INSANE Users Manual

Bruce A. Mah
bmah@CS.Berkeley.EDU
The Tenet Group
Computer Science Division
Univerisity of California at Berkeley

Abstract

This document is the users manual for an Internet Simulated ATM Networking Environment (INSANE). INSANE is an object-oriented, event-driven simulator written in C++ [Stroustrup91] and Tcl [Ousterhout94], designed to simulate a medium-to-large IP-over-ATM internetworking environment.

This version of the manual corresponds to INSANE 1.0a8+ (change bars correspond to changes made since INSANE 1.0a8). Depending on recent changes to the INSANE source code, it is possible that this may not the be the most recent release of the software.

1 Introduction

The Internet Simulated ATM Networking Environment (INSANE) is an object-oriented, discrete-event simulator. It is designed to investigate the performance effects of different IP-over-ATM designs, in a heterogeneous internetwork.

INSANE simulates the operation of a set of typical Internet applications (ftp, telnet, WWW browser, audio, video) using empirical traffic models derived from [Danzig91] and other similar sources. These applications run on top of a simulated Internet protocol stack, consisting of TCP, UDP, and IP. Two different datalink layers (ATM and a generic shared-media LAN) are in turn available to IP. The ATM stack uses FIFO and Rate-Controlled Static Priority (RCSP) [Zhang93] queueing. ATM signalling is performed by a protocol based on the Real-Time Channel Administration Protocol (RCAP). [Mah93]

The simulation core, as well as a set of primitive objects, is implemented in C++. [Stroustrup91] Instances of the various primtive objects can be created and configured into complex, composite objects using the Tcl scripting language. [Ousterhout94] Because Tcl provides general-purpose programming langauge constructs such as loops, conditionals, and procedure definitions, the creation of a large simulation scenario is fairly easy.

A graphical user interface (*insanemon*) facilitates easy monitoring of multiple simulation jobs running on various machines of a distributed computing cluster.

2 Object Hierarchy (still under construction)

This section of the manual describes the portions of the C++ object hierarchy most pertinent to developing new data objects.

The `Pdu` hierarchy is a representation of network packets. We note that the different subclasses of `Pdu` are peers with respect to the C++ class hierarchy, although they may occupy different layers in the protocol stack. These classes are shown in Figure 1, with the parent Pdu class at the top of the figure. Although not shown here, many of the packets can "contain" other packets, representing, for example, the inclusion of a TCP segment inside an IP packet.

The `ScheduledObject` hierarchy contains all objects in the simulation that can send or receive events. These correspond to the primitive objects that can be created by the Tcl interface. Figure 2 shows the various objects of this type. The subclasses of App are shown separately in Figure 3 due to space considerations.

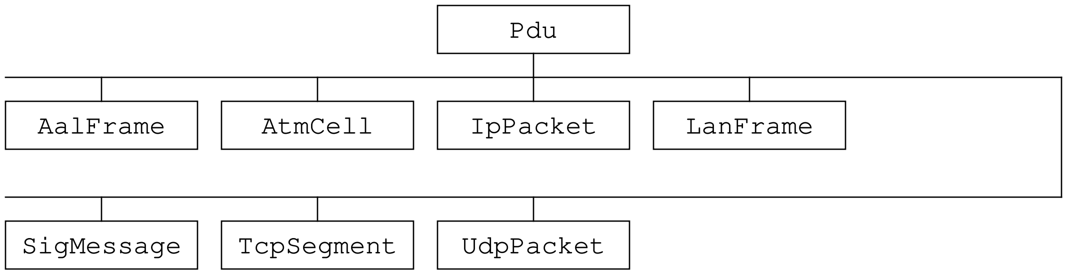

FIGURE 1. INSANE Network Packets. Most network packets shown here point to other packets, symbolizing the inclusion, for example, of a TCP segment inside an IP packet.

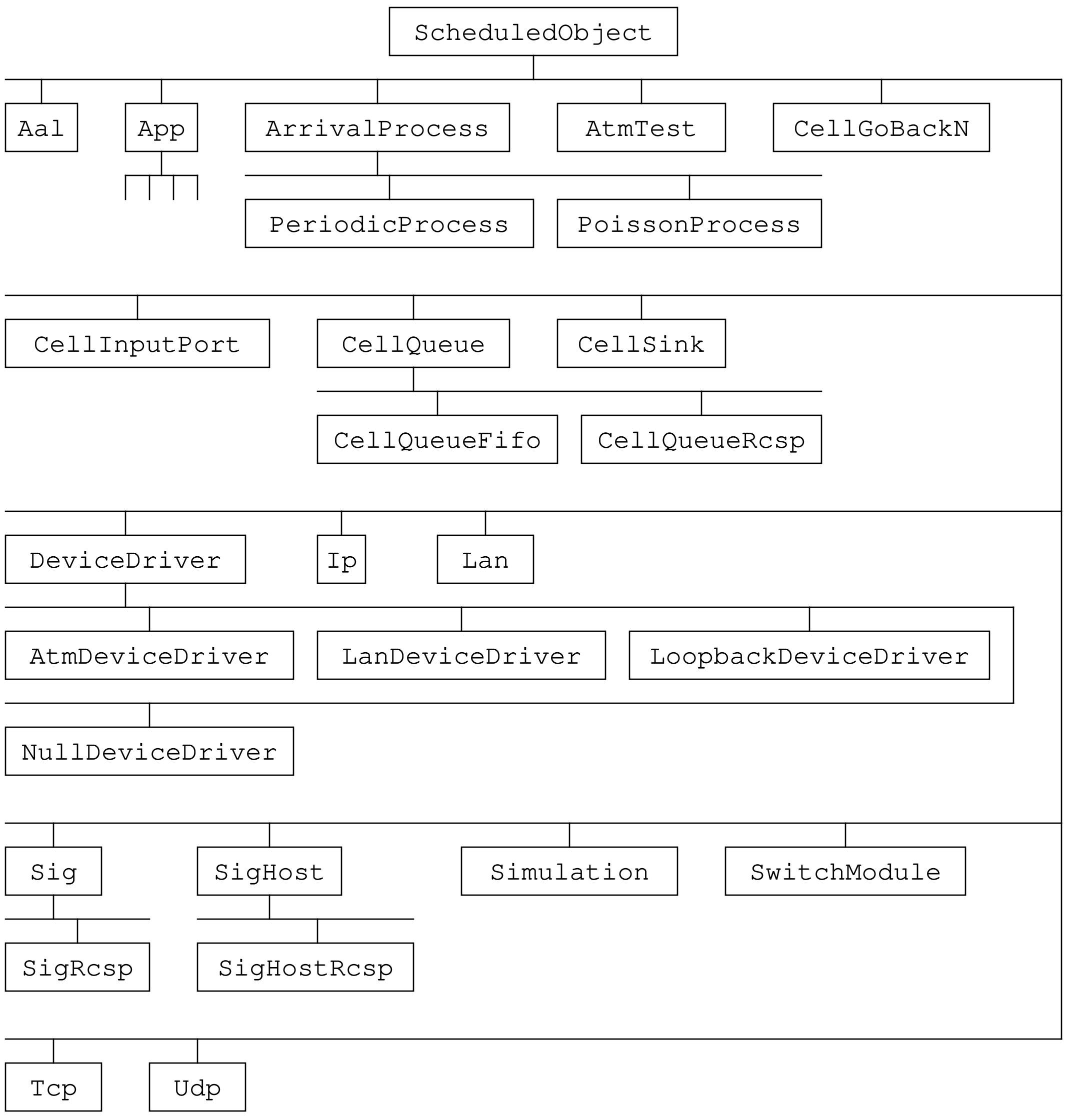

FIGURE 2. INSANE Schedulable Objects. The majority of classes in this figure (plus the Internet applications shown in Figure 3) comprise the set of primitive objects available for simulations.

An important subclass of `ScheduledObject` is `App`, which contains all of the application process classes. These classes stand for various Internet applications that make use of INSANE's network. INSANE currently supports four TCP applications (telnet, FTP, HTTP, and a sample benchmark), as well as two UDP applications (audio and compressed video). Figure 3 shows the object hierarchy for the application classes.

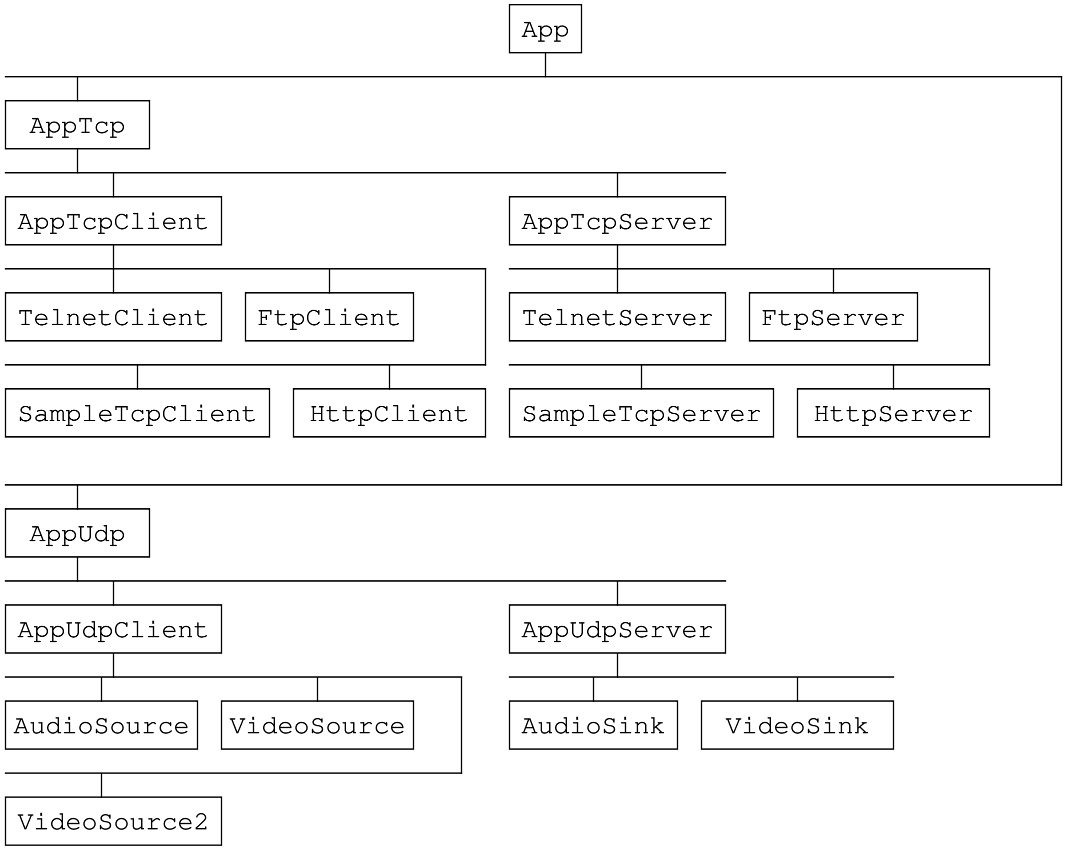

FIGURE 3. INSANE Application Classes.

INSANE includes several random number distribution classes, which are all subclasses of the `g++ Random` class. These random number distributions include several types of empirical distributions (defined by data tables given as arguments to their constructors) and a Zipf's Law distribution. These various distributions are all used by the traffic models and workload generators. The object hierarchy for the random number distribution classes is shown in Figure 4.

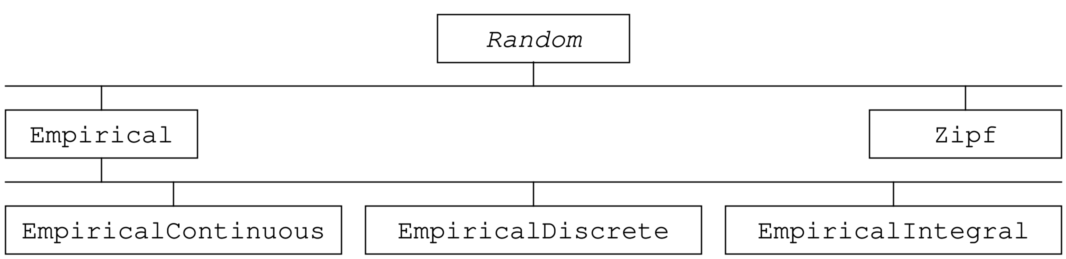

FIGURE 4. INSANE Random Number Distributions.

3 Manual Pages

This section contains documentation on the various classes and commands available in INSANE. It is structured like a set of UNIX manual pages, and lists the various options and commands available to the user from INSANE's command line or within an INSANE configuration file. Section 2 of this documentation describes the built-in INSANE commands. Section 3 describes library functions, which either construct composite objects or perform other utility functions. File formats are listed in Section 5. Finally, Section 7 describes some miscellanous aspects of INSANE.

NAME

insane – Internet Simulated ATM Networking Environment

SYNOPSIS

insane [**-d** *file*] [**-D** *var***=***val*] [**-E**] [**-f** *file*] [**-F**] [**-h**] [**-i**] [**-l** *file*] [**-L** *dir*] [**-R** *host***/***port*] [**-t** *events*] [**-T** *expr*] [**-v**] [**-z** *seed*]

DESCRIPTION

insane is a simulator designed to simulate the use of ATM networks to transport IP datagrams. Its core is a discrete-event simulator. A set of primitive, atomic objects is supplied which simulate various entities inside an IP-over-ATM internetworking environment. The simulator core and atomic objects are all implemented in C++. Tcl scripts can be used to configure each simulation scenario; a library of Tcl scripts is also provided to construct various complex, composite objects.

OPTIONS

-d *file* Set file for debugging output. Setting the debugging file to − (the default), sends debugging output to standard output.

-D *var***=***val*
Define a Tcl variable. This definition is executed during simulation startup, after the Tcl command handlers are installed into the Tcl interpreter but before the configuration file is read. Both the **-D** and **-T** command line options may appear multiple times on the command line, and are evaluated in the order the are encountered. Their intended use it to allow minor inputs to a simulation to come from the command line (for example, to be set from within a batch script).

-E Turn on event tracing, which prints out the timestamp, posting object, and recipient object of every event. This generates large amounts of output; a more workable approach (usually) is to execute **sim trace 1** at some point during the simulation run.

-f *file* Set configuration file (defaults to `./insane.conf`). This file is a Tcl script that specifies the initial setup for the simulator.

-F Flush the debugging stream after every debugging output. Setting this option slows down the simulator, but may be helpful when trying to find bugs in the simulator.

-h Print help information.

-i Interactive mode. INSANE will enter its Tcl command line interface (CLI) before beginning to process events. INSANE provides command-line editing and history functions, if it has been configured to use the GNU readline library.

-l *file* Set file for logging output. Setting the logging file to − (the default) sends logging output to standard output.

-L *dir* Set Tcl autoload library directory (defaults to `$PREFIX/lib/tcl`).

-R *host***/***port*
Enable remote logging with an **insanemon**(1) process listening on host *host* at UDP port *port*. This facility allows easy remote monitoring of multiple INSANE processes running on various hosts, for example the nodes of a distributed computing cluster.

-t *events*
Write a timestamp to the logging stream every *events* events. When the **-R** option is given, **-t** specifies how often to send updates to the remote monitoring host.

-T *expr* Evaluate a Tcl expression. This definition is executed during simulation startup, after the Tcl command handlers are installed into the Tcl interpreter but before the configuration file is read. Both the **-D** and **-T** command line options may appear multiple times on the command line, and are evaluated in the order the are encountered. Their intended use it to allow minor inputs to a simulation to come from the command line (for example, to be set from within a batch script).

-v Print version number, copyright, and build information. Note that this information is also printed in the help message.

-z *seed* Set random number generator seed.

COMMAND SUMMARY

insane has an interactive command line interface (CLI) in which the user can enter Tcl commands to interact with the simulator. All of the commands specified in manual sections 2 and 3, are available, as well as almost all standard Tcl primitives.

CONVENTIONS

In general, commands whose names begin with capital letters create objects; various operations on those objects can then be performed by invoking the name of an object as a Tcl command. Commands whose names begin with lowercase letters usually perform a function.

The simulator core, as well as many objects, have no dependencies on any particular timescale. However, for objects which do need to express time, time values are assumed to be in seconds. All time values are maintained in double-precision floating-point.

BUGS

Probably too many to name, but there are a couple that deserve mention. The random number generator classes used by INSANE (from `libg++`) return results which are dependent on an architecture's byte-order. Since many objects in INSANE depend on random numbers (sometimes in unexpected ways, such as random phase offsets for timers), simulations run with identical input parameters and random number seeds may produce different results on different processors (for example, a little-endian Digital Alpha and a big-endian Sparcstation). Simulations run on different machines with the same byte-ordering (for example, a HP PA-RISC workstation and an SGI MIPS machine), should return the same results.

AUTHOR

Bruce A. Mah <bmah@CS.Berkeley.EDU>.

NAME

insanemon – INSANE monitoring process

SYNOPSIS

insanemon *port*

DESCRIPTION

insanemon is a Tcl/Tk-based monitor for running **insane**(1) processes.

OPTIONS

port UDP port to listen on for status reports.

BUGS

insanemon should flag simulation processes which have finished running, or which have not been heard from recently.

AUTHOR

Bruce A. Mah <bmah@CS.Berkeley.EDU>.

Dialog box artwork by Judy Chan <jchan@mindseye.Berkeley.EDU>.

NAME

intro – Introduction to INSANE built-in primitive commands

SYNOPSIS

object-class name options...

command options...

DESCRIPTION

This section contains information relevent to all INSANE built-in primitive commands.

There are two types of primitive commands. The first type is used to create new objects. A new object is created by invoking the name of the class with some number of parameters. The first parameter is always the name of the new object. A method of the object can then be invoked by using the name of the object as a Tcl command and the name of the method as the first parameter. This convention is patterned after that used by Tk widgets. In this documentation, the parameters used to create an object are listed under the “Options” heading and the object-specific commands are listed under the “Command Summary” heading.

The other type of primitive command performs some function, such as **quit**.

COMMAND SUMMARY

Certain subcommands are common to all object types. The functionality for these subcommands are implemented in the `ScheduledObject` C++ class, which is a parent class for all simulation objects.

object **address**
: Print the address of the C++ structure corresponding to an INSANE object. Mainly for use while debugging new classes.

object **debug** *level*
: Set the debugging level for an object. The higher the debugging level, the more debugging output is generated (each level includes the output for all lower-numbered levels). The default level for all objects is 0, which prints only error messages to the debugging stream.

object **print**
: Print information about a given object. Obviously these data will be object-dependent.

NAME

Aal – Simple ATM adaptation layer object

SYNOPSIS

Aal *name maxVcids gap portNumber*

DESCRIPTION

The **Aal** object implements a simple ATM Adaptation Layer. Although it is similar in functionality to AAL5, the actual implementation is optimized for fast execution in the simulation environment, without actually needing to deal with checksums and packet reassembly.

OPTIONS

name Name of new **Aal** object.

maxVcids
Maximum number of VCIDs supported by this **Aal** instance.

gap Inter-cell gap for cells transmitted by this **Aal** instance.

portNumber
Port number that this **Aal** instance is attached to. **Aal** objects are designed to connect directly to **Switch** objects; the *portNumber* tags each cell with its input port number so that the **Switch** object can correctly forward it downstream.

COMMAND SUMMARY

aal **down** *object*
Set the lower-layer object which handles cell transport. In most cases, this object will be a **Switch** object.

aal **up** *object*
Set the upper-layer object which is a client of the **Aal** instance.

NAME

AtmDeviceDriver – ATM IP device driver

SYNOPSIS

AtmDeviceDriver *name maxAddresses hashSize maxRequests*

DESCRIPTION

This object is a device driver which supports the transmission of IP datagrams over ATM networks. It is responsible for virtual circuit management for use by an IP layer object, and uses a statically-configured translation table to do resolution of IP addresses to ATM addresses.

This class currently implements a very simple set of QOS, multiplexing, and timeout policies, which can be selected by the **muxtype**, **qostype**, and **vctype** options below.

OPTIONS

name
Name of new **AtmDeviceDriver** object.

maxAddresses
Maximum number of addresses supported by the ARP translation table.

hashSize
Size of routing hash table.

maxRequests
Maximum number of pending signalling requests.

COMMAND SUMMARY

add **atmarp** *ip-address atm-address*
Load an entry into the ATM address resolution table. This table provides the ATM address (for establishing a connection) associated with the IP address of each interface directly attached to the ATM subnet.

add **down** *object*
Set the lower-layer protocol object to be used by the **AtmDeviceDriver**. In most cases this will be an **Aal** object.

add **maxcache** *time*
For drivers using the **svccache** VC use policy, sets the maximum time that a virtual circuit can be cached before it is torn down.

add **maxidle** *time*
For drivers using the **svc** or **svccache** VC use policies, sets the maximum idle time that a virtual circuit remains bound. After that time, drivers using the svc policy tear down the virtual circuit, while drivers using the svccache policy

add **muxtype** *type*
Set the multiplexing type. Supported values for type are:

router All packets between a router pair share a virtual circuit.

conv All packets for a given conversation (defined as a TCP connection or a stream of UDP datagrams between a pair of ports) share a virtual circuit.

app All packets between a pair of hosts, for the same application type, share a virtual circuit.

add **netstat** [**-a**] [**-d**]
Print various statistics about this object. This option must be followed by one of the following flags:

-a Print the state of all established IP-over-ATM routes from this host.

-d Print the state of pending downcalls into the ATM signalling system.

add **qostype** *type*
Set the QOS policy. Currently-supported QOS policies are:

noqos All virtual circuits best-effort.

qos1 Virtual circuits are assigned QOS parameters by application type (see source code for details). If a suitable virtual circuit cannot be established, a best-effort virtual circuit is attempted.

fallback
A testing mode in which virtual circuits are set to have establishment parameters that are guaranteed to fail. This mode is useful in testing the establishment of best-effort fallback virtual circuits.

telnet Only telnet traffic is prioritized.

http Only HTTP traffic is prioritized.

audio Only audio traffic is prioritized.

video Only video traffic is prioritized.

add **sighost** *object*
Set the signalling object. This **SigHost** type object must be the one associated with the data delivery object specified by the **down** command above.

add **up** *object*
Set the upper-layer protocol object which calls on this **AtmDeviceDriver**. This object should be an **Ip** object.

add **vctype** *type*
Set the virtual circuit use policy. Allowable values for *type* are:

pvc Pseudo-permanent virtual circuits only. Virtual circuits are established on demand, but never torn down.

svc Switched virtual circuits, fixed idle timeout (specified by the maxidle subcommand).

svccache
Switched virtual circuits, fixed idle timeout and cache time (specified by the **maxidle** and **maxcache** subcommands).

NAME

AtmTest – Native ATM test object

SYNOPSIS

AtmTest *name init sighost aal dest d xmin ave i num size gap*

DESCRIPTION

An object intended to test the ATM protocol stack., including signalling, AAL, and cell transport. It opens up a SVC with the specified parameters, sends some number of AAL frames along the connection, and exits. While not terribly useful in the intended IP-over-ATM simulation environment, the AtmTest object can be used as a starting point for native ATM applications in a different simulation scenario.

OPTIONS

name Name of new AtmTest object.

init Initialization (startup) time for this object.

sighost Name of local signalling entity.

aal Name of adaptation layer object to use.

dest ATM address of destination of cells.

d Delay bound specification for SVC.

xmin Minimum cell interarrival specification for SVC.

xave Average cell interarrival specification for SVC.

i Averaging interval specification for SVC.

num Number of AAL frames to send.

size Size of AAL frames to send.

gap Interframe spacing, in seconds, when sending data.

NAME

AudioSink – Sink for CBR on/off audio source

SYNOPSIS

AudioSink *name start udp port*

DESCRIPTION

The AudioSink object is a sink for CBR audio packets sent via UDP. It keeps statistics on all received packets.

OPTIONS

name Name of new **AudioSink** object.

start Starting time of the process.

udp Name of the **Udp** object to use.

port UDP port number to use.

COMMAND SUMMARY

as **lifetime** [*time*]
Query the lifetime of this **AudioSink** object. If optional *time* parameter is given, set the lifetime.

NAME

AudioSource – Source of CBR on/off audio data

SYNOPSIS

AudioSource *name start udp sink port payload-size gap*

DESCRIPTION

The AudioSource object is a source for CBR audio packets sent via UDP. Data should be sent to a corresponding **AudioSink** object (thus, these two types of objects are normally instantiated in pairs).

The traffic source is an on-off CBR source, with on- and off-times taken from the `phone_talkspurt()` and `phone_pause()` distributios of **tcplib**(3).

OPTIONS

name Name of new **AudioSource** object.

start Starting time of the process.

udp Name of the **Udp** object to use.

server IP address of sink machine.

port UDP port number of **AudioSink** process.

payload-size
Size of audio packet payloads to send.

gap Interarrival time between audio packets.

COMMAND SUMMARY

as **lifetime** [*time*]
Query the lifetime of this **AudioSource** object. If optional *time* parameter is given, set the lifetime.

NAME

CellGoBackN – Reliable retransmission module

SYNOPSIS

CellGoBackN *name numPorts maxCells timeout poll*

DESCRIPTION

A **CellGoBackN** object implements a reliable cell transmission service. It is intended to support ATM signaling entities that require a reliable cell transport, but which must communicate over unreliable networks..

OPTIONS

name Name of new **CellGoBackN** object.

numPorts
Number of ports supported. In general this will be the number of **CellGoBackN** objects that will be directly exchanging cells with the current object.

maxCells
Number of cell buffers. This is the maximum number of outstanding (unacknowledged) cells that can be stored at one time, per port.

timeout Retransmission timeout.

poll Polling interval for retransmission. Every poll seconds, the **CellGoBackN** object scans its list of outstanding cells; those that have been outstanding for longer than *timeout* seconds are resent.

COMMAND SUMMARY

cgbn **down** *object*
Set the lower-layer protocol object that handles cell transport. Typically this will be a **Switch** object.

cgbn **netstat**
Print the state of the retransmission queues.

cgbn **up** *object*
Set the upper-layer protocol object that is a client of this **CellGoBackN** object. This is typically an ATM signalling entity, such as a **Sig** or **SigHost** object.

BUGS

The *timeout* parameter is currently not used.

This object is pretty hacked up. A few version-specific implementation details: This version of the **CellGoBackN** object retransmits at most *maxCells* cells during each polling interval. After a number of cells equal to one-half of *maxCells* queued up, cells are merely queued. The setting of maxCells cannot be too large (the **CellQueue*** objects carrying signalling data need to be able to absorb the bursts of retransmissions)..

NAME

CellInputPort – Input port to ATM switching entity

SYNOPSIS

CellInputPort *name sink port latency*

DESCRIPTION

Due to the object model used by INSANE, an ATM switching entity (**SwitchModule**) cannot determine the input port of incoming ATM cells. Cells need to be explicitly tagged on arrival at the ATM switch. This object provides that functionality.

OPTIONS

name Name of this **CellInputPort** object.

sink Sink for ATM cells received by this object. Typically this will be a **Switch** object.

port Port number to tag cells. In INSANE, this port number is stored as a part of the cell data structure.

latency Delay incurred by a single cell. An assumption made here is that a **CellInputPort** is fast enough to not be a bottleneck to cell processing. As the functionality of a **CellInputPort** is done explicitly in most ATM switches, this appears to be reasonable.

BUGS

This object is a hack. Its functionality shouldn't be needed; **Switch** objects should ideally be able to tell where cells came from.

NAME

CellQueueFifo – FIFO cell queue

SYNOPSIS

CellQueueFifo *name*

DESCRIPTION

A **CellQueueFifo** is a FIFO queue for ATM cells.

OPTIONS

name Name of this **CellQueueFifo** object.

COMMAND SUMMARY

cqf **bitrate** *bitrate*
Set a new bitrate for this **CellQueueFifo**.

cqf **latency** *latency*
Set a new latency for this **CellQueueFifo**.

cqf **output** *object*
Set the sink of the ATM cells passing through this **CellQueueFifo**. For an output-queued ATM switch, this will typically the be the input port (e.g. **CellInputPort**) of the ATM switch on the other end of the link between the two.

NAME

CellQueueRcsp – Rate Controlled Static Priority cell queue

SYNOPSIS

CellQueueRcsp *name maxvcid levels subqueues ticktime*

DESCRIPTION

A **CellQueueRcsp** implements a Rate Controlled Static Priority (RCSP) queue [Zhang93] for ATM cells. A **SigRcsp** or **SigHostRcsp** object (or similar entity) must be used to set the traffic and performance parameters for each virtual circuit before data will flow on that connection.

OPTIONS

name Name of this **CellQueueRcsp** object.

maxvcid
Maximum number of VCIDs supported.

levels Number of priority levels

subqueues
Number of subqueues for calendar queue.

ticktime Interval at which packets become eligible.

COMMAND SUMMARY

cqr **bitrate** *bitrate*
Set a new bitrate for this **CellQueueRcsp**.

cqr **latency** *latency*
Set a new latency for this **CellQueueRcsp**.

cqr **mode** *mode*
Set the scheduling mode, for schedulers that support multiple modes of operation. Valid modes are:

work-conserving
Schedulers will send data whenever the line is idle.

non-work-conserving
Schedulers will buffer data (up to the limit of delay bounds) in order to provide, for example, per-hop jitter control.

no-rate-control
Rate control is disabled. Note that this automatically implies the scheduler will be rate-conserving.

cqr **netstat**
Print the level and buffer usage of all currently active queues.

cqr **output** *object*

Set the sink of the ATM cells passing through this **CellQueueRcsp**. For an output-queued ATM switch, this will typically the be the input port (e.g. **CellInputPort**) of the ATM switch on the other end of the link between the two.

cqr **statinterval** *time*

Set the statistics-printing interval. Statistics are printed at debug level 0.

NOTE

Of the RCSP parameters, *bitrate*, *levels*, and *ticktime* (corresponding to the variables l, n, and $Tick$ in [Zhang93]) are the only ones iassociated with the datapath (we note that the packet size P is fixed because the ATM cell size is fixed). Other parameters are specified in the signalling entity (such as d_m) or on a per-connection basis (such as X_{min}).

BUGS

The implementation of the non-work-conserving RCSP scheduler is not as efficient as it could be.

This class has a "voodoo constant" controlling the amount of buffer allocation for each of the reserved channels (`CellQueueRcspReservedBuffers`). It should somehow be coordinated with the sizes of the retransmission bursts generated by the **CellGoBackN** objects.

NAME

CellSink – Sink for ATM cells

SYNOPSIS

CellSink *name*

DESCRIPTION

A **CellSink** object eats all of the ATM cells sent to it. It is normally used only for debugging purposes.

OPTIONS

name Name of this **CellSink** object.

NAME

EventSink – Sink for simulation events

SYNOPSIS

EventSink *name*

DESCRIPTION

An **EventSink** receives (and discards) events posted to it. It is used almost exclusively for debugging; there should be no need for one of these objects in a normal simulation.

OPTIONS

name Name of this **EventSink** object.

NAME

FtpClient – File Transfer Protocol client application

SYNOPSIS

FtpClient *name start tcp server port*

DESCRIPTION

The **FtpClient** class simulates client programs for the File Transfer Protocol (FTP), for example **ftp**(1) or **ncftp**(1). Once instantiated, the behavior of an **FtpClient** object is dictated by the `ftp_nitems()` and `ftp_ctlsize()` distributions of **tcplib**(3).

OPTIONS

name Name of new **FtpClient** object.

start Starting time of the **FtpClient** process.

tcp Tcp object to be used by this object.

server IP address of FTP server (this is a machine upon which an **FtpServer** object is running).

port TCP port number to use for control data (the Internet standard is 21).

BUGS

In the current version of the **FtpClient** and **FtpServer** objects, all file transfers are downloads; there are no simulated uploads.

The exchange of control messages between client and server is not simulated very well.

There are assumed to be no bottlenecks in the host I/O system on either client or server (i.e. filesystems and disks are assumed to be infinitely fast).

NAME

FtpServer – File Transfer Protocol daemon

SYNOPSIS

FtpServer name start tcp ctlport dataport

DESCRIPTION

The **FtpServer** object represents an File Transfer Protocol (FTP) server running on a machine. It serves requests from **FtpClient** objects, choosing file sizes in accordance with the `ftp_itemsize()` distribution of **tcplib**(3).

OPTIONS

name Name of this **FtpServer**.

start Starting time of the **FtpServer** process.

tcp Tcp class entity to be used.

ctlport Control port used to accept new control connections from clients. Should correspond to the port parameter used by **FtpClient** objects (the Internet default is 21).

dataport Local data port to be used for outbound data transfers (the Internet default is 20).

BUGS

See the corresponding section for the **FtpClient** object.

NAME

HttpClient – HyperText Transfer Protocol client application (WWW browser)

SYNOPSIS

HttpClient *name start tcp servers port threshold*

DESCRIPTION

The **HttpClient** object simulates a client program for the HyperText Transfer Protocol (HTTP). Real-life examples of HTTP client programs are **netscape**(1) and **Mosaic**(1). The object depicts the network traffic generated by a user browsing the World Wide Web using an HTTP browser. User behavior and file retrieval behavior are governed by **tcplib**(3)-style empirical distributions gathered by the author.

OPTIONS

name Name of the **HttpClient** object.

start Starting time of the **HttpClient** process.

tcp **Tcp** object to be used.

servers A Tcl list of IP addresses of HTTP servers to be used. Servers are selected from this list according to a Zipf's Law distribution.

port TCP port to contact servers on (the Internet standard is 80).

threshold
Inactivity threshold. A user whose "think time" between pages exceeds this limit is assumed to have exited her Web browser; the corresponding **HttpClient** object is destroyed.

BUGS

Some real-life Web browsers (in particular, Netscape Navigator), utilize multiple parallel TCP connections to improve interactive response. The **HttpClient** simulated application uses only a single TCP connection at a time.

Most Web browsers can communicate with various kinds of information servers (such as FTP servers and gopher servers). The browser simulated here only uses HTTP.

NAME

HttpServer – HyperText Transfer Protocol server (WWW server)

SYNOPSIS

HttpServer *name start tcp port*

DESCRIPTION

The **HttpServer** simulates server processes for the HyperText Transfer Protocol (HTTP) used on the World Wide Web. It serves requests for files according to a set of empirical distributions gathered by the author.

OPTIONS

name Name of the **HttpServer** object.

start Starting time of the **HttpServer** process.

tcp **Tcp** object to be used.

port TCP port to listen for new requests (the Internet standard is 80).

BUGS

See the manual page for **HttpClient**.

NAME

Ip – Internet Protocol module

SYNOPSIS

Ip *name*

DESCRIPTION

Ip implements a service intended to simulate the IP protocol layer. [Postel81a]

OPTIONS

name Name of this **Ip** module.

COMMAND SUMMARY

ip **ifconfig** *name address netmask broadcast*
Attach a new interface to this **Ip** module. *address*, *netmask*, and *broadcast* are all IP addresses in dotted-quad notation. Note that the **ifconfig** subcommand does not automatically create a route for the interface; this action must be done manually with the **route add** subcommand.

Note that at present there is no way to detatch an interface or change its characteristics.

ip **netstat interfaces**
Print a list of interfaces attached to this **Ip** module.

ip **netstat proto**
Print a list of attached transport layer protocols.

ip **netstat routes**
Print a list of current routes in the routing table.

ip **proto attach** *proto handler*
Attach a transport-layer protocol to this **Ip** module. *proto* is the protocol number from the IP header, and is used to demultiplex inbound packets to the transport layer object *handler*. Common values for *proto* are:

1	ICMP
2	IGMP
6	TCP
17	UDP
255	Raw IP

ip **proto detatch** *proto*
Detatch a transport-layer protocol from this Ip module.

ip **route add** *destination netmask gateway* [*flags*]
Add a route to the routing table. *destination*, *netmask*, and *gateway* are all IP addresses in dotted-

quad notation. *flags* is an optional parameter, consisting of a character string which set bitflags for this route. Current flags are

G Gateway route. A route to a set of hosts not on the current subnet; packets must pass through the router gateway.

Note that default and host routes can easily be expressed using suitable values for *netmask*. A *netmask* of 0.0.0.0 corresponds to a default route. A *netmask* of 255.255.255.255 corresponds to a host route.

Note that there is no concept (at least for now) of a routing metric associated with each routing table entry. For the network topologies anticipated for this simulator, the addition of routing metrics is unnecessary.

ip **route delete** *destination netmask gateway*
Delete a route from the routing table. *destination*, *netmask*, and *gateway* are all IP addresses in dotted-quad notation, and together must match exactly one entry in the routing table.

ip **route query** *dest*
Query the IP routing table for a route to the IP destination dest. If a route exists, returns the IP address of the next-hop gateway router (or destination address, if the last hop). If no route exists, returns an empty string.

NAME

Lan – Shared-medium local area network object

SYNOPSIS

Lan *name maxArp latency bandwidth*

DESCRIPTION

The **Lan** object simulates a shared-medium local area network (similar to Ethernet or FDDI). Machines use the **LanDeviceDriver** object to attach to a **Lan**.

In the interest of simplicity and ease of implementation, the **Lan** class differs in several ways from “real” networks such as Ethernet. First, the **Lan** class uses a “magic”, omniscient FIFO queueing discipline without overhead. Packets are served in the order that they are presented to the **Lan** object.

Second, there is no overhead for contention or collisions, in other words, no attempt to simulate the exponential backoff after Ethernet collisions or the token rotation of FDDI.

Third, there is no analog to the Address Resolution Protocol (ARP); the **LanDeviceDriver** objects obtain this information by accessing a shared public table in the **Lan** object.

OPTIONS

name Name of this **Lan** object.

maxArp Maximum number of ARP entries.

latency Per-packet fixed latency.

bandwidth
Link bandwidth.

COMMAND SUMMARY

lan **arp** *ip-address name*
Load an entry into the address resolution table. *name* should be the object name of the **LanDeviceDriver** that will handle packets destined for *ip-address*. This is accessed by all **LanDeviceDriver** objects associated with this subnet.

lan **statinterval** *time*
Set the statistics-printing interval. Statistics are printed at debug level 0.

NAME

LanDeviceDriver – Simulated host interface and device driver for shared-medium LAN

SYNOPSIS

LanDeviceDriver *name maxqueue latency*

DESCRIPTION

The **LanDeviceDriver** class simulates the device driver and network host interface for a shared-medium local area network. It maintains a FIFO queue for outgoing packets with a fixed maximum length. For a summary of the limitations of the **Lan** and **LanDeviceDriver** class, see the **Lan** documentation.

OPTIONS

name Name of new **LanDeviceDriver** object.

maxqueue
Maximum outgoing queue length. Frames exceeding the capacity of this FIFO queue will be dropped.

latency Driver latency. This latency is incurred on both the inbound and outbound paths.

COMMAND SUMMARY

ldd **down** *object*
Set the lower-layer protocol object to be used by the **LanDeviceDriver**. In all cases this should be a **Lan** class object.

ldd **up** *object*
Set the upper-layer protocol object that calls on this **LanDeviceDriver**. This object should be an **Ip** object.

NAME

LoopbackDeviceDriver – Software loopback device driver

SYNOPSIS

LoopbackDeviceDriver *name latency*

DESCRIPTION

LoopbackDeviceDriver does exactly what its name implies; it is a device driver that reflects packets back to the IP layer (an **Ip** object) in software.

OPTIONS

name Name of this **LoopbackDeviceDriver** object.

latency Latency to process a given packet.

NAME

NullDeviceDriver – Device driver that throws away all packets it receives

SYNOPSIS

NullDeviceDriver *name*

DESCRIPTION

The **NullDeviceDriver** discards all packets given to it by its associated IP layer. It is intended to simulate very primitive forms of filtering (for example, to prevent misrouted packets at a site from leaking out into the backbone).

OPTIONS

name Name of new **NullDeviceDriver** object.

NAME

PeriodicProcess – Execute a Tcl script periodically

SYNOPSIS

PeriodicProcess *name start lambda script*

DESCRIPTION

The **PeriodicProcess** causes repeated invocation of the Tcl script *script* every *lambda* time units.

OPTIONS

name Name of the PeriodicProcess.

start Starting time (first arrival time).

lambda Interarrival time of script invocations

script Tcl script or command to be invoked. Commands that take arguments should be quoted.

NAME

PoissonProcess – Execute a Tcl script according to a Poisson arrival process

SYNOPSIS

PoissonProcess *name start lambda script*

DESCRIPTION

This object class causes repeated invocation of the Tcl script *script*, with interarrivals according to a Poisson process with average interarrival time lambda.

OPTIONS

name Name of **PoissonProcess** object.

start Starting time of **PoissonProcess** (and first arrival time).

lambda Average interarrival time.

script Tcl command or script to be executed.

NAME

quit – Exit insane

SYNOPSIS

quit

DESCRIPTION

Exit INSANE immediately, without doing any cleanup or end-of-simulation statistics gathering. Note that this command only really makes sense from within the Tcl command line interpreter (CLI). Placing this command in a script (such as `insane.conf`) is legal, but fairly meaningless.

NAME

SampleTcpClient – Test client application for TCP

SYNOPSIS

SampleTcpClient *name start tcp server port length count gap*

DESCRIPTION

The **SampleTcpClient** object is a test application originally used in the debugging of the Tcp module. It attempts a TCP connection to a **SampleTcpServer** and performs a number of sends as determined by the *length*, *count*, and *gap* parameters.

OPTIONS

name Name of the new **SampleTcpClient** object.

start Session start time.

tcp Name of the **Tcp** object to use. Specifying this object automatically associates the **SampleTcpClient** object with a specific **Ip** node in the simulation.

server Server IP address, in dotted quad notation.

port Server port address.

length Length of TCP sends in bytes. Note that this parameter does not completely determine the size of outgoing TCP segments; the MSS parameter of TCP may break up each send into smaller segments.

count Number of TCP sends.

gap Gap between TCP sends.

NAME

SampleTcpServer – Test server application for TCP

SYNOPSIS

SampleTcpServer *name start tcp port*

DESCRIPTION

The **SampleTcpServer** is a test application originally written to help debug the **Tcp** class. It accepts connections from a **SampleTcpClient** and sinks all data sent on the connections.

OPTIONS

name Name of new **SampleTcpServer** object.

start Starting time of the new **SampleTcpServer** object. This time should be relatively close to the simulation start time, so that the **SampleTcpServer** is initialized before any new TCP connection requests arrive.

tcp Name of underlying **Tcp** object. Specifying this object automatically associates the **SampleTcpServer** object with an IP node.

port Port number on which to listen for connection requests

NAME

Sig – ATM signalling entity (no admission control)

SYNOPSIS

Sig *name numPorts maxVcids maxRoutes*

DESCRIPTION

The **Sig** object is a signalling entity for a simulated ATM switch. In the terminology of the ATM Forum, this object implements the NNI. It (and its host-based counterpart, the **SigHost** object) implement a signalling protocol very similar to the Real-Time Channel Administration Protocol. [Mah93] It uses the **CellGoBackN** object for reliable delivery of signalling messages sent across the ATM network.

This class of **Sig** object is designed to run on networks without admission control; various methods in this class can be overridden to work with networks requiring admission tests at connection setup time.

OPTIONS

name Name of new **Sig** object.

numPorts
Number of switch ports supported.

maxVcids
Maximum number of VCIDs (and hence connections) supported per switch port. Note that VCIDs on the input and output "sides" of a given switch port are counted separately.

maxRoutes
Maximum number of routes supported. The Sig object supports a simple table-driven static routing scheme. *maxRoutes* also implicitly defines the possible range of ATM host addresses (they occupy the range $[0 \ldots maxRoutes]$.

COMMAND SUMMARY

sig **gobackn** *object*
Specify the reliable transport entity (typically a **CellGoBackN** object) to be used by this Sig object.

sig **netstat** [**-v**]
Prints out the number of active connections (as seen by the signalling software). If the **-v** flag is given, prints the current connection table.

sig **output** *port object*
Specify an output port *object* (typically a **CellQueue*** object). This option allows a **Sig** object to specify traffic parameters or other per-channel information to output queues of an output-queued switch.

sig **mode** *mode*
Set the scheduling mode, for schedulers that support multiple modes of operation. Valid modes are:

work-conserving
Schedulers will send data whenever the line is idle.

non-work-conserving

Schedulers may, at their option, buffer data (up to the limit of delay bounds) in order to provide, for example, per-hop jitter control.

no-rate-control

Rate control is disabled for schedulers that support that feature. Note that this automatically implies the scheduler will be rate-conserving.

This distinction may be important to signalling entities that need to perform buffer space calculations during admission control. Not all scheduler/admission control schemes support this switch.

sig **route** *dest port*

Specify the *port* number to use as the next hop towards a destination address *dest*.

sig **set** *input-port input-vcid output-port output-vcid*

Manually configure a virtual circuit association. This command sets up the appropriate signalling state for a connection from (*input-port*, *input-vcid*) to (*output-port*, *output-vcid*). The main use of this command is to make the state of a **Sig** object consistent with a manually-entered connection in an associated **SwitchModule**.

sig **switch** *object*

Specify the **SwitchModule** attached to this signaling entity.

NAME

SigHost – ATM host signalling entity

SYNOPSIS

SigHost *name maxVcids downCalls address*

DESCRIPTION

SigHost objects implement the host portion of an ATM signalling protocol (in ATM Forum terminology, the UNI). This signalling protocol is based on the Real-Time Channel Administration Protocol (RCAP) described in [Mah93].

This class of **SigHost** object is designed to run on networks without admission control; various methods in this class can be overridden to work with networks requiring admission tests at connection setup time.

OPTIONS

name Name of this **SigHost** object.

maxVcids
Maximum number of VCIDs supported by this **SigHost** object.

downCalls
Maximum number of signalling downcalls supported from higher-layer protocol objects. This option sets the maximum number of simultaneous pending signalling calls.

address Address of this ATM host (an integer greater than zero).

COMMAND SUMMARY

sighost **aal** *object*
Specify the host **Aal** object, in order that the **SigHost** object can properly reset new inbound connections.

sighost **mode** *mode*
Set the scheduling mode, for schedulers that support multiple modes of operation. Valid modes are:

work-conserving
Schedulers will send data whenever the line is idle.

non-work-conserving
Schedulers may, at their option, buffer data (up to the limit of delay bounds) in order to provide, for example, per-hop jitter control.

no-rate-control
Rate control is disabled for schedulers that support that feature. Note that this automatically implies the scheduler will be rate-conserving

This distinction may be important to signalling entities that need to perform buffer space calculations during admission control. Not all scheduler/admission control schemes support this switch.

sighost **net** *object*

Specify the object which handles cells bound for the network, usually a **CellQueue*** object. This specification is needed so that the **SigHost** object can set quality-of-service parameters (if applicable) for outgoing channels.

sighost **netstat** [**-v**]

Prints out the number of active connections (as seen by the signalling software). If the **-v** flag is given, prints the current connection table.

sighost **switch** *object*

Specify the object that handles cells bound for the switch (typically a **CellGoBackN** object).

NAME

SigHostRcsp – ATM host signalling entity with RCSP support

SYNOPSIS

SigHostRcsp *name maxVcids downCalls address*

DESCRIPTION

A **SigHostRcsp** object implements the host portion of the signalling for the simulated ATM network. It performs admission control tests for the Rate Controlled Static Priority (RCSP) service discipline. [Zhang93, Zhang94]

OPTIONS

name Name of this **SigHostRcsp** object.

maxVcids
Maximum number of VCIDs supported by this **SigHostRcsp** object.

downCalls
Maximum number of signalling downcalls supported from higher-layer protocol objects. This option sets the maximum number of simultaneous pending signalling calls.

address Address of this ATM host (an integer greater than zero).

COMMAND SUMMARY

sighostrcsp **aal** *object*
Specify the host **Aal** object, in order that the **SigHostRcsp** object can properly reset new inbound connections.

sighostrcsp **excess** *buffers*
Set the number of excess buffers to be granted to each connection established, above and beyond that specified by the buffer calculations performed by the admission control tests. The default value of this parameter is 0.

sighostrcsp **mode** *mode*
Set the scheduling mode, for schedulers that support multiple modes of operation. Valid modes are:

work-conserving
Schedulers will send data whenever the line is idle.

non-work-conserving
Schedulers may, at their option, buffer data (up to the limit of delay bounds) in order to provide, for example, per-hop jitter control.

no-rate-control
Rate control is disabled for schedulers that support that feature. Note that this automatically implies the scheduler will be rate-conserving

This distinction may be important to signalling entities that need to perform buffer space calculations during admission control. Not all scheduler/admission control schemes support this switch.

sighostrcsp **net** *object bitrate ticktime delays*

Specify the **CellQueueRcsp** object which handles cells bound for the network. For the purpose of admission control tests, the network port has a line speed of *bitrate* and an RCSP *Tick* parameter of *ticktime*. delays is a Tcl list of the delay bounds for each level, in increasing order. Note that no sanity or consistency checking is made between these parameters and those specified to the corresponding **CellQueueRcsp** object.

sighost **netstat** [**-v**]

Prints out the number of active connections (as seen by the signalling software). If the **-v** flag is given, prints the current connection table.

sighostrcsp **switch** *object*

Specify the object that handles cells bound for the switch (typically a **CellGoBackN** object).

NAME

SigRcsp – Switch signalling entity supporting Rate Controlled Static Priority scheduling

SYNOPSIS

SigRcsp *name numPorts maxVcids maxRoutes*

DESCRIPTION

The **SigRcsp** entity handles the signalling for one node of an ATM network using Rate Controlled Static Priority (RCSP) queueing. [Zhang93] Its behavior is very similar to the **Sig** class from which it is derived, but it implements admission control tests for RCSP. [Zhang94]

OPTIONS

name Name of new **SigRcsp** object.

numPorts
Number of switch ports supported.

maxVcids
Maximum number of VCIDs (and hence connections) supported per switch port. Note that VCIDs on the input and output "sides" of a given switch port are counted separately.

maxRoutes
Maximum number of routes supported. The Sig object supports a simple table-driven static routing scheme. *maxRoutes* also implicitly defines the possible range of ATM host addresses (they occupy the range $[0 \ldots maxRoutes]$.

COMMAND SUMMARY

sigrcsp **excess** *buffers*
Set the number of excess buffers above and beyond those specified by the buffer calculations in the RCSP admission control tests. The default value for this parameter is 48, which gives

sigrcsp **gobackn** *object*
Specify the reliable transport entity (typically a **CellGoBackN** object) to be used by this Sig object.

sigrcsp **netstat** [**-v**]
Prints out the number of active connections (as seen by the signalling software). If the **-v** flag is given, prints the current connection table.

sigrcsp **mode** *mode*
Set the scheduling mode, for schedulers that support multiple modes of operation. Valid modes are:

work-conserving
Schedulers will send data whenever the line is idle.

non-work-conserving
Schedulers may, at their option, buffer data (up to the limit of delay bounds) in order to provide, for example, per-hop jitter control.

no-rate-control

Rate control is disabled for schedulers that support that feature. Note that this automatically implies the scheduler will be rate-conserving

This distinction may be important to signalling entities that need to perform buffer space calculations during admission control. Not all scheduler/admission control schemes support this switch.

sigrcsp **output** *port object bitrate ticktime delays*

Specify an output port *object* (specifically, a **CellQueueRcsp** object). The output port (and hence the outgoing link) has a line speed of *bitrate* and an RCSP *Tick* parameter of *ticktime*. delays is a Tcl list of the delay bounds for each level, in increasing order. Note that no sanity or consistency checking is made between these parameters and those specified to the corresponding **CellQueueRcsp** object.

sigrcsp **route** *dest port*

Specify the *port* number to use as the next hop towards a destination address *dest*.

sigrcsp **set** *input-port input-vcid output-port output-vcid*

Manually configure a virtual circuit association. This command sets up the appropriate signalling state for a connection from (*input-port*, *input-vcid*) to (*output-port*, *output-vcid*). The main use of this command is to make the state of a **SigRcsp** object consistent with a manually-entered connection in an associated **SwitchModule**. Note that no resource parameters are set for these manually entered connection; thus it is unlikely that any performance guarantees can be honored.

sigrcsp **switch** *object*

Specify the **SwitchModule** attached to this signaling entity.

NAME

Simulation – INSANE simulation meta-object

SYNOPSIS

(none – the **Simulation** object is internally generated)

DESCRIPTION

The **Simulation** object is a “meta-object” that helps to manage certain aspects of a running simulation. A single **Simulation** object is internally generated at startup time; this object (currently named `sim`) can be given subcommands as with any other object.

COMMAND SUMMARY

sim at *time script*
Cause execution of the Tcl script at time time. Generally this command is useful for increasing/decreasing debugging levels or event tracing during debugging, in situations where a large amount of output is required for a small interval of time during the run.

sim debug *time*
Cause the simulation to stop execution at a given time. The user can interact with the simulator through the Tcl command line interface (CLI) and can restart the simulation by issuing the command **sim go**.

sim go [*time*]
This command is only meaningful in the CLI. It causes simulation execution will continue. If the optional *time* parameter is given, simulation execution will continue until the specified time.

sim halt *time*
Halt the simulation at a given time. The simulation process exits.

sim list Return a Tcl list containing the names of all objects in the simulation.

sim mdebug *value*
This command is only supported if INSANE is running with a debugging version of the CSRI malloc package that supports debugging. *value* is used as the argument to `mal_debug()`, which controls the amount of error checking performed during each memory allocation operation. For more details, consult the CSRI malloc documentation.

sim trace *value*
Enable or disable INSANE’s event tracing mechanism. When event tracing is enabled, every event is printed to the debugging stream before it is delivered to its recipient. If *value* is non-zero, event tracing is enabled; if *value* is zero, event tracing is disabled. Note that event tracing can also be enabled from the command line. Due to the voluminous output produced, it will probably be most useful to use this command in conjunction with the **sim at** subcommand, described above.

sim when
Return the current simulation time.

NAME

SmtpClient – SMTP mail transfer client program

NAME

SwitchModule – ATM switching entity

SYNOPSIS

SwitchModule *name numPorts maxVcids*

DESCRIPTION

A **SwitchModule** object forwards cells from an input (port, vcid) to an output (port, vcid). It is the central element on the data forwarding path of ATM switches, but also is used in ATM host adapters. A **SwitchModule** is generally controlled by a **Sig** or a **SigHost** signalling entity.

OPTIONS

name Name of this **SwitchModule** object.

numPorts
Number of data ports supported by this **SwitchModule** object. Note that for an $N \times N$ ATM switch, *numPorts* should be set to $N + 1$, as the signalling entity will always consume a port (by convention, the INSANE library uses port 0 for this purpose).

maxVcids
Maximum number of VCIDs supported by this SwitchModule.

COMMAND SUMMARY

sm **set** *input-port input-vcid output-port output-vcid*
Manually configure an association that maps cells arriving on (*input-port*, *input-vcid*) to (*output-port*, *output-vcid*).

sm **output** *num object*
Specify the object that receives cells for a given output port. Typically this will be a **CellQueue*** object.

NAME

Tcp – Transmission Control Protocol module

SYNOPSIS

Tcp *name*

DESCRIPTION

As its name implies, this class of objects implements TCP for applications. A large subset of the TCP functionality is supported, including connection management, TCP slowstart, fast retransmit and recovery, congestion control, retransmission, and keepalives. Simulated applications can tag TCP segments with user-level data, thus allowing them to exchange small amounts of data (such as sequence numbers or delimiters for application-layer protocols).

This implementation is based heavily on the TCP implementation found in BSD 4.4Lite. [Postel81b, Stevens94, Wright95]

OPTIONS

name Name of new **Tcp** module.

COMMAND SUMMARY

tcp **down** *ipmodule*

Name of **Ip** module used on the current network node. *ipmodule* will be used for the transport of all TCP segments.

tcp **netstat**

Print current TCP entity state. All PCB blocks are printed, showing the local and source IP addresses and ports, as well as the TCP connection state.

METHODS

Several methods are available to simulated applications using Tcp objects. In general, they are similar to the similarly-named BSD socket calls.

```
TcpPcb *Tcp::UserListen(double time, unsigned short port,
       ScheduledObject *app)
```

Listen on a TCP port for incoming connections. Returns a pointer to a `TcpPcb` structure, roughly analogous to a socket. `app` object will receive an EventTcpConnect event when a new connection is accepted. This call is roughly analogous to a **socket**(2), **bind**(2), and **listen**(2) sequence.

```
TcpPcb *Tcp::UserConnect(double time, InAddr addr, unsigned short port,
       ScheduledObject *app)
```

Initiate a TCP connection to another process, specified by `addr` and `port`. Returns a pointer to a `TcpPcb` structure, roughly analogous to a socket, that can be used by future operations which need to refer to this TCP connection. `app` object will receive an EventTcpConnect event when a new connection is accepted. This call is similar to a **socket**(2), **bind**(2), and **accept**(2) sequence.

```
void Tcp::UserClose(double time, TcpPcb *tp)
```

Close a TCP connection. Analogous to performing a **close**(2) socket call.

`void Tcp::UserSend(double time, TcpPcb *tp, short len)`
Send data on a TCP connection.

`void Tcp::UserSend(double time, TcpPcb *tp, short len, TcpData *data)`
Send tagged data on a TCP connection. The data tag pointed to by data will be delivered to the destination process when the last byte of this send has been successfully delivered to the destination. The destination process can read the tag by examining the data member of the `TcpCall` structure delivered with the corresponding EventTcpArrival event.

`void Tcp::UserSet(double time, TcpPcb *tp, int parm)`
Set flags in a TCP PCB. Allowable values for `parm` are:

`TF_NODELAY`
Don't delay data to coalesce small packets.

`void Tcp::UserReset(double time, TcpPcb *tp, int parm)`
Reset flags in a TCP PCB. The allowable values for `parm` are the same as for `Tcp::UserSet`.

BUGS

Only a subset of the full TCP functionality is supported. Features that were omitted in the interests of simplicity include:

TCP checksums

Out-of-band (urgent) data

TCP options

Window scaling

RFC 1323 timestamps

The various methods can take an arbitrary time as an argument, but they may behave rather strangely if the `time` argument is too far in the future. Specifically, the **Tcp** module processes the method invocations in the order of invocation, not in the order of timestamps, so it is possible to invoke methods in an order different than they would be called "in real life". To avoid this problem, applications should be designed to call the Tcp methods with a `time` parameter equal to the the `time` parameter of a currently running method handler, or with an small increment.

NAME

TelnetClient – Remote login client

SYNOPSIS

TelnetClient *name start tcp server port timestamps*

DESCRIPTION

The **TelnetClient** object simulates the operation of a remote login client such as **telnet**(1) or **rlogin**(1). It uses the distributions from **tcplib**(3) to determine the length of each conversation and the interarrival time of keystrokes at the client side of an application (these are the `telnet_duration()` and `telnet_interarrival()` distributions, respectively). At the end of the conversation, statistics are printed to the logging stream.

OPTIONS

name Name of the new **TelnetClient** object.

start Session start time. The session ending time is determined by **tcplib**.

tcp Name of the **Tcp** object to use. Specifying this object automatically associates the **TelnetClient** object with a specific **Ip** node in the simulation.

server Server IP address, in dotted quad notation.

port Server port address. Any port number is allowed here (as long as it matches with the port being used by a **TcpServer** object), but for consistency and authenticity, the Internet standard **telnet** port number of 23 is recommended.

timestamps
Size of timestamps array. The **TelnetClient** keeps an array of timestamps used to help measure the roundtrip time of the simulated **telnet** packets. This parameter controls the number of outstanding client packets that may be measured. If this number is exceeded, an error message is printed to the debugging stream.

NAME

TelnetServer – Remote login server

SYNOPSIS

TelnetServer *name start tcp port*

DESCRIPTION

The **TelnetServer** object simulates the operation of a remote login server (essentially this is the functionality of **telnetd**(1) or **rlogind**(1), plus a portion of the functionality of **inetd**(1), for starting new sessions). There will be typically a single **TelnetServer** object per IP node.

The **TelnetServer** accepts TCP connections on a port. For each tagged data byte received from a **TelnetClient**, it sends back a response drawn from the `telnet_pktsize()` distribution of **tcplib**(3).

OPTIONS

name Name of new **TelnetServer** object.

start Starting time of the new **TelnetServer** object. This time should be relatively close to the simulation start time, so that the **TelnetServer** is initialized before any new TCP connection requests arrive.

tcp Name of underlying **Tcp** object. Specifying this object automatically associates the **TelnetServer** object with an IP node.

port Port number on which to listen for connection requests. Using the Internet standard **telnet** port of 23 is recommended.

NAME

Udp – User Datagram Protocol module

SYNOPSIS

Udp *name*

DESCRIPTION

Not too surprisingly, **Udp** objects implement UDP [Postel80] for applications.

OPTIONS

name Name of new **Udp** module.

COMMAND SUMMARY

udp **down** *ipmodule*
Name of **Ip** module used on the current network node. *ipmodule* will be used for the transport of all UDP datagrams.

udp **netstat**
Print the state of all UDP PCBs, showing the local and foreign IP address and port bindings.

METHODS

Several methods are available to simulated applications using Udp objects. Their names and functionality are similar to that of the BSD socket system calls.

```
UdpPcb *Udp::UserBind(double time, InAddr faddr, unsigned short fport,
       InAddr laddr, unsigned short lport, ScheduledObject *owner)
```
Create a new UDP "socket" bound to the appropriate IP addresses and ports. IP addresses and ports set to zero are wildcards. The object `owner` (almost always will be the application process) will receive EventUdpArrival events for incoming packets to this "socket", until it executes a `Udp::UserClose` call.

```
void Udp::UserClose(double time, UdpPcb *pcb)
```
Close a previously bound UDP "socket". This call destroys the UDP PCB.

```
void Udp::UserSend(double time, UdpPcb *up, short len)
```
Send data on a UDP "socket".

```
void Udp::UserSend(double time, UdpPcb *up, short len, UdpData *data)
```
Send data on a UDP "socket". The datagram includes a single data tag.

```
void Udp::UserSend(double time, UdpPcb *up, short len, UdpData *data,
       UdpPayloadType type)
```
Send data on a UDP "socket". The datagram includes a data tag and is also marked as having a payload type of `type`.

NAME

uniform – Uniform random number generator

SYNOPSIS

uniform

DESCRIPTION

uniform returns a random number from a Uniform[0,1) distribution. It uses the single random number generator used by the rest of INSANE; therefore, changing the seed used by INSANE will change the stream of random numbers produced by **uniform** as well.

NAME

VideoSink – Sink of CBR video data

SYNOPSIS

VideoSink *name start udp port*

DESCRIPTION

The **VideoSink** object is a sink for CBR audio packets sent via UDP. It keeps statistics on all received packets.

OPTIONS

name Name of new **VideoSink** object.

start Starting time of the process.

udp Name of the **Udp** object to use.

port UDP port number to use.

COMMAND SUMMARY

vs **lifetime** [*time*]
Query the lifetime of this **VideoSink** object. If optional *time* parameter is given, set the lifetime.

NAME

VideoSource – Source of CBR video data

SYNOPSIS

VideoSource *name start udp server port payload-size gap*

DESCRIPTION

The **VideoSource** object is a source for CBR audio packets sent via UDP. Data should be sent to a corresponding **VideoSink** object (thus, these two types of objects are normally instantiated in pairs).

OPTIONS

name Name of new **VideoSource** object.

start Starting time of the process.

udp Name of the **Udp** object to use.

server IP address of sink machine.

port UDP port number of **VideoSink** process.

payload-size
Size of audio packet payloads to send.

gap Interarrival time between audio packets.

COMMAND SUMMARY

vs **lifetime** [*time*]
Query the lifetime of this **VideoSource** object. If optional *time* parameter is given, set the lifetime.

NAME

VideoSource2 – Source of VBR video data

SYNOPSIS

VideoSink2 *name start udp server port bitrate framerate*

DESCRIPTION

The **VideoSource2** object transmits VBR video data to a **VideoSink** object. This object uses a two-state model, corresponding to video tools which do conditional replenishment of data, combined with background updates. Examples of these video tools are **vic**(1) and **vgw**(1). Within each of these transmission states, packet sizes are determined by empirical distributions derived from actual packet traces. Transitions between the states (more specifically, the distribution of lifetimes in the two states) are also determined by data from packet traces.

OPTIONS

name Name of new **VideoSource2** object.

start Starting time of the process.

udp Name of the **Udp** object to use.

server IP address of sink machine.

port UDP port number of **VideoSink2** process.

bitrate Target bitrate. This parameter determines the interpacket gap when sending conditional replenishment packets.

framerate
Target framerate. This parameter determines the interpacket gap when sending background update packets.

COMMAND SUMMARY

vs **lifetime** [*time*]
Query the lifetime of this **VideoSource2** object. If optional *time* parameter is given, set the lifetime.

NAME

zipf – Zipf's Law distribution

SYNOPSIS

zipf [*n*]

DESCRIPTION

zipf generates values distributed according to a Zipf's Law distribution.

OPTIONS

n Number of items to choose from, a parameter to the Zipf's Law distribution. If *n* is not provided, **zipf** reuses the value from the last invocation.

NOTES

When using **zipf** to choose an element from a Tcl list, note that **zipf** returns values in the range $[1 \ldots n]$, but **lindex**(n), *et al*. require values in the range $[0 \ldots n-1]$.

NAME

intro – Introduction to library commands

SYNOPSIS

object-class name options...

command options...

DESCRIPTION

This section describes the various INSANE library functions. The library functions are all implemented as Tcl scripts placed in the autoload directory (by default, this directory is `$PREFIX/lib/tcl`). (This implementation is in contrast to the built-in commands, which are all implemented in C++.)

As with built-in commands (see **intro**(2)), there are two types of library commands. The first creates objects, in this case composite objects built up from one or more primitive objects. The primitive objects can be referred to separately from the composite object; this allows one to, for example, define an object using a library command and later manipulate the components of the object to set various parameters or to specify connections to other objects. Commands of this type generally begin with a capital letter; any primitive objects created by an invocation of these commands will generally have the name of the composite object as a prefix.

The second type of library command performs various utility functions. These commands have names beginning with a lowercase letter.

A number of the library routines use global Tcl variables to configurate the various composite objects created. We present a short synopsis of them here:

ConfAtmDeviceDriverQosType
Simulation global IP-over-ATM quality-of-service policy (used as a parameter to **AtmDeviceDriver qostype** command). Defaults to **noqos**.

ConfAtmDeviceDriverMuxType
Simulation global IP-over-ATM multiplexing policy (used as a parameter to **AtmDeviceDriver muxtype** command). Defaults to **router**.

ConfAtmDeviceDriverVcType
Simulation global IP-over-ATM virtual circuit policy (used as a parameter to **AtmDeviceDriver vctype** command). Defaults to **svc**.

ConfHostRcspExcess
Parameter for **SigHostRcsp excess** command. No default.

ConfHostRcspMode
Scheduling mode (**CellQueueRcsp mode**) for RCSP host queues. Defaults to **non-work-conserving**.

ConfRccpExcess
Setting for **SigRcsp excess** command. No default

ConfRcspMode

Scheduling mode (**CellQueueRcsp mode**) for RCSP switch queues. Defaults to **non-work-conserving**.

NAME

atmFifoAdd – Add a FIFO ATM interface to an existing IP host

SYNOPSIS

atmFifoAdd *hostname maxvcid bw addr num ip netmask*

DESCRIPTION

atmFifoAdd creates and attaches a new ATM interface to an existing IP host. The interface is made known to the **Ip** module on the host and an appropriate route is installed in the IP routing table.

OPTIONS

hostname
Prefix of the host.

maxvcid
Maximum number of VCIDs supported.

bw Bitrate of the outgoing line.

addr ATM-layer address of this end station.

num Interface number. The interface will appear to the IP layer with the name `atmd`*num*.

ip IP address to be assigned to this interface

netmask Netmask for the subnet.

COMPONENTS

hostname.**atm***num*
Host adapter providing a frame transport interface (**HostAdapterFifo**).

hostname.**atmd***num*
Device driver object visible to IP (**AtmDeviceDriver**).

FILES

`$PREFIX/lib/tcl/AtmHost.tcl`
Location of the Tcl source code.

NAME

atmRcspAdd – Add an RCSP ATM interface to an existing IP host

SYNOPSIS

atmRcspAdd *hostname maxvcid bw np addr num ip netmask*

DESCRIPTION

atmRcspAdd creates and attaches a new ATM interface to an existing IP host. The interface is made known to the **Ip** module on the host and an appropriate route is installed in the IP routing table.

OPTIONS

hostname
: Prefix of the host.

maxvcid
: Maximum number of VCIDs supported.

bw
: Bitrate of the outgoing line.

np
: Number of RCSP scheduling priority levels.

addr
: ATM-layer address of this end station.

num
: Interface number. The interface will appear to the IP layer with the name `atmd`*num*.

ip
: IP address to be assigned to this interface

netmask
: Netmask for the subnet.

COMPONENTS

hostname.**atm***num*
: Host adapter providing a frame transport interface (**HostAdapterRcsp**).

hostname.**atmd***num*
: Device driver object visible to IP (**AtmDeviceDriver**).

NOTES

The global variables `$ConfAtmDeviceDriverQosType`, `$ConfAtmDeviceDriverMuxType`, and `$ConfAtmDeviceDriverVcType` select the appropriate characteristics of the device driver (see **AtmDeviceDriver** for more details). They define the device driver behavior governing QOS, IP multiplexing, and virtual circuit usage. The default behavior results in best-effort SVCs with per-router pair multiplexing. These variables can be overridden by command-line settings, thus resulting in an easy way to vary the ATM adapters' behavior without modifying the configuration file.

FILES

`$PREFIX/lib/tcl/AtmHost.tcl`
: Location of the Tcl source code.

NAME

connect – Connect input/output ports of two switch-like objects

SYNOPSIS

connect *object1 port1 object2 port2 delay*

DESCRIPTION

connect creates a two-way connection with the specified *delay* between *port1* of *object1* and *port2* of *object2*. It assumes that the various switch ports (or whatever objects) have names of the form *O*.**input***N* or *O*.**output***N*, as appropriate, and that the output ports both support the **output** and **latency** subcommands (see **CellQueueFifo**(2) for more details).

OPTIONS

object1, object2
: Names of objects.

port1, port2
: Port numbers of objects to connect.

delay
: Latency of the connection. It is assumed that the delays in each direction are symmetric.

FILES

`$PREFIX/lib/tcl/util.tcl`
: Location of the Tcl source code.

NAME

HostAdapterFifo – Create an ATM host adapter

SYNOPSIS

HostAdapterFifo *name maxvcid bitrate addr*

DESCRIPTION

HostAdapterFifo creates and configures an ATM host adapter composite object.

OPTIONS

name Base name of the new host adapter object. This name is used as a prefix for all of the created objects.

maxvcid
Maximum number of VCIDs supported by the forwarding and signalling components.

bitrate Bitrate of the outgoing link.

addr ATM address of the host interface.

COMPONENTS

name.**gobackn**
Reliable retransmission entity for use by name.sighost signalling entity (**CellGoBackN**).

name.**input***x*
Input ports (**CellInputPort**).

name.**output***x*
Output ports (**CellQueueFifo**).

name.**sighost**
Signalling entity (**SigHost**).

name.**switch**
Cell switching entity (**SwitchModule**).

NOTES

The *name*.**switch** object has three ports. Port 0 is used by the **SigHost** signalling entity, port 1 is used for the host, and port 2 is used for the link to the network.

FILES

`$PREFIX/lib/tcl/HostAdapterFifo.tcl`
Location of Tcl source code.

NAME

HostAdapterRcsp – Create an ATM host adapter (RCSP queueing)

SYNOPSIS

HostAdapterRcsp *name maxvcid bitrate np addr*

DESCRIPTION

HostAdapterRcsp creates and configures an ATM host adapter composite object. It supports a variable number of priority levels (with delay bounds starting at sixteen cell times for the highest-priority level, doubling for successive lower-priority levels). The RCSP variable *Tick* is set for this queue to half the lowest delay bound (eight cell times in this case).

OPTIONS

name Base name of the new host adapter object. This name is used as a prefix for all of the created objects.

maxvcid
Maximum number of VCIDs supported by the forwarding and signalling components.

bitrate Bitrate of the outgoing link.

np Number of RCSP priority queueing levels.

addr ATM address of the host interface.

COMPONENTS

name.**gobackn**
Reliable retransmission entity for use by name.sighost signalling entity (**CellGoBackN**).

name.**input***x*
Input ports (**CellInputPort**).

name.**output***x*
Output ports (**CellQueueFifo** or **CellQueueRcsp**).

name.**sighost**
Signalling entity (**SigHostRcsp**).

name.**switch**
Cell switching entity (**SwitchModule**).

NOTES

The *name*.**switch** object has three ports. Port 0 is used by the **SigHostRcsp** signalling entity, port 1 is used for the host, and port 2 is used for the link to the network.

FILES

`$PREFIX/lib/tcl/HostAdapterRcsp.tcl`
Location of Tcl source code.

NAME

inet_addr – Parse dotted-quad IP address representation

SYNOPSIS

inet_addr *quad*

DESCRIPTION

inet_addr parses a dotted-quad representation of an IP address and returns an integer representation, convenient for manipulating using various Tcl logical operators. It is a direct counterpart to **inet_addr**(3) from the C library.

OPTIONS

quad Dotted-quad representation of an IP address (for example, `128.32.33.172`).

FILES

`$PREFIX/lib/tcl/LanHost.tcl`
Location of Tcl source code.

NAME

inet_ntoa – Create dotted-quad representation of an IP address

SYNOPSIS

inet_ntoa *address*

DESCRIPTION

inet_ntoa returns (as a string) the dotted-quad representation of an IP address as given by the integer *address*.

OPTIONS

address IP address, expressed as an integer.

FILES

`$PREFIX/lib/tcl/LanHost.tcl`
Location of Tcl source code.

NAME

lanAdd – Add a LAN interface to an existing host's IP module

SYNOPSIS

lanAdd *name lan number ip netmask*

DESCRIPTION

lanAdd is a procedure to attach an additional **LanDeviceDriver** to an existing host's **Ip** module. It can be used to make a multi-homed host (i.e. a router).

OPTIONS

name Name of the IP host (a prefix for all the objects that get created).

lan Name of **Lan** object to attach to.

number Interface number. The UNIX naming convention is to number interfaces of a given type in sequence beginning with 0; that convention is followed, but not enforced, by the various library routines.

ip IP address of the **Lan** interface, in dotted-quad notation.

netmask
IP netmask to use for the **Lan** subnet, in dotted-quad notation.

COMPONENTS

name.**lan***number*
Lan interface (**LanDeviceDriver**).

FILES

`$PREFIX/lib/tcl/LanHost.tcl`
Location of Tcl source code.

NAME

LanHost – Create an IP host, with a single network attachment

SYNOPSIS

LanHost *name lan ip netmask* [*gateway*]

DESCRIPTION

LanHost creates an IP stack which has a single network attachment to a (pre-existing) **Lan** object. The necessary network device drivers are created and appropriate routes loaded into the **Ip** routing tables. Instantiations of the transport layer protocols UDP and TCP are also created.

OPTIONS

name Name of the IP host (a prefix for all the objects that get created).

lan Name of **Lan** object to attach to.

ip IP address of the **Lan** interface, in dotted-quad notation.

netmask
IP netmask to use for the **Lan** subnet, in dotted-quad notation.

gateway
Gateway host for default route (optional). If specified, a default route is set with *gateway* as the next-hop router.

COMPONENTS

name.**ip** IP protocol object (**Ip**).

name.**lan0**
Lan interface (**LanDeviceDriver**).

name.**lo0**
Loopback interface (**LoopbackDeviceDriver**).

name.**tcp**
Local TCP entity (**Tcp**).

name.**udp**
Local UDP entity (**Udp**).

FILES

`$PREFIX/lib/tcl/LanHost.tcl`
Location of Tcl source code.

NAME

objects – Print state of all simulation objects

SYNOPSIS

objects

DESCRIPTION

The **objects** command prints out the state of every object in the simulation. by invoking every objects' **print** command. As this includes all primtive objects making up composite objects, the output of this command can be somewhat verbose.

FILES

`$PREFIX/lib/tcl/util.tcl`
Location of Tcl source code.

NAME

Site – Create an Internet site

SYNOPSIS

Site *name bw latency clients servers prefix*

DESCRIPTION

Site creates a new Internet site consisting of a single **Lan** and populates it with some number of servers and workstations (also referred to as clients). A router with a single interface onto the Lan is also created; the hosts assume that the router is the single gateway to the outside world.

OPTIONS

name Tcl prefix for all created objects at this site.

bw **Lan** bandwidth.

latency **Lan** latency.

clients Number of client machines to create on this subnet. Client machines have all network- and transport-layer objects instantiated, as well as a **TelnetServer** process to support remote logins.

servers Number of servers to create on this subnet. Servers are configured similarly to clients; in addition they also have **HttpServer** and **FtpServer** processes running.

prefix IP prefix for all machines at this site. The IP prefix must be a 24-bit prefix, expressed exactly in the form `128.32.131` (three octets, no trailing dot). This restriction is mandated by some difficulties in manipulating IP addresses within Tcl scripts; it may be lifted in the future.

COMPONENTS

name.**client***x*
Workstation client machine (**LanHost**).

name.**lan0**
Shared-medium local area network at this site (**Lan**).

name.**router0**
Gateway router to other networks (**LanHost**). Note that this router initially has only its `lan0` interface; additional interfaces must be created and attached using **atmFifoAdd**, **atmRcspAdd**, or **lanAdd** in order for the router to do anything useful.

name.**server***x*
Server host (**LanHost**).

NOTES

The following global variables contain the names and IP addresses of the router, servers, and workstations: *name*.**serverNames**, *name*.**serverAddresses**, *name*.**clientNames**, *name*.**clientAddresses**, *name*.**routerNames**, *name*.**routerAddresses**. These variables are primarily intended to be used by workload generators to find hosts for new user processes or targets for new network connections.

By using **subnetAdd**, larger (multi-LAN) sites may be constructed. Many configurations are possible.

FILES

`$PREFIX/lib/tcl/Site.tcl`
Location of the Tcl source code.

NAME

subnetAdd – Add a subnet to a site

SYNOPSIS

subnetAdd *name bw latency clients servers prefix router num*

DESCRIPTION

subnetAdd is a procedure that aids in the creation of large simulated Internet sites. After creating a **Site** object, additional subnets and hosts may be added to it to produce an Internet network larger than would fit on a single LAN.

OPTIONS

name Tcl prefix (same as the corresponding Site).

bw **Lan** bandwidth.

latency **Lan** latency.

clients Number of client machines to create on this subnet. Client machines have all network- and transport-layer objects instantiated, as well as a **TelnetServer** process to support remote logins.

servers Number of servers to create on this subnet. Servers are configured similarly to clients; in addition they also have **HttpServer** and **FtpServer** processes running.

prefix IP prefix for all machines at this site. The IP prefix must be a 24-bit prefix, expressed exactly in the form `128.32.131` (three octets, no trailing dot). This restriction is mandated by some difficulties in manipulating IP addresses within Tcl scripts; it may be lifted in the future.

router Name of router object (typically will be *name*.**router0**)

num Network ID, used to identify the different **Lan objects** at one site and also to number the **Lan** interfaces on the router.

COMPONENTS

name.**client***x*
Workstation client machine (**LanHost**).

name.**lan0**
Shared-medium local area network at this site (**Lan**).

name.**server***x*
Server host (**LanHost**).

NOTES

The global variables created by **Site** are all updated.

FILES

`$PREFIX/lib/tcl/site.tcl`
Location of the Tcl source code.

NAME

SwitchFifoN – Create ATM switch composite object

SYNOPSIS

SwitchFifoN *name numPorts maxVcids bw*

DESCRIPTION

SwitchFifoN creates a $NumPorts \times NumPorts$ output-queued switch (with FIFO queues). A **Sig** entity handles SVC setup and teardown by communicating with the **Sig** entities on adjacent switches (and **SigHost** entities on any directly-attached ATM hosts).

OPTIONS

name Name of the switch (a prefix for all created objects).

numPorts
Number of ports on this switch.

maxVcids
Maximum number of virtual circuits (and thus virtual circuit identifiers) supported by the switch translation module, the signalling entity, and the switch ports.

bw Default link bandwidth.

COMPONENTS

name.**gobackn**
Reliable retransmission module, used by *name*.**sig**.

name.**input***n*
Input port object for this switch. Can be connected to the output port of another ATM switch (or host) using the **connect** command. Note that “real” switch ports are numbered 1 through *numPorts* (port 0 is reserved for a connection to *name*.**sig**).

name.**output***n*
Output port object for this switch. Can be connected to the output port of another ATM switch (or host) using the **connect** command. Note that “real” switch ports are numbered 1 through *numPorts* (port 0 is reserved for a connection to *name*.**sig**).

name.**sig**
Signalling entity for this ATM switch.

FILES

`$PREFIX/lib/tcl/SwitchFifoN.tcl`
Location of Tcl source code.

NAME

SwitchRcspN – Create ATM switch composite object

SYNOPSIS

SwitchRcspN *name numPorts maxVcids bw np*

DESCRIPTION

SwitchRcspN creates a $NumPorts \times NumPorts$ output-queued switch (with RCSP queues). It supports a variable number of priority levels (with delay bounds starting at sixteen cell times for the highest-priority level, doubling for successive lower-priority levels). The RCSP variable *Tick* is set for this queue to half the lowest delay bound (eight cell times in this case). A **SigRcsp** entity handles SVC setup and teardown by communicating with the **SigRcsp** entities on adjacent switches (and **SigHostRcsp** entities on any directly-attached ATM hosts).

OPTIONS

name Name of the switch (a prefix for all created objects).

numPorts
Number of ports on this switch.

maxVcids
Maximum number of virtual circuits (and thus virtual circuit identifiers) supported by the switch translation module, the signalling entity, and the switch ports.

bw Default link bandwidth.

np Number of RCSP priority levels.

COMPONENTS

name.**gobackn**
Reliable retransmission module, used by *name*.**sig**.

name.**input***n*
Input port object for this switch. Can be connected to the output port of another ATM switch (or host) using the **connect** command. Note that "real" switch ports are numbered 1 through *numPorts* (port 0 is reserved for a connection to *name*.**sig**).

name.**output***n*
Output port object for this switch. Can be connected to the output port of another ATM switch (or host) using the **connect** command. Note that "real" switch ports are numbered 1 through *numPorts* (port 0 is reserved for a connection to *name*.**sig**).

name.**sig**
Signalling entity for this ATM switch.

FILES

`$PREFIX/lib/tcl/SwitchRcspN.tcl`
Location of Tcl source code.

NAME

cdf, pdf – File format for cumulative distribution data files

DESCRIPTION

INSANE relies on probability distributions derived from empirical network measurements. These data are distributed in **cdf** and **pdf** files, which are compiled into C++ arrays during compilation.

cdf files specify cumulative probability distributions. They are line-oriented files, consisting of lines of the form:

value number cdf

Where the fields in each lines have the following contents:

value Random variable value.

number Number of samples of the random variable.

cdf Cumulative distribution

The **pdf** probability density files have an identical file format, but have their corresponding C++ data structures generated differently. In some cases, some of the tcplib(3)-derived distributions did not have sufficient precision in their *cdf* fields. For these files, the *number* field is used to re-derive the cumulative distribution functions.

FILES

`$PREFIX/lib/*.cdf`

`$PREFIX/lib/*.pdf`

BUGS

The choice of the suffix `*.pdf` is unfortunate, as it has the potential to become confused with Portable Document Format files (which also use the `*.pdf`) suffix. But heck, probability theory has been around a lot longer than Adobe has.

NAME

insane.conf – INSANE configuration file

DESCRIPTION

insane.conf is the default configuration file for INSANE. It specifies the objects in a given simulation scenario, as well as their relationship to each other.

The configuration file is a script written in the Tcl [Ousterhout94] scripting language. All of the Tcl built-in primitives are available to the script, as well as the commands listed in sections 2 and 3 of the INSANE manual.

FILES

Sample configuration files are provided in the INSANE distribution.

NAME

events – INSANE event types

DESCRIPTION

EventNull

Null event. Should only be used for debugging; most objects will treat reception of an **EventNull** event as an error. No *data*.

EventCellArrival

Indicates arrival of an AtmCell at an object. *data* field contains a pointer to the AtmCell. Note that there is no **EventCellSend** message.

EventCellTransmitComplete

Indicates that an ATM cell has finished transmission and that the queue object controlling transmission should select the next cell to transmit. No *data*.

EventFrameSend

Posted by device drivers to lower-layer objects to indicate a datalink-layer frame ready to transmit. *data* is a pointer to the frame.

EventFrameArrival

Posted by device drivers to network layer (i.e. **Ip**) objects to signal reception of a datalink-layer frame. *data* is a pointer to the frame.

EventTimer

Generic timer event. In general, the *data* field is ignored, but it may be used to differentiate between different types of timers.

EventTimer2

Secondary generic event timer. Principally used by the **Tcp** class for its slow (500 ms) timer (the 200 ms fast timer uses **EventTimer**). No *data*.

EventCall

Signalling call to and from **SigHost** clients. These events are used both for downcalls into the signalling systems and returns from those downcalls.

EventHalt

This event type is only used by **Simulation** class objects. Upon reception of this event, the **Simulation** object causes the simulator to halt and begin end-of-run processing. No *data*.

EventDebug

This event type is only used by **Simulation** class objects. Upon reception of this event, the **Simulation** object causes the simulator to enter the Tcl command line interface (CLI), allowing the user to interact with the simulator. No *data*.

EventPacketArrival

Indicates arrival of a packet passed from a device driver object to an **Ip** object. *data* points to the inbound IP packet.

EventPacketSend

Posted by **Ip** objects to device drivers. *data* is a pointer to the outbound IP packet.

EventIpArrival
Posted by **Ip** modules to transport layer objects to indicate an inbound IP packet. *data* is a pointer to the complete IP packet.

EventIpSend
Posted by transport layer objects to **Ip** modules to indicate a packet to send. *data* is a pointer to a complete IP packet.

EventUdpArrival
Posted by **Udp** modules to application programs to indicate that a UDP packet just arrived. *data* is a pointer to a call block.

EventUdpSend
Posted by applications to **Udp** modules to indicate a packet ready to send. *data* is a pointer to a UDP call block.

EventTcpArrival
Posted by **Tcp** modules to application programs to indicate successful reception of TCP data. *data* is a pointer to a `TcpCall` call block.

EventTcpSend
Posted by applications to **Tcp** modules to indicate TCP data ready to send. *data* is a pointer to a `TcpCall` call block.

EventTcpConnect
Posted by **Tcp** modules to applications to indicate a successful TCP connection. *data* is a pointer to the TcpPcb structure for the connection in question. For an active open (client application), this pointer should be equal to that returned by `Tcp::UserConnect()`. For server-type applications doing a passive open, this pointer will point to a new TcpPcb structure. Note that servers are not told which `Tcp::UserListen()` corresponds to the newly-opened connection, although they can examine the port numbers (and thus deduce the corresponding `Tcp::UserListen()` call) by looking in the `TcpPcb` structure..

EventTcpClosed
Posted by **Tcp** modules to applications to indicate that a TCP connection has been closed. The `TcpPcb` has already been deallocated at the time this event is posted; applications should not rely on its contents. *data* is a pointer to the `TcpPcb` structure for the connection in question.

EventTcpError
Posted by **Tcp** modules to applications to indicate an error with a TCP connection. The `TcpPcb` has already been deallocated at the time this event is posted; applications should not rely on its contents. *data* is a pointer to the `TcpPcb` structure for the connection in question.

4 Availability

INSANE is not (yet) generally available. The author has been known to make pre-release distributions of the source code, with the provision that they not be distributed and that no support can be provided.

Contact `bmah@CS.Berkeley.EDU` for details.

5 References

[Danzig91] Peter B. Danzig and Sugih Jamin. "tcplib: A library of TCP internetwork traffic characteristics." Technical Report USC-CS-91-495, Computer Science Department, University of Southern California, Los Angeles, CA, 1991.

[Mah93] Bruce A. Mah. "A mechanism for the administration of real-time channels." Masters report, University of California at Berkeley, April 1993.

[Ousterhout94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[Postel80] Jon Postel. *User Datagram Protocol*. Internet Request for Comments 768, August 1980.

[Postel81a] Jon Postel. *Internet Protocol*. Internet Request for Comments 791, September 1981.

[Postel81b] Jon Postel. *Transmission Control Protocol*. Internet Request for Comments 793, September 1981.

[Stevens94] W. Richard Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[Stroustrup91] Bjarne Stroustrup. *The C++ Programming Langauge*. Addison-Wesley Publishing Company, 1991.

[Wright95] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume 2*. Addison-Wesley Publishing Company, 1995.

[Zhang93] Hui Zhang and Domenico Ferrari. "Rate-controlled static-priority queueing." In *Proceedings of IEEE INFOCOM '93*, San Francisco, CA, 1993.

[Zhang94] Hui Zhang and Domenico Ferrari. "Improving utilization in deterministic service in multimedia communication." In *Proceedings of the 1994 IEEE International Conference on Multimedia Computing and Systems*, pages 295–304, Boston, MA, May 1994.