVERT_START_RULE	Start rule number for divert rules
	C_DIVERT_INSERT_RULE	Last rule number for divert rules
	C_TUN_NAME	TUN interface name for this interface
	XIF_TUN_NAME	TUN interface name for this interface
	XIF_TAP_NAME	TAP interface name for this interface
	C_OTHER_PROTO_QLEN	Length of queue for non-TCP based traffic –TAP access only (Default is 5)
	C_OTHER_PROTO_XRATE_DROP	Policy for non-TCP based traffic in rate is not available – TAP access only. 0 – Do not drop (Default) 1 – Drop if rate not available
	C_TAP_REMOTE_ACCESS	Ability to access the gateway via either the LAN or WAN interface
Debugging Directives		
	C_CLUST_THRESH	Stops accepting new connections if the number of allocated cluster is greater that X where X is the number of clusters.
	C_CLUST_FILENAME	Activates 'C_CLUST_THRESH' if the filename exists
	C_PKT_IO_FILENAME	Displays a packet trace through the gateway if the filename exists.
	C_NETSTAT_INTERVAL	Interval in seconds to display cluster and socket usage 0 – Do not display usage (Default) X – Display every X seconds.

APPENDIX B - Interaction of TCP and Satellite Environments

This section provides both a description of Transmission Control Protocol (TCP) [RFC 793] and insight into why communicating over satellite and other wireless technologies causes performance degradation. .

The Internet has seen exponential growth over the last decade. TCP, which provides a reliable in-order mechanism for transferring data between systems without duplications, has become ubiquitous throughout the Internet. In addition to providing a reliable stream TCP also contains congestion control mechanisms to help prevent the network meltdown that occurred in the late 1980s. The majority of Internet applications (i.e., FTP, HTTP, E-mail, etc) rely on the underlying transport mechanism provided by TCP. There are two mechanisms to control the transmission of data from a sender; flow control and congestion control. The amount of data the TCP sender may emit at a point in time is essentially the minimum of these two techniques. These functions will be described and how communications over satellite networks effect performance.

Flow control is used to ensure that the sender does not overwhelm the receiver's buffers. In actuality, both the TCP sender and receiver have buffers. The sender uses the buffers as a retransmission buffer (i.e., to store data for possible retransmission) while the receiver uses these buffers as an out-of-sequence queue (i.e., to save packets received by the sender that are not in sequence to avoid the sender from possibly having to retransmit them.) Therefore if the buffer sizes are different only the minimum buffer size of the two is used for flow control. When looking at flow control within TCP, it is quite useful to examine the Bandwidth Delay Product (BDP). The BDP provides a relationship between the bandwidth of a network and the round trip delay though the network to establish the minimum TCP window size required under ideal conditions (i.e., no segment loss or network queuing causing the round trip time to increase). Therefore if the bandwidth of the network (i.e., the slowest link in the network) is 1 Mbps and the round trip time through the entire network is 600 milliseconds (which is typically for satellites operating in a geo-synchronous orbit) the minimum TCP buffer size needs to be 75,000 bytes. While the default buffer sizes (which range from 8K to 32K bytes on most common operating systems) work for terrestrial networks, when operating over satellite and other large delay networks the defaults may be a limiting factor in performance. The BDP equation can be turned around. If the buffer size is 16K and the round trip time is 600 milliseconds then the theoretical maximum throughput possible is 218,480 bps. Most modern operating systems allow the default buffer sizes to be changed on a system wide basis. However this is rarely performed since changing the buffers on a system wide basis will cause the system memory pool to be quickly exhausted resulting in an inability to establish new connections. Unfortunately most network applications are not able to dynamically tune their buffer, although there is research in the area of buffer tuning [SMM98].

Congestion control, on the other hand, tries to prevent the sender from overrunning network resources (i.e., intermediate system's interface queues) thus resulting in packet loss and subsequent retransmissions. TCP implementations are required to implement the Van Jacobson's Congestion Control (VJCC) [JAC88] [RFC 2581] algorithm, which has been refined over time. The majority of systems implement the variant known as Reno. VJCC is essentially a reactive protocol – it tries to drive the network into congestion by increasing the packet transmission rate to determine when congestion occurs. Once TCP drives the network into congestion, which is determined by packet loss, it will essentially cut its transmission rate in half and slowly try to drive the network back into congestion. TCP assumes all loss is due to congestion, where in fact the loss could be due to congestion, corruption, or link outage. In a terrestrial (i.e., wired environment) the assumption that all loss is due congestion seem reasonable. However in a satellite environment bit error rates are orders of magnitude higher than their wired counterpart.

VJCC has two phases, *slow start* and *congestion avoidance*. The slow start phase occurs at the beginning of a connection or after a Retransmission Time Out (RTO) and its purpose is to quickly increase the transmission rate to determine when congestion occurs. The congestion avoidance phase occurs after loss and its purpose is to slowly increase the transmission rate to determine when congestion occurs. During slow start the sender will send one packet to the destination and wait for a positive acknowledgement. The TCP sender then emits two packets and waits for an acknowledgement. Essentially during slow start, for each acknowledgement the sender receives, it may emit that number of packets being acknowledged plus one. During slow start, the number of packets in flight is also known as the congestion window (cwnd). This exponential growth increases until one of three conditions happen: 1) flow control via the receiver's buffers becomes a limiting factor; 2) slow start threshold (ssthresh)[4] is reached; or 3) a loss occurs. Once a loss occurs, which VJCC interprets as network congestion, cwnd is cut in half (in reality the new value of cwnd is the minimum of the old value of cwnd and the receivers advertised window divided by two.) This results in the transmission rate being cut in half. During congestion avoidance the sender increases cwnd by a MSS (i.e., a full sized packet) each time it sends a cwnds worth of data. For example, if the current value of cwnd is 8192, after it emits 8192 bytes, cwnd becomes 8192 + 1460 or 9652 bytes. This linear increase in cwnd continues until another loss occurs.

When operating over a satellite network the interaction of VJCC with the satellite characteristics of both the increase in packet corruption and round trip delay adversely effect the performance of TCP. The increase in packet corruption impacts the performance in two ways. First, with any loss event, TCP will assume that the network in congested and will essentially cut its transmission rate in half. If the loss event is due to corruption, rather than congestion, the halving of cwnd is unnecessary. Second, it will

[4] TCP implementations typically have two choices in setting ssthresh. Typically the sender has no notion of network conditions and ssthresh is set to the sender's maximum value of send buffer. TCP senders may also set ssthresh to a cached value, which may not reflect current network conditions. If ssthresh is set too low, slow start will terminate prematurely and the congestion window will be grown linearly, a lower throughput will result. If this value is set too high slow start will drive the network into congestion loss at the bottleneck router resulting in multiple packet loss; requiring packet retransmission, and resulting is a lower throughput.

prematurely terminate slow start. Thus it will take much longer to obtain the theoretical maximum instantaneous throughput. For example let's assume buffers are set to 75,000 bytes – which can accommodate a maximum throughput of 1.0 Mbps. Let's say that when the cwnd is at 75,000 bytes and a loss event occurs so the resulting cwnd becomes 37,500. The sending window may only send 37,500 bytes in 600 milliseconds. Once it admits and receives an acknowledgement for all 37,500 bytes it may send 38,960 (assuming a MSS of 1460 bytes). Assuming it can send and receive its entire cwnd in 600 msec, it will take over 25 congestion window increases (i.e., (75000-37500)/1460) for the TCP sender to emit data at the maximum throughput rate of 1.0 Mbps.

The increased in round trip delay also affects the performance in TCP during the essentially two ways. First, it will take significantly longer during slow start for the sender to obtain the maximum theoretical rate. Let's compare how long it takes to reach the maximum transfer rate of 1.0 Mbps in both a terrestrial and satellite environments. Again let's assume the buffer sizes are set to 75,000 bytes and the round trip delay in 600 msec. For the terrestrial environment, assuming a round trip time of 50 msec only a buffer size of 6,250 bytes is required, it will take slow start about 4 round trips to reach the maximum throughput. For the satellite environment, assuming a round trip time of 600 msec and a buffer size of 75,000 bytes, it will now take slow start about 10 round trips to reach maximum throughput.[5] Therefore in the terrestrial environment is taking about 200 msec to reach the maximum throughput versus 6,000 msec for the satellite environment. Let's also see how long it takes to recover from an error (i.e., cutting the cwnd and the effective transfer rate) in half. In the above example, where a loss occurs after the maximum throughput is obtained, it takes 25 increases in the congestion window to turn to the precut value. This takes over 15 seconds. In the terrestrial example in create the cwnd from 3,225 bytes back to 6,250 bytes, it takes 4 congestion window increases. This takes only 200 msec for the effective rate to be increased back to 1.0 Mbps.

[5] These calculations assume a delayed acknowledgement strategy, typical in most TCP implementations, results in an exponential increase (1.5) in the value of the cwnd during slow start.

APPENDIX C - Network Analysis and Test Tools

Network and protocol analysis can be performed by the following utilities: tcpdump, tcptrace, xplot, and ttt. Tcpdump is a network utility developed to collect packets seen by a machine's network interface. This packet trace can be used to analyze the behavior of a protocol on a network. All data gathered in this study has been gathered via tcpdump. The packet traces are then processed via a program called tcptrace. This program outputs various graphs (including time sequence graphs, throughput graphs, and round trip time graphs.) These graphs are then displayed in an X-windows environment, by a program called xplot. Xplot allows users to "zoom-in" or "zoom-out" to examine the various aspects of the TCP protocol. Tele Traffic Tapper (ttt) is capability of real-time monitoring of network traffic. Ttt can determine how individual flows consume the underlying network capacity. The ttt graphs the amount of bandwidth consumed by each of the underlying streams of data. The graphs used in this paper are plots of tcptrace's graphs and results of ttt.

Tcpdump

Tcpdump is a network utility developed to collect packets seen by a machine's network interface. This packet trace can be used to analyze the behavior of a protocol on a network. For example tcpdump can be used to gather TCP packets from the ttcp utility. This packet trace can be processed to obtain throughput graphs and any other types of information. Tcpdump was developed and maintained by Van Jacobson, Craig Leres, and Steven McCanne, all of Lawrence Berkeley Laboratory, to be portable about various Unix operating systems. The latest version of tcpdump can be found at ftp://ftp.ee.lbl.gov/tcpdump.tar.Z.

Tcptrace

Tcptrace is a TCP dump file analysis tool written by Shawn Ostermann at Ohio University. The input to tcptrace is a TCP packet trace gathered from tcpdump. The output contains various statistics about the TCP connections. Tcptrace also produces the following output: 1) time sequence graphs, 2) throughput graphs, and 3) round trip time graphs. These graphs can be displayed by Tim Shepard's xplot program. The latest version of tcptrace can be found at http://jarok.cs.ohiou.edu/software/tcptrace.

Ttcp

Ttcp is a network utility used to characterize network performance. Originally developed at the US Army Ballistics Research Lab (BRL), ttcp's the source code has been placed in the public domain. Since then it has been ported to most Unix environment including Microsoft's Windows 95 environment. It allows the configuration of number of packet transmitted, the packet size, port numbers, transport buffer sizes (assuming operating system support) and other parameters. Ttcp reports on the throughput and delay of the transfer test. Ttcp can be found at. ftp://ftp.arl.mil/pub/ttcp/.

TTT: Tele Traffic Tapper

ttt is yet another descendant of tcpdump but it is capable of real-time, graphical, and remote traffic-monitoring. ttt won't replace tcpdump, rather, it helps you find out what to look into with tcpdump. ttt monitors the network and automatically picks up the main contributors of the traffic within the time window. The graphs are updated every second by default. Ttt can be found at http://www.csl.sony.co.jp/person/kjc/kjc/software.html#ttt

Xplot

Xplot is an X-windows graphing package written by Tim Shepard at MIT as part of his master's thesis. This program reads output from tcptrace and displays the data graphically allowing users to "zoom-in" or "zoom-out" to examine the various aspects of the TCP protocol. The latest version of xplot can be found at ftp://mercury.lcs.mit.edu/pub/shep.

APPENDIX D – Testbed Setup Thoughts

The following was sent to a recipient of the SCPS RI over 2 years ago, I thought it might still be useful and thus including it.

1. Simple test setup would include 7 machines. A source and destination machines, two gateway machines, a satellite emulator, a network analyzer, and a test harness

2. With the lab setup, it is very useful to have a control LAN as well as a test LAN to configure the topology of the network. In this way any traffic generated to control the machines will not interact with the data you are collecting. In other words each machine have an Ethernet interface which is connected to a common LAN segment, as well as interfaces for interconnecting the machines to what ever topology is required.

3. Most of the time it is easier to connect the machines via one 'large' programmable switch versus dumb Ethernet hubs. This allows you to make topology changes to the machines a little easier. Using VLANs to connect the machines would do this. Make sure your switch has the ability to mirror data to allow for traffic monitoring.

4. If using tcpdump or other utilities to capture traffic while the test is in progress, this machine should not be active in any other aspects of the test. You don't want to have the system resources required to capture and store the information on disk effect performance.

5. Invest the time required to learn and use a test scripting language to fully automate the test. Fat fingers and memory lapses will cause you more problems that could be imagine. Expect is the tool that we use.

6. Take some time to figure out what a correct answer should be to help validate your results. For example let's say you are doing a test in which four independent streams will share a common satellite resource (1-Megabit capacity) where each stream will transfer a 5 Megabytes file. If we assume that each stream will share the bandwidth equally than the file transfer should take (best case) then

   (5 Megabytes * 8 (bits/byte) * 1500/1448 (protocol overhead))/ 250000 (bps) or best cast it will take 165 seconds to transfer the file.

   If the results from you experiment say it took on average 155 seconds for each of the file transfer to complete, then something is rotten in the state of Denmark.

7. When looking at throughput and delay results from an application, always make sure it is from the perspective of the receiver of the data. For example if you are using FTP, do a get, not a put.: Sockets, one of the most common networking APIs, will serve as an excellent example. From an application perspective, once an application

establishes a connection, sends data and issues a close, the application has completed the transfer and will report the results to the use. Just because data is written to a socket and the application closes the socket it does not in any way shape form or fashion mean the remote systems has reliable received the data. It only means that the local socket has consumed the data from the application. Therefore an application can close the connection and data not received by the applications peer. The receiving side can only report on the statistics once it has received and consumed the data.

8. For throughput testing a simple FTP or ttcp may suffice, you way want to test more than one flow of traffic through a system though. There may be a limit on the number of active connection a system may handle. Along with performance testing, you may want to get into stress testing. For example, if on average I initiate 10 seconds per second for 1 hour how does the system perform. A good tool for this is http://www.web-polygraph.org/

9. When tcpdumping traffic for protocol/traffic analysis you want to tap the traffic at the source not the sink side; for example at the near side gateway.

[SCPS FP]	Space Communications Protocol Specification (SCPS)—File Protocol (SCPS-FP). Recommendation for Space Data System Standards, CCSDS 717.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1999
[SCPS TP]	Space Communications Protocol Specification (SCPS)—Transport Protocol (SCPS-TP). Recommendation for Space Data System Standards, CCSDS 714.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1999
[SCPS SP]	Space Communications Protocol Specification (SCPS)—Security Protocol (SCPS-SP). Recommendation for Space Data System Standards, CCSDS 713.5-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1999
[SCPS NP]	Space Communications Protocol Specification (SCPS)—Network Protocol (SCPS-NP). Recommendation for Space Data System Standards, CCSDS 713.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1999
[RFC 793]	Postel, J. “Transmission Control Protocol”, RFC 793, September 1981
[SMM98]	J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP Buffer Tuning", Computer Communications Review, a publication of ACM SIGCOMM, volume 28, number 4, October 1998
[JAC88]	V. Jacobson, “Congestion Avoidance and Control,” *Proceedings of ACM SIGCOMM ’88*, pp. 314-329, Stanford, CA, August 1988
[RFC 2581]	Allman, V. Paxson, and W. Stevens, “TCP Congestion Control”, RFC 2581, April 1999.