NAME

troll − network simulator / datagram forwarder

SYNOPSIS

troll [*options*] **port**
#include <sys/types.h>
#include <netinet/in.h>

DESCRIPTION

Troll is an application designed to simulate the various quirks and vagaries of a network environment. The troll forwards UDP datagrams between processes, possibly dropping, delaying, garbling, and/or duplicating those datagrams.

COMMAND LINE ARGUMENTS

port The UDP port number on which the troll will listen for incoming datagrams. This argument is mandatory, and must be an integer between 1024 and 65535.

−S*TCPend1*
The IP address of one end of the UDP connection.

−C*TCPend2*
The IP address of other end of the UDP connection.

−a*port1* The UDP port on which TCPend1 sends/receives datagrams.

−b*port2* The UDP port on which TCPend2 sends/receives datagrams.

The following options override various default parameters. They can also be set by interactive commands, as described below. Options specified as *percent* must be an integer in the range from 0 through 100.

−f *file* The distribution of datagram delays is to be taken from the named file rather than computed from the *delay* parameter. See the description of the **delay file** interactive command below.

−c*number* The capacity of the simulated network, in datagrams (default is 16). If this capacity is exceeded, newly arriving datagrams will be discarded immediately.

−g*percent* The percentage of datagrams that should be garbled (default is 0%).

−l*number* The maximum datagram lifetime, in milliseconds. If this value is specified as zero (the default), no maximum is enforced. Otherwise, datagrams are discarded on *output* from the troll if they have been delayed too long.

−m*percent*
The percentage of datagrams that should be duplicated (sent multiple times; default is 0%).

−r Prevent reordering. With this option, delays are all interpreted as relative: Datagrams are queued first-in-first out, with delays between datagrams calculated according to the delay parameter (see *−s*, *−se*, and *−f*). Without the *−r* option, delays are absolute. Since each datagram is independently delayed, datagrams may be sent out in a different order than they arrived. See further discussion under **TECHNICAL DETAILS** below.

–s*number*

–se*number*
The separation of datagrams in milliseconds (default is 100; see also *–f*). If **e** is specified, the delay is an exponential random variable with mean *number*. Otherwise, it is a constant.

–x*percent* The percentage of datagrams that should be dropped (default is 10%).

INTERACTIVE MODE

After being started, the troll normally runs in an interactive mode, and will prompt for commands with the message

troll>

A command may be abbreviated to any unambiguous prefix of the name specified here.

capcity *number*
Sets the capacity of the network, in datagrams. If this capacity is exceeded, newly arriving datagrams will be discarded immediately.

drop *percent* Sets the percentage of datagrams that will be dropped (i.e., will not be forwarded).

maxlifetime *number*
Specifies a maximum datagram lifetime, in milliseconds. Just before sending a datagram, the troll checks whether it has been holding on to it for too long, and if so, drops it rather than forwarding it. If this parameter is set to zero (the default), no maximum lifetime is enforced.
Note that the troll can buffer only a limited number of datagrams, as specified by the **capacity** parameter; if a datagram arrives and the capacity is exceeded, the datagram is dropped. Datagrams dropped as specified by the **drop** parameter do not consume any resources, but datagrams that exceed the **maxlifetime** parameter occupy space until they are discared on output. Setting **delay** or **duplicate** to a sufficiently high value may result datagrams being dropped even if the **drop** and **maxlifetime** parameters are zero.

delay *milliseconds*

delay exponential *mean*

delay file *filename*
These commands control the **delay** distribution. Depending on the setting of the **reorder** option, the **delay** parameter controls the delay of datagrams or separation between them. The first form sets **delay** to a specific value. The second sets it to an exponentially distributed random variable with the indicated mean. (Other distributions may be provided by future releases of the troll). The third form sets it to a random variable with a distribution indicated by the histogram contained in the named file. The first line of the file should contain an integer indicating the precision, in milliseconds. The remainder of the file should have have one line for each possible delay value (in multiples of the precision). The length of each line (not including the newline character at its end) controls the relative frequency of corresponding delays. (The actual contents of the lines are ignored). For example, if file contains 100 <two empty lines> xx xxxxxx xxx x x then 2/13 of the datagrams will be delayed 200ms, 6/13 will be delayed 300ms, 3/13 400ms, 1/13 500ms, 1/13 600ms, and no datagrams will be delayed less than 200 or more than 600 milliseconds. As a special case, a file with only one non-empty line after the first will generate a deterministic distribution (all datagrams delayed the same amount of time).
Warning: Although delays are specified in milliseconds, the Unix timing facilities are usually less precise, so delays may actually be longer than specified.

reorder Toggles the **reorder** option (initially *on* unless cleared by the **–r** command-line option). If **reorder** is off, the **delay** distribution is used to compute the delay *between* datagrams. If **reorder** is on, the **delay** distribution is used to compute the *total* delay for each datagram (or duplicate). If the **reorder** option is on and **delay** is not a constant, datagrams may not be forwarded in the same order they were received.

garble *percent*
Sets the percent of forwarded datagrams that will be garbled (i.e., have one or more bits changed).

duplicate *percent*
Sets the percent of forwarded datagrams that will be duplicated (i.e., transmitted more than once); of those, *percent* will be duplicated again, etc. Each duplicate of a particular datagram is independently delayed and garbled.

source *filename*
Reads "interactive" commands from. *filename.*

The remaining commands do not take any argument.

parameters Prints the current values of the reliability parameters.

statistics Prints statistics about the number of datagrams dropped, garbled, duplicated, etc. These statistics are also printed on termination.

distribution Prints a histogram of datagram delays.

clearstats Clears all statistics to zero (the startup value).

quit Causes the troll to print statistics and terminate. The same effect is produced by an interactive "command" of control-D or by a keyboard interrupt (or kill -INT).

trace Toggles the **trace** option. When tracing is enabled, the **troll** prints a message starting with '<' on each datagram arrival, and a message starting with '>' on each datagram transmission.

EXAMPLE

Suppose you want to send traffic from **A (IP :128.105.112.101, port : 5201)** to **B (IP :128.105.112.102, port :5202)** through **troll** software running at **C (IP :128.105.181.31, port :5200)** ,then the following commands should be executed to setup troll with default parameters:
On C execute
troll -S 128.105.112.101 -C 128.105.112.102 -a 5201 -b 5202 5200
On A execute
tcpend -p 5201 -s 128.105.181.31 -a 5200 -f filename -m mtu -c sws
On B execute
tcpend -p 5202 -m mtu -c sws
Here the filename,mtu and sws refer to trhe parameters fropm programming assignment 3

TECHNICAL DETAILS

Whenever a datagram arrives, the troll examines the header to see whether the sockaddr there meets minimal sanity checks (it requires that ntohs(header.sin_family) is AF_INET and that ntohs(header.sin_port) is in the range 1024..65535; it does not check whether header.sin_addr is reasonable). If the check fails, or the number of queued datagrams exceeds the current setting of the **capacity** parameter, the troll increments a counter and drops the datagram.

Next, the troll flips a coin, and with probability **drop** it throws away the datagram (incrementing a counter). Otherwise, it flips another coin and with probability **garble** garbles the datagram. The garbling algorithm chooses 5 to 10 bytes of the datagram and xor's them with a random pattern. The first garbled byte is one of the first 36 bytes of the datagram; the remaining bytes are chosen randomly from anywhere in the datagram.

Next, the troll calculates when to transmit the datagram and schedules a timer event to send it. A random value is computed according to the current **delay** settings. If the **reorder** parameter is *on*, the dispatch time is calculated by adding the random value to the to the current time. Otherwise, it is calculated to adding the random value to the latest dispatch time any datagram currently queued for delivery. The timer queue is maintained such that requests to send two or more datagrams at the same time are satisified first-come-firstserved.

After the troll queues a datagram for subsequent transmission, it flips a coin and with probability **duplicate** makes a copy of the datagram. The copy is treated as an independent arrival and duplicated, delayed, garbled, etc. as described above.

SURGEON GENERAL'S WARNING

Since UDP does no flow control, datagrams will be dropped if you send them to the troll too fast, or if you don't receive them fast enough from the troll. Thus datagrams may be dropped even if the troll parameters are set for 'perfect reliability' (all probabilities 0).

BUGS

There should be more convenient ways to specify other distributions of delay (such as normal or hyperexponential) and to specify distributions for other random variables. There should be some way to make the delay depend in complicated ways on the number of datagrams currently queued, to simulate congestion in the network.

AUTHORS

The troll was originally written by Mitchell Tasman. It has been extensively modified and enhanced by Marvin Solomon.

COPYRIGHT