

sctp.README - README file for NS-2 SCTP module release 3.3

Armando L. Caro Jr. <acarо@cis,udel,edu>

Please submit bug reports to pel@cis,udel,edu

@(#) \$Header: /cvsroot/nsnam/ns-2/sctp/sctp.README,v 1.1 2003/08/21 18:29:14 haldar
Exp \$

INSTALL:

1. For first installs only:

a. download ns-allinone-2.1b8 from:
http://www.isi.edu/nsnam/dist/ns-allinone-2.1b8.tar.gz

b. untar the file by doing:
% gtar zxvf ns-allinone-2.1b8.tar.gz

2. Download the appropriate patch (<http://pel.cis.udel.edu>):

ns-allinone-2.1b8.sctp-rel3.3.patch.orig (for first installs)
or
ns-allinone-2.1b8.sctp-rel3.3.patch.relx.x (x.x is ver of prev install)

3. At the parent directory of the untar'd source, do:

% patch -p0 < ns-allinone-2.1b8.sctp-rel3.3.patch.orig
or
% patch -p0 < ns-allinone-2.1b8.sctp-rel3.3.patch.relx.x

4. Run the ns-2 install script:

% cd ns-allinone-2.1b8
% ./install

5. For first installs only: add ns-allinone-2.1b8/bin to your path

6. (optional) Download latest test-output-sctp from <http://pel.cis.udel.edu>

MODULE INFO:

The SCTP module for NS-2 currently supports the features in the following sections of **RFC4460** (with draft-ietf-tsvwg-sctpimpguide-04.txt):

- 5.1 Normal Establishment of an Association (rudimentary handshake)
- 5.4 Path Verification**
- 6.1 Transmission of DATA Chunks
- 6.2 Acknowledgement of Reception of DATA Chunks
- 6.3 Management Retransmission Timer

- 6.4 Multi-homed SCTP Endpoints
- 6.5 Stream Identifier and Stream Sequence Number
- 6.6 Ordered and Unordered Delivery
- 6.7 Report Gaps in Received DATA TSNs
- 7.2 SCTP Slow-Start and Congestion Avoidance
- 8.1 Endpoint Failure Detection
- 8.2 Path Failure Detection
- 8.3 Path Heartbeat (without upper layer control)

Also, the unreliable data mode extension (U-SCTP) is supported as of draft-ietf-tsvwg-usctp-01.txt.

The SCTP module has NS-2 upper layer API support (but many features are still lacking). It supports all legacy NS-2 applications, but they obviously aren't able to exploit all SCTP's features. However, SCTP-aware application modules can be written to do so.

NS-2 applications wishing to become SCTP-aware can use the sendmsg API as follows (see sctp_appl.cc and sctp_appl.h for an example). Note: this feature hasn't been tested extensively, so please report bugs!

1. create and fill an AppData_S object (see sctp.h)

- usNumStreams

The number of outgoing streams to setup during negotiation. Although this field is passed with every sendmsg call, it is only used when associating the connection.

- usNumUnreliable

The number of outgoing streams which are unreliable. The sender simply sets the lowest outgoing stream to unreliable; the remaining ones are reliable. This field is also only used when associating the connection.

- usStreamId

The stream ID of this message.

- usReliability

The reliability level (k-rtx value) of this message.

- eUnordered

The unordered boolean flag for this message.

- uiNumBytes

The number of bytes in this message.

2. pass this object as the second parameter in SCTP's sendmsg:

```
sctpAgent->sendmsg(numBytes, (char *)appData);
```

The SCTP module uses several variables which can be bound from within tcl. The corresponding variables to be used in tcl are (with explanations):

debugMask_

The 32-bit mask to turn on/off debugging for particular functions. See sctpDebug.h for the mappings of the bitmask. To set all bits, a -1 may be used. The default is 0, which means all debugging is off.

Note: ns must be compiled with -DDEBUG for this option to work. Also, if debug_ (the standard debug flag) is set to 1, then all the bits in debugMask_ are set.

debugFileIndex_

Each SCTP peer can independently output debugging info to a separate file. For example, the data sender can log debugging output to one file, while the receiver does it to another file. The file index specifies the file to be used for the particular peer. ie, If debugFileIndex_ is set to 0, the file used will be named 'debug.SctpAgent.0'. If -1 is used, the debug output is sent to stderr. Also, two peers should NOT send debug output to the same file. The default is -1.

Note: ns must be compiled with -DDEBUG for this option to work.

associationMaxRetrans_

Sets the parameter Association.Max.Retrans. The default is 10 attempts.

pathMaxRetrans_

Sets the parameter Path.Max.Retrans. The default is 5 attempts.

maxInitRetransmits_

Sets the parameter Max.Init.Retransmits. The default is 8 attempts.

heartbeatInterval_

Sets the parameters HB.interval. Setting this parameter to 0 turns off HEARTBEATING. The default is 30 seconds.

oneHeartbeatTimer_

Toggles the use of one HEARTBEAT timer for all destinations or a timer per destination. If 0 is used, there is a separate HEARTBEAT timer for each destination. If 1 is used, there is a single HEARTBEAT timer for all destinations (similar to the reference implementation). **For all CMT and CMT-PF experiments the oneHeartbeatTimer should be set to 0.** The default is 1.

mtu_

Sets the MTU for the path. Consequently, it sets the maximum packet size as well, since our sender tries to fill packets up to MTU. The actual data is MTU - SCTP/IPv4 header (32 bytes). The default is 1500.

initialRwnd_

Sets the initial rwnd value which will be advertised to the peer. The default is 65536.

initialSsthresh_

Sets the initial ssthresh value (in bytes). The default is 65536.

initialCwnd_

Cwnd will be initialized to (initialCwnd_ * MTU). The default is 2.

numOutStreams_

Sets the number of outgoing streams from the sender. Although the sender does transmit this value as the RFC specifies, the receiver simply accepts without negotiation. The default is 1.

numUnrelStreams_

Sets the number of outgoing streams which are unreliable. The sender simply sets the lowest outgoing stream to unreliable; the remaining ones are reliable. The default is 0.

reliability_

Sets the default reliability level on all the unreliable streams. An SCTP-aware ns-2 application can control the reliability level on a per chunk basis by using the upper layer API. On the other hand, since legacy ns-2 applications do not use this API, they do not have individual control. The default is 0, which means zero retransmissions on the unreliable streams.

unordered_

Sets the default unordered flag for all chunks on all streams. An SCTP-aware ns-2 application can control the unordered flag on a per chunk basis by using the upper layer API. On the other hand, since legacy ns-2 applications do not use this API, they do not have individual control. The default is 0, which means all chunks are ordered.

ipHeaderSize_

Sets the IP header size (in bytes) of each packet. The default is 20 (IPv4 without any options).

dataChunkSize_

Sets the size of each data chunk. Between this and the MTU, the

number of chunks per packet can be controlled. We do not support variable length chunks yet. The default is 512.

useDelayedSacks_

Sets whether delayed sacks will be used or not. 0 means generate a sack for every incoming packet. 1 means do the normal delayed ack algorithm. The default is 1.

useMaxBurst_

Sets whether the Max.Burst restriction will be applied or not. 0 means it is off. 1 means it is on and follows the SCTP implementor's guide. The default is 1, which means the option is on.

rtxToAlt_

RFC2960 says that retransmissions should go to an alternate destination when available. An experimental behavior is to retransmit to the same destination, unless it is inactive. The default is 1, which means that retransmission go to an alternate destination.

cwnd_

Used to trace the CWNDs of all paths. It can be used with both the sampling ("print") and non-sampling ("trace") method of tracing.

rto_

Used to trace the RTOs of all paths. It can be used with both the sampling ("print") and non-sampling ("trace") method of tracing.

errorCount_

Used to trace the error counters of all paths. It can be used with both the sampling ("print") and non-sampling ("trace") method of tracing.

trace_all_

Possible values are 0 and 1. This is used to determine whether all the trace variables should be printed out any time a single traced variable is triggered. It can be used with both the sampling ("print") and non-sampling ("trace") method of tracing. Default is 0, which means the option is off.

The SCTP module provides certain "commands" that can be used within tcl scripts. The commands are (with explanations):

print

Provides the non-sampling method of tracing. Takes 1 argument: one of the trace variables presented above.

set-multihome-core

Set the core node for multihomed endpoints. Takes 1 argument of type node. Mandatory for multihomed endpoints and must not be set more than once per endpoint. (see "interesting features" 6. below)

multihome-add-interface

Add an interface to a multihomed endpoint. Takes 2 arguments of type node. Argument 1 is the core node of the multihomed endpoint. Argument 2 is the interface node to be added. Mandatory for multihomed endpoints. All interfaces must be added after set-multihome-core is called and before multihome-attach-agent is called. (see "interesting features" 6. below)

multihome-attach-agent

Attach an SCTP agent to a multihomed endpoint. Takes 2 arguments. Argument 1 is the core node. Argument 2 is the SCTP agent. Mandatory for multihomed endpoints. (see "interesting features" 6. below)

set-primary-destination

Set the interface node of the peer endpoint as the primary destination. Takes 1 argument of type node. Optional and may be set more than once per endpoint. If not used, a primary destination is chosen automatically. (see "interesting features" 6. below)

force-source

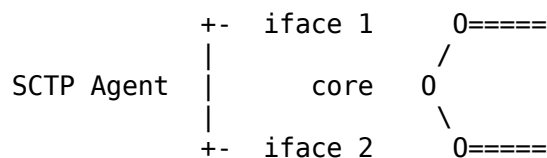
Set the interface node that packets will be sent from. Takes 1 argument of type node. Optional and may be set more than once per endpoint. If not used, routing will automatically choose the source on a per packet basis. (see "interesting features" 6. below)

The SCTP module also comes with some "interesting features" :-).

1. Since legacy ns-2 applications are unaware of SCTP and the module's upper layer API, they lack fine level control of stream unreliability levels and data chunks' unordered flags. Hence, only the tcl default variables can be used for those applications.
2. There is a 4-way handshake which is used to initialize an association, but it is done as simply as possible. Hence, there is no actual exchange of tags, etc as detailed in RFC2960. Also, the INIT and COOKIE-ECHO do have retransmission timers associated with them in case of loss, but these chunks are not used to update the RTT. RTT estimation calculations begin with the first DATA chunk.
3. While ns-2 starts numbering packets at 0, the SCTP module starts numbering DATA chunk TSNs at 1 and assigns undefined TSN values to non-DATA chunks (ie, INIT, SACK, HEARTBEAT, etc). In addition, the 4 packets exchanged during the association initialization are not included in the traces and do not show up in the graphs, but they are still counted in the packet enumeration. This information is important when doing things like specifying a drop list for the ErrorModel object. For example, if a List is specified for the ErrorModel object, packet 2 actually refers to the first SCTP packet

with DATA chunk(s).

4. RTT measurements are done once per packet containing DATA chunk(s) with no retransmissions. This is more frequent than what RFC2960 specifies (" RTT measurements SHOULD be made no more than once per round-trip"). This may change to follow the RFC in a later release.
5. A proper shutdown is not done yet. The association is abruptly terminated when the simulated connection ends.
6. Due to ns-2's architecture, a node cannot actually be multihomed. We have gotten around this limitation in the following way. Each multihomed node is actually made up of more than one node. There is a "core node" and multiple "interface nodes" to simulate the interfaces. The SCTP agent resides on all the nodes, but traffic only goes to and from the interface nodes. The core node is used for routing and is connected to each interface node via a uni-directional link towards the interface node. Traffic does not traverse this link. Instead, these links are used to dynamically determine which outgoing interface node to use for sending to a particular destination. NAM can help better visualize the architecture, but the diagram below gives some idea.



New SCTP module introduces a new type of SCTP agent (Agent/SCTP/CMT) which introduces two experimental extensions to SCTP: Concurrent Multipath Transfer (CMT) and Concurrent Multipath Transfer with Potentially Failed (CMT-PF).

CMT uses the SCTP's multihoming feature to distribute data across multiple end-to-end paths in a multihomed SCTP association to increase an application's throughput. CMT is the concurrent transfer of new data from a source to a destination host via two or more end-to-end paths.

One of the negative impacts of CMT is receiver buffer (rbuf) blocking that occurs during congestion. Rbuf blocking occurs where TPDU losses halt the sender because the SCTP receiver's buffer is filled with out-of-order data. To improve CMT's performance during failure, a new state called "Potentially Failed" is introduced for each destination. When a timeout occurs for a destination, the destination moves to PF state and heartbeats (HB) are sent exponentially to verify that the destination is active. When a HB is acked the destinations state goes back to active state. No new data and retransmissions are sent to a destination with PF state.

The SCTP module uses several variables for CMT and CMT-PF which can be bound from within tcl. The corresponding variables to be used in tcl are (with explanations):

CMT variables:

useCmtReordering_

Variable to turn on/off CMT's Reordering algorithm which eliminates sending

the unnecessary fast retransmissions caused due to reordering. Sending unnecessary fast retransmissions have 2 negative consequences: 1) reduced cwnd for the destination on which the retransmitted data was outstanding. 2) aggressive cwnd growth for the destination on which the retransmissions are sent. The details of the algorithm can be found at [1]. The default value is 1 (ON).

useCmtCwnd_

Variable to turn on/off CMT's Cwnd Update for CMT (CUC) algorithm. CUC algorithm eliminates reduced cwnd growth due to fewer cwnd updates at the sender due to reordering. Reordering introduces many sacks sent containing gaps reports but not new cum acks. Cwnd growth occurs only with a new cum ack (but only for the data newly acked in the most recent sack). Data previously acked through gap reports does not contribute to cwnd growth even though data may have reached the receiver "in order per destination". With CUC algorithm, cwnd is updated even in the absence of new cum acks. The details of the algorithm can be found at [1]. The default value is 1 (ON).

useCmtDelAck_

Variable to turn on/off CMT's Delayed Ack for CMT (DAC) algorithm. DAC algorithm eliminates the increased ack traffic due to fewer delayed acks. Acks are delayed only as long as the receiver receives data in order. Due to reordering introduced with CMT, an SCTP receiver does not frequently delay acks. With DAC algorithm, a CMT receiver always delays acks irrespective of whether or not data is received in order. The details of the algorithm can be found at [1]. The default value is 1 (ON).

eCmtRtxPolicy_

Variable to define retransmission policy for CMT sender. Currently there are 5 retransmission policies for CMT. Three of the retransmission policies are experimental policies and other two policies are suggested ones. The retransmission policies for CMT are as follows:

RTX_ASAP (experimental) - A retransmission of a data chunk is sent to any destination for which the sender has cwnd space available at the time of retransmission. If multiple destinations have available cwnd space, one is chosen randomly. The value to set RTX_ASAP policy is 0.

RTX_SAME (experimental) - All retransmissions of the data chunk are sent to the same destination until the destination is deemed inactive due to failure. The value to set RTX_SAME policy is 1.

RTX_SSTHRESH (suggested) - A retransmission is sent to the destination for which the sender has the largest ssthresh value. A tie is broken by random selection. The value to set RTX_SSTHRESH policy is 2.

RTX_LOSSRATE (experimental) - A retransmission is sent to the destination with the lowest loss rate path. A tie is broken by random selection. The value to set RTX_LOSSRATE policy is 3.

RTX_CWND (suggested) - A retransmission is sent to the destination for which the sender has the largest cwnd. A tie is broken by random selection. This policy is default policy for CMT. The value to set RTX_CWND policy is

4.

CMT-PF variables:

useCmtPF_

This variable is used to turn on/off CMT-PF extension. The default value is 0 (OFF).

cmtPFCwnd

The variable to set cwnd in MTUs after HB-ACK is received (PF->ACTIVE) at the sender. Two possible values for this variable are 1 or 2. The default value is 1.

```
-----
# EXAMPLE SCRIPT 1
#
# Simply drops one packet and later fast rtx it.

# this needs to be set for tracing SCTP packets
Trace set show_sctphdr_ 1

set ns [new Simulator]
set nf [open sctp.nam w]
$ns namtrace-all $nf

set allchan [open all.tr w]
$ns trace-all $allchan

proc finish {} {
    global ns nf allchan

    set PERL "/usr/bin/perl"
    set USERHOME [exec env | grep "^HOME" | sed /^HOME=/s/^HOME=//]
    set NSHOME "$USERHOME/proj/ns-allinone-2.1b8.sctp"
    set XGRAPH "$NSHOME/bin/xgraph"
    set SETFID "$NSHOME/ns-2.1b8/bin/set_flow_id"
    set RAW2XG_SCTP "$NSHOME/ns-2.1b8/bin/raw2xg-sctp"

    $ns flush-trace
    close $nf
    close $allchan

    exec $PERL $SETFID -s all.tr | \
        $PERL $RAW2XG_SCTP -A -q > temp.rands
    exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands &

    exec nam sctp.nam &

    exit 0
}

set false 0
set true 1

set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 .5Mb 200ms DropTail
```

```

$ns duplex-link-op $n0 $n1 orient right
#$ns queue-limit $n0 $n1 93000

set err [new ErrorModel/List]
$serr droplist {15}
$ns lossmodel $serr $n0 $n1

# NOTE: The debug files (in this example, they would be debug.SctpAgent.0
#       and debug.SctpAgent.1) contain a lot of useful info. They can be
#       used to trace every packet sent/rcvd/processed.
#
set sctp0 [new Agent/SCTP]
$ns attach-agent $n0 $sctp0
$sctp0 set fid_ 0
$sctp0 set debugMask_ 0x00303000 # u can use -1 to turn on everything
$sctp0 set debugFileIndex_ 0
$sctp0 set mtu_ 1500
$sctp0 set dataChunkSize_ 1468
$sctp0 set numOutStreams_ 1
$sctp0 set initialCwndMultiplier_ 2
$sctp0 set useMaxBurst_ $true

set trace_ch [open trace.sctp w]
$sctp0 set trace_all_ 0 # do not trace all variables on one line
$sctp0 trace cwnd_
$sctp0 attach $trace_ch

set sctp1 [new Agent/SCTP]
$ns attach-agent $n1 $sctp1
$sctp1 set debugMask_ -1
$sctp1 set debugFileIndex_ 1
$sctp1 set mtu_ 1500
$sctp1 set initialRwnd_ 131072
$sctp1 set useDelayedSacks_ $true

$ns color 0 Red
$ns color 1 Blue

$ns connect $sctp0 $sctp1

set ftp0 [new Application/FTP]
$ftp0 attach-agent $sctp0

$ns at 0.5 "$ftp0 start"
$ns at 5.0 "finish"

$ns run

-----
# EXAMPLE SCRIPT 2
#
# Demonstrates multihoming. Two endpoints with 2 interfaces with direct
# connections between each pair. There is a single HEARTBEAT timer for all
# the destinations (similar to the reference implementation). In the
# middle of the association, a change primary is done.
#
#       host0_if0      0=====0      host1_if0

```

```

#          host0_core  0  /          \  0  host1_core
#          host0_if1   \  0=====0  /          host1_if1

```

```
Trace set show_sctphdr_ 1
```

```
set ns [new Simulator]
set nf [open sctp.nam w]
$ns namtrace-all $nf
```

```
set allchan [open all.tr w]
$ns trace-all $allchan
```

```
proc finish {} {
    global ns nf allchan

    set PERL "/usr/bin/perl"
    set USERHOME [exec env | grep "^HOME" | sed /^HOME=/s/^HOME=//]
    set NSHOME "$USERHOME/proj/ns-allinone-2.1b8.sctp"
    set XGRAPH "$NSHOME/bin/xgraph"
    set SETFID "$NSHOME/ns-2.1b8/bin/set_flow_id"
    set RAW2XG_SCTP "$NSHOME/ns-2.1b8/bin/raw2xg-sctp"
    set GETRC "$NSHOME/ns-2.1b8/bin/getrc"

    $ns flush-trace
    close $nf
    close $allchan

    # Graph the host0_if0 <-> host1_if0
    exec $PERL $GETRC -s 1 -d 4 all.tr > all.tr.if0
    exec $PERL $GETRC -s 4 -d 1 all.tr >> all.tr.if0
    exec $PERL $SETFID -s all.tr.if0 | \
        $PERL $RAW2XG_SCTP -A -q -t if0 > temp.rands.if0
    exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands.if0 &

    # Graph the host0_if1 <-> host1_if1
    exec $PERL $GETRC -s 2 -d 5 all.tr > all.tr.if1
    exec $PERL $GETRC -s 5 -d 2 all.tr >> all.tr.if1
    exec $PERL $SETFID -s all.tr.if1 | \
        $PERL $RAW2XG_SCTP -A -q -t if1 > temp.rands.if1
    exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands.if1 &

    # Graph the entire association on one graph
    exec $PERL $SETFID -s all.tr | \
        $PERL $RAW2XG_SCTP -A -q -t all > temp.rands
    exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands &

    exec nam sctp.nam &

    exit 0
}
```

```
set host0_core [$ns node]
set host0_if0 [$ns node]
set host0_if1 [$ns node]
$host0_core color Red
$host0_if0 color Red
```

```

$host0_if1 color Red
$ns multihome-add-interface $host0_core $host0_if0
$ns multihome-add-interface $host0_core $host0_if1

set host1_core [$ns node]
set host1_if0 [$ns node]
set host1_if1 [$ns node]
$host1_core color Blue
$host1_if0 color Blue
$host1_if1 color Blue
$ns multihome-add-interface $host1_core $host1_if0
$ns multihome-add-interface $host1_core $host1_if1

$ns duplex-link $host0_if0 $host1_if0 .5Mb 200ms DropTail
$ns duplex-link $host0_if1 $host1_if1 .5Mb 200ms DropTail

set sctp0 [new Agent/SCTP]
$ns multihome-attach-agent $host0_core $sctp0
$sctp0 set fid_ 0
$sctp0 set debugMask_ -1
$sctp0 set debugFileIndex_ 0
$sctp0 set mtu_ 1500
$sctp0 set dataChunkSize_ 1468
$sctp0 set numOutStreams_ 1
$sctp0 set oneHeartbeatTimer_ 1 # one heartbeat timer shared for all dests

set trace_ch [open trace.sctp w]
$sctp0 set trace_all_ 1 # trace them all on oneline
$sctp0 trace cwnd_
$sctp0 trace rto_
$sctp0 trace errorCount_
$sctp0 attach $trace_ch

set sctp1 [new Agent/SCTP]
$ns multihome-attach-agent $host1_core $sctp1
$sctp1 set debugMask_ -1
$sctp1 set debugFileIndex_ 1
$sctp1 set mtu_ 1500
$sctp1 set initialRwnd_ 131072
$sctp1 set useDelayedSacks_ 1

$ns color 0 Red
$ns color 1 Blue

$ns connect $sctp0 $sctp1

set ftp0 [new Application/FTP]
$ftp0 attach-agent $sctp0

# set primary before association starts
$sctp0 set-primary-destination $host1_if0

# do a change primary in the middle of the association
$ns at 7.5 "$sctp0 set-primary-destination $host1_if1"
$ns at 7.5 "$sctp0 print cwnd_"

$ns at 0.5 "$ftp0 start"
$ns at 10.0 "finish"

```

```
$ns run
```

```
-----  
# EXAMPLE SCRIPT 3  
#  
# Demonstrates multihoming. One endpoint is single homed while the other  
# is multihomed. They are connected through a router. Shows that the they  
# can be combined. The sender is multihomed. The sender has the HEARTBEAT  
# mechanism turned off. In the middle of the association, the sender does  
# a "force-source", which means that the sender forces the source  
# interface used for outgoing traffic.  
#
```

```
#      host0_if0      0===  
#      host0_core    0      \\  
#      host0_if1      0===  
#                      0=====0  host1  
#                      //
```

```
Trace set show_sctphdr_ 1
```

```
set ns [new Simulator]  
set nf [open sctp.nam w]  
$ns namtrace-all $nf
```

```
set allchan [open all.tr w]  
$ns trace-all $allchan
```

```
proc finish {} {  
    global ns nf allchan  
  
    set PERL "/usr/bin/perl"  
    set USERHOME [exec env | grep "^HOME" | sed /^HOME=/s/^HOME=//]  
    set NSHOME "$USERHOME/proj/ns-allinone-2.1b8.sctp"  
    set XGRAPH "$NSHOME/bin/xgraph"  
    set SETFID "$NSHOME/ns-2.1b8/bin/set_flow_id"  
    set RAW2XG_SCTP "$NSHOME/ns-2.1b8/bin/raw2xg-sctp"  
    set GETRC "$NSHOME/ns-2.1b8/bin/getrc"
```

```
    $ns flush-trace  
    close $nf  
    close $allchan
```

```
    exec nam sctp.nam &
```

```
    exit 0  
}
```

```
set host0_core [$ns node]  
set host0_if0 [$ns node]  
set host0_if1 [$ns node]  
$host0_core color Red  
$host0_if0 color Red  
$host0_if1 color Red  
$ns multihome-add-interface $host0_core $host0_if0  
$ns multihome-add-interface $host0_core $host0_if1
```

```

set host1 [$ns node]
$host1 color Blue

set router [$ns node]

$ns duplex-link $host0_if0 $router .5Mb 200ms DropTail
$ns duplex-link $host0_if1 $router .5Mb 200ms DropTail

$ns duplex-link $host1 $router .5Mb 200ms DropTail

set sctp0 [new Agent/SCTP]
$ns multihome-attach-agent $host0_core $sctp0
$sctp0 set fid_ 0
$sctp0 set debugMask_ -1
$sctp0 set debugFileIndex_ 0
$sctp0 set mtu_ 1500
$sctp0 set dataChunkSize_ 1468
$sctp0 set numOutStreams_ 1
$sctp0 set oneHeartbeatTimer_ 0 # turns off heartbeating

set sctp1 [new Agent/SCTP]
$ns attach-agent $host1 $sctp1
$sctp1 set debugMask_ -1
$sctp1 set debugFileIndex_ 1
$sctp1 set mtu_ 1500
$sctp1 set initialRwnd_ 131072
$sctp1 set useDelayedSacks_ 1

$ns color 0 Red
$ns color 1 Blue

$ns connect $sctp0 $sctp1

set ftp0 [new Application/FTP]
$ftp0 attach-agent $sctp0

# force the source interface
$ns at 6.0 "$sctp0 force-source $host0_if1"

$ns at 0.5 "$ftp0 start"
$ns at 8.0 "finish"

$ns run

```

EXAMPLE SCRIPT 4

```

#
# Demonstrates multihoming. Two endpoints with 2 interfaces each all
# connected via a router. The sender has a HEARTBEAT timer for each
# destination. In the middle of the association, a change primary is done.
#
#      host0_if0      0===      ===0      host1_if0
#      /              \ \ //              \
#      host0_core    0          0          0      host1_core
#      \              // \ \              /
#      host0_if1      0===      ===0      host1_if1

```

```
Trace set show_sctphdr_ 1
```

```
set ns [new Simulator]
set nf [open sctp.nam w]
$ns namtrace-all $nf
```

```
set allchan [open all.tr w]
$ns trace-all $allchan
```

```
proc finish {} {
    global ns nf allchan

    set PERL "/usr/bin/perl"
    set USERHOME [exec env | grep "^HOME" | sed /^HOME=/s/^HOME=//]
    set NSHOME "$USERHOME/proj/ns-allinone-2.1b8.sctp"
    set XGRAPH "$NSHOME/bin/xgraph"
    set SETFID "$NSHOME/ns-2.1b8/bin/set_flow_id"
    set RAW2XG_SCTP "$NSHOME/ns-2.1b8/bin/raw2xg-sctp"

    $ns flush-trace
    close $nf
    close $allchan

    exec nam sctp.nam &

    exit 0
}
```

```
set host0_core [$ns node]
set host0_if0 [$ns node]
set host0_if1 [$ns node]
$host0_core color Red
$host0_if0 color Red
$host0_if1 color Red
$ns multihome-add-interface $host0_core $host0_if0
$ns multihome-add-interface $host0_core $host0_if1
```

```
set host1_core [$ns node]
set host1_if0 [$ns node]
set host1_if1 [$ns node]
$host1_core color Blue
$host1_if0 color Blue
$host1_if1 color Blue
$ns multihome-add-interface $host1_core $host1_if0
$ns multihome-add-interface $host1_core $host1_if1
```

```
set router [$ns node]
```

```
$ns duplex-link $host0_if0 $router .5Mb 200ms DropTail
$ns duplex-link $host0_if1 $router .5Mb 200ms DropTail
```

```
$ns duplex-link $host1_if0 $router .5Mb 200ms DropTail
$ns duplex-link $host1_if1 $router .5Mb 200ms DropTail
```

```
set sctp0 [new Agent/SCTP]
$ns multihome-attach-agent $host0_core $sctp0
$sctp0 set fid_ 0
$sctp0 set debugMask_ -1
```

```

$sctp0 set debugFileIndex_ 0
$sctp0 set mtu_ 1500
$sctp0 set dataChunkSize_ 1468
$sctp0 set numOutStreams_ 1
$sctp0 set oneHeartbeatTimer_ 0 # each dest has its own heartbeat timer

```

```

set trace_ch [open trace.sctp w]
$sctp0 set trace_all_ 1 # trace them all on oneline
$sctp0 trace cwnd_
$sctp0 trace rto_
$sctp0 trace errorCount_
$sctp0 attach $trace_ch

```

```

set sctp1 [new Agent/SCTP]
$ns multihome-attach-agent $host1_core $sctp1
$sctp1 set debugMask_ -1
$sctp1 set debugFileIndex_ 1
$sctp1 set mtu_ 1500
$sctp1 set initialRwnd_ 131072
$sctp1 set useDelayedSacks_ 1

```

```

$ns color 0 Red
$ns color 1 Blue

```

```

$ns connect $sctp0 $sctp1

```

```

set ftp0 [new Application/FTP]
$ftp0 attach-agent $sctp0

```

```

$sctp0 set-primary-destination $host1_if0

```

```

# change primary
$ns at 7.5 "$sctp0 set-primary-destination $host1_if1"
$ns at 7.5 "$sctp0 print cwnd_"

```

```

$ns at 0.5 "$ftp0 start"
$ns at 12.0 "finish"

```

```

$ns run

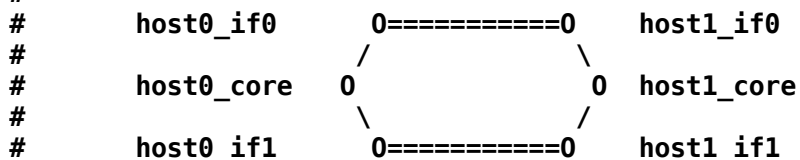
```

EXAMPLE SCRIPT 5

```

#
# Demonstrates Concurrent Multipath Transfer (CMT). Two endpoints with
# 2 interfaces with direct connections between each pair. Data is transfered
# using both paths concurrently. There is a HEARTBEAT timer for each destination.
#

```



```

Trace set show_sctphdr_ 1

```

```

set ns [new Simulator]
set nf [open sctp.nam w]
$ns namtrace-all $nf

```

```

set allchan [open all.tr w]
$ns trace-all $allchan

proc finish {} {
    global ns nf allchan

    set PERL "/usr/bin/perl"
    set NSHOME "/home/nekiz/ns-allinone-2.31"
    set XGRAPH "$NSHOME/bin/xgraph"
    set SETFID "$NSHOME/ns-2.31/bin/set_flow_id"
    set RAW2XG_SCTP "$NSHOME/ns-2.31/bin/raw2xg-sctp"
    set GETRC "$NSHOME/ns-2.31/bin/getrc"

    $ns flush-trace
    close $nf
    close $allchan

    # Graph the host0_if0 <-> host1_if0
    exec $PERL $GETRC -s 1 -d 4 all.tr > all.tr.if0
    exec $PERL $GETRC -s 4 -d 1 all.tr >> all.tr.if0
    exec $PERL $SETFID -s all.tr.if0 | \
        $PERL $RAW2XG_SCTP -A -q -t if0 > temp.rands.if0
    exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands.if0 &

    # Graph the host0_if1 <-> host1_if1
    exec $PERL $GETRC -s 2 -d 5 all.tr > all.tr.if1
    exec $PERL $GETRC -s 5 -d 2 all.tr >> all.tr.if1
    exec $PERL $SETFID -s all.tr.if1 | \
        $PERL $RAW2XG_SCTP -A -q -t if1 > temp.rands.if1
    exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands.if1 &

    # Graph the entire association on one graph
    exec $PERL $SETFID -s all.tr | \
        $PERL $RAW2XG_SCTP -A -q -t all > temp.rands
    exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands &

    exec nam sctp.nam &

    exit 0
}

set host0_core [$ns node]
set host0_if0 [$ns node]
set host0_if1 [$ns node]

$host0_core color Red
$host0_if0 color Red
$host0_if1 color Red

$ns multihome-add-interface $host0_core $host0_if0
$ns multihome-add-interface $host0_core $host0_if1

set host1_core [$ns node]
set host1_if0 [$ns node]
set host1_if1 [$ns node]

$host1_core color Blue
$host1_if0 color Blue

```

```

$host1_if1 color Blue

$ns multihome-add-interface $host1_core $host1_if0
$ns multihome-add-interface $host1_core $host1_if1

$ns duplex-link $host0_if0 $host1_if0 10Mb 45ms DropTail
[$ns link $host0_if0 $host1_if0] queue] set limit_ 50
$ns duplex-link $host0_if1 $host1_if1 10Mb 45ms DropTail
[$ns link $host0_if1 $host1_if1] queue] set limit_ 50

set sctp0 [new Agent/SCTP/CMT]
$ns multihome-attach-agent $host0_core $sctp0
$sctp0 set fid_ 0
$sctp0 set debugMask_ -1
$sctp0 set debugFileIndex_ 0
$sctp0 set mtu_ 1500
$sctp0 set dataChunkSize_ 1468
$sctp0 set numOutStreams_ 1
$sctp0 set useCmtReordering_ 1 # turn on Reordering algo.
$sctp0 set useCmtCwnd_ 1 # turn on CUC algo.
$sctp0 set useCmtDelAck_ 1 # turn on DAC algo.
$sctp0 set eCmtRtxPolicy_ 4 # rtx. policy : RTX_CWND
$sctp0 set oneHeartbeatTimer_ 0 # one heartbeat timer per dest.
$sctp0 set useCmtPF_ 1 # turn on CMT-PF
$sctp0 set cmtPFCwnd_ 2 # cwnd : 2*MTU after HB-ACK

set trace_ch [open trace.sctp w]
$sctp0 set trace_all_ 1 # trace them all on one line
$sctp0 trace cwnd_
$sctp0 trace rto_
$sctp0 trace errorCount_
$sctp0 attach $trace_ch

set sctp1 [new Agent/SCTP/CMT]
$ns multihome-attach-agent $host1_core $sctp1
$sctp1 set debugMask_ -1
$sctp1 set debugFileIndex_ 1
$sctp1 set mtu_ 1500
$sctp1 set initialRwnd_ 65536
$sctp1 set useDelayedSacks_ 1
$sctp1 set useCmtDelAck_ 1

$ns color 0 Red
$ns color 1 Blue

$ns connect $sctp0 $sctp1

set ftp0 [new Application/FTP]
$ftp0 attach-agent $sctp0

# set primary before association starts
$sctp0 set-primary-destination $host1_if0

$ns rtmodel-at 10.0 down $host0_if1 $host1_if1

$ns at 0.5 "$ftp0 start"
$ns at 75.0 "finish"

```

```
$ns run
```

```
# EXAMPLE SCRIPT 6
```

```
#
```

```
# Demonstrates Concurrent Multipath Transfer with Potentially Failed (CMT-PF).  
# Two endpoints with 2 interfaces with direct connections between each pair.  
# A link failure is introduced at 10.0 sec to one of the paths. Data transfer  
# continues with the other path. There is a HEARTBEAT timer for each destination.  
#
```

```
#      host0_if0      0=====0      host1_if0  
#      host0_core    0 \              0 host1_core  
#      host0_if1      0=====0      host1_if1
```

```
Trace set show_sctphdr_ 1
```

```
set ns [new Simulator]  
set nf [open sctp.nam w]  
$ns namtrace-all $nf
```

```
set allchan [open all.tr w]  
$ns trace-all $allchan
```

```
proc finish {} {
```

```
    global ns nf allchan
```

```
    set PERL "/usr/bin/perl"  
    set NSHOME "/home/nekiz/ns-allinone-2.31"  
    set XGRAPH "$NSHOME/bin/xgraph"  
    set SETFID "$NSHOME/ns-2.31/bin/set_flow_id"  
    set RAW2XG_SCTP "$NSHOME/ns-2.31/bin/raw2xg-sctp"  
    set GETRC "$NSHOME/ns-2.31/bin/getrc"
```

```
    $ns flush-trace  
    close $nf  
    close $allchan
```

```
# Graph the host0_if0 <-> host1_if0  
exec $PERL $GETRC -s 1 -d 4 all.tr > all.tr.if0  
exec $PERL $GETRC -s 4 -d 1 all.tr >> all.tr.if0  
exec $PERL $SETFID -s all.tr.if0 | \  
    $PERL $RAW2XG_SCTP -A -q -t if0 > temp.rands.if0  
exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands.if0 &
```

```
# Graph the host0_if1 <-> host1_if1  
exec $PERL $GETRC -s 2 -d 5 all.tr > all.tr.if1  
exec $PERL $GETRC -s 5 -d 2 all.tr >> all.tr.if1  
exec $PERL $SETFID -s all.tr.if1 | \  
    $PERL $RAW2XG_SCTP -A -q -t if1 > temp.rands.if1  
exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands.if1 &
```

```
# Graph the entire association on one graph  
exec $PERL $SETFID -s all.tr | \  
    $PERL $RAW2XG_SCTP -A -q -t all > temp.rands  
exec $XGRAPH -bb -tk -nl -m -x time -y packets temp.rands &
```

```

    exec nam sctp.nam &

    exit 0
}

set host0_core [$ns node]
set host0_if0 [$ns node]
set host0_if1 [$ns node]

$host0_core color Red
$host0_if0 color Red
$host0_if1 color Red

$ns multihome-add-interface $host0_core $host0_if0
$ns multihome-add-interface $host0_core $host0_if1

set host1_core [$ns node]
set host1_if0 [$ns node]
set host1_if1 [$ns node]

$host1_core color Blue
$host1_if0 color Blue
$host1_if1 color Blue

$ns multihome-add-interface $host1_core $host1_if0
$ns multihome-add-interface $host1_core $host1_if1

$ns duplex-link $host0_if0 $host1_if0 10Mb 45ms DropTail
[[$ns link $host0_if0 $host1_if0] queue] set limit_ 50
$ns duplex-link $host0_if1 $host1_if1 10Mb 45ms DropTail
[[$ns link $host0_if1 $host1_if1] queue] set limit_ 50

set sctp0 [new Agent/SCTP/CMT]
$ns multihome-attach-agent $host0_core $sctp0
$sctp0 set fid_ 0
$sctp0 set debugMask_ -1
$sctp0 set debugFileIndex_ 0
$sctp0 set mtu_ 1500
$sctp0 set dataChunkSize_ 1468
$sctp0 set numOutStreams_ 1
$sctp0 set useCmtReordering_ 1      # turn on Reordering algo.
$sctp0 set useCmtCwnd_ 1           # turn on CUC algo.
$sctp0 set useCmtDelAck_ 1        # turn on DAC algo.
$sctp0 set eCmtRtxPolicy_ 4       # rtx. policy : RTX_CWND
$sctp0 set oneHeartbeatTimer_ 0   # one heartbeat timer per dest.
$sctp0 set useCmtPF_ 1            # turn on CMT-PF
$sctp0 set cmtPFCwnd_ 2          # cwnd : 2*MTU after HB-ACK

set trace_ch [open trace.sctp w]
$sctp0 set trace_all_ 1           # trace them all on one line
$sctp0 trace cwnd_
$sctp0 trace rto_
$sctp0 trace errorCount_
$sctp0 attach $trace_ch

set sctp1 [new Agent/SCTP/CMT]
$ns multihome-attach-agent $host1_core $sctp1
$sctp1 set debugMask_ -1

```

```
$sctp1 set debugFileIndex_ 1
$sctp1 set mtu_ 1500
$sctp1 set initialRwnd_ 65536
$sctp1 set useDelayedSacks_ 1
$sctp1 set useCmtDelAck_ 1
```

```
$ns color 0 Red
$ns color 1 Blue
```

```
$ns connect $sctp0 $sctp1
```

```
set ftp0 [new Application/FTP]
$ftp0 attach-agent $sctp0
```

```
# set primary before association starts
$sctp0 set-primary-destination $host1_if0
```

```
$ns rtmodel-at 10.0 down $host0_if1 $host1_if1
```

```
$ns at 0.5 "$ftp0 start"
$ns at 75.0 "finish"
```

```
$ns run
```