Traffic Generator Software Release Notes

Paul E. McKenney

Dan Y. Lee

Barbara A. Denny

SRI International and USC/ISI Postel Center for Experimental

Networking

January 8, 2002

1 INTRODUCTION

This document describes a packet Traffic Generator (TG) program that can be used to characterize the performance of packet-switched network communication protocols. The TG program generates and receives one-way packet traffic streams transmitted from the UNIX* user level process between traffic source and traffic sink nodes in a network. Different protocols, or versions of the same protocol, may be tested to ascertain differences in performance.

The TG program is controlled by a simple, but flexible, specification language that allows access to different operating modes, protocols, addressing functions, and experimentation with traffic parameters. The specification language allows traffic of several different packet lengths and interarrival time distributions to be generated. In the current implementation, the TCP and UDP transport protocols, with unicast and multicast addressing (UDP only), are supported. The scope of a multicast flow can be controlled by setting the time-to-live (TTL) field. Quality/Type of Service parameters include setting the DiffServ field† and controlling the send and receive buffers.

While the name "Traffic Generator" focuses on the packet generator aspect of the program, the TG program also serves as a traffic sink. In order to record packet transmission statistics, a TG configured as a traffic sink must be active to receive the test traffic. The received traffic stream is logged in a file by the traffic sink for post-test analysis. When connection-oriented transport services are used, a traffic sink is needed to accept an incoming connection request. The TG serving as the traffic source always logs datagram transmit times. This mode of operation may be useful for analyzing network blocking characteristics or for loading a network.

Depending upon whether connection-oriented or connectionless transport services are used, one or more traffic sinks may be needed. For datagram protocols like UDP, multiple traffic sources may send their traffic to the same sink process (a many-to-one relationship). The TG serving as a traffic sink logs all received datagrams.

---

*UNIX is a registered trademark of UNIX System Laboratories, Inc.

†The DiffServ field is set by calling **`setsockopt`** for TOS. By choosing an appropriate value, you can also set the other bits not used by DiffServ. In the future this may change as operating systems get updated.

When TCP protocols are used, there must be a one-to-one mapping between a traffic source and a traffic sink for each connection. Each invocation of the TG program is able to sustain a single stream; the TG is currently unable to service more than one active connection at a time. Consequently, if two connections are desired between a pair of nodes, two traffic sinks are required.

2 USER GUIDE

This section describes how to use the TG program. We describe the function of the TG program and the syntax of its command specification language.

Note that keywords and names of programs are shown in **`typewriter`** font, while parameters (values that you supply) are shown in *italic* font. Optional parameters are enclosed in brackets [ ]: these parameters may be safely omitted. During our discussion of the syntax of TG specification and log files, angled brackets < > are used to delimit metavariables.

In our discussion, all numeric parameters can be entered as decimal, octal, hexadecimal integer, or floating-point numbers. In general, a number may be an integer (1), a decimal (1.0), or a floating-point number (1E-3). Octal (010) and hexadecimal (0x10) numbers are also supported, although they are not too useful for our application.

All time values supplied to the TG program are expressed as hours:minutes:seconds (H:M:S) separated only by colons and without intervening white spaces. These time fields are not limited, so you can, if you desire, work only in seconds or minutes.

If the H:M:S time format is not followed, the fundamental unit of measure for time defaults to units of “seconds,” or fractions thereof. Note that the unit of measure for the packet size random variables is bytes.

Internally, the TG deals with most parameters as double precision floating-point values, so that even very small values are represented correctly. The precision of a number will be preserved, in spite of the potential inability of a system to generate packets at the specified resolution.

2.1 SOFTWARE DISTRIBUTION HIERARCHY

The software distribution hierarchy consists of several top-level directories, which follow common UNIX conventions. Three programs form the TG distribution: the first is the traffic generator (**`tg`**), the second is a filter (**`dcat`**) for reading TG log files, and the third is a perl script (**`gengraph.pl`**) that generates information used by graphing tools (**`Xplot`**, **`Gnuplot`**, and **`Xgraph`** are the currently supported graphing tools).

The directories and their contents are as follows.

- **`./bin`**—Executable versions of Solaris*, SunOS†, FreeBSD and Linux‡binaries, and the perl script for graphing TG data
- **`./docs`**—Documents and other notes
- **`./examples`**—Sample TG specification files**
- **`./src`**—Source code for the TG distribution.

2.2 BUILDING THE TG PROGRAM

The TG is released with source code so that enhancements can be added to support new protocols or additional statistical distributions that may be needed for future experimentation. Note that the TG distribution includes compiled binaries in the **`./bin`** directory.

*Solaris is a trademark of Sun Microsystems, Inc.
†SunOS is a trademark of Sun Microsystems, Inc.
‡Linux RedHat distribution 7.1 and 7.2 have been tested.
**TG specification files are also referred to as scripts.

To rebuild the traffic generator, change to the **./src** directory and execute the **make** command. Your system will also need the **lex** and **yacc** programs to rebuild the TG program. If FreeBSD is used, flex can be used in place of lex. Note that the software has been compiled and tested only on the SunOS, Solaris, FreeBSD, and Linux operating systems; however, the software should work on other UNIX-based operating systems.

2.3 INVOKING THE TG

The syntax for invoking the TG program is:

tg [-f] [-i *input-file* **] [-o** *output-file* **]**

By default, the TG program reads the standard input (stdin) device for a TG specification file and writes its log file to the standard output device (stdout). The traffic generator accepts the **-i** and **-o** options, which override the default input/output behavior. The **-i** option specifies an input file that contains TG test specification entries. The destination of the log file can be changed with the **-o** option. The **-f** option specifies that the buffer be flushed to the log file after every write.

Alternatively, specification and log files may be stored on disk by using the I/O redirection features present in many command interpreter shells. For example, the following command in the UNIX **csh** redirects the contents of *input-file* into the traffic generator program, and writes the log file to *output-file*.

tg < *input-file* **>** *output-file*

2.4 TG SPECIFICATION FILE

A specification file consists of commands in the following forms, and in the order specified:

< start-time >
< *association-spec* >
< tg-action-list >

The *start-time* clause indicates the time that a test run is supposed to start, relative to the time the clause is parsed. Upon satisfying the *start-time* clause, the actions listed in the *association-spec* and *tg-action-list* clauses are executed. Before test traffic can be generated, an *association-spec* entry must be provided to specify an association, or binding, between a traffic source and a traffic

sink. The *association-spec* also provides the ancillary test parameters used in a test session. The *tg-action-list* is a list of *tg-action* clauses; each clause specifies an action to be performed, such as causing the TG program to generate test traffic with specified distributions.

A file inclusion capability is supported. A line of the form

include *"filename"*

causes the named file to be read and parsed. Double quotes must enclose the filename. The maximum depth of inclusion is precompiled into the program and can be changed by redefining **MAX_INCLUDE_LEVEL**. A specification file, therefore, contains a complete set of commands—i.e., *start-time*, *association-spec*, and *tg-action-list*—or an include file followed by zero or more *tg-action* clauses.

2.4.1 Starting Time

The experiment starting time is specified relative to the current time as an offset value. The offset value is in modulus-time form, where the supplied parameter specifies the next time boundary that an action will take place. The syntax for the start time is

on *H:M:S*

During an invocation of the TG, the **on** clause can appear at most once in a specification file.

This scheme is relatively independent of local time. The advantage of the scheme is scripts can be rerun without modification. It also allows scripts to synchronize to within a few milliseconds of each other, despite having been started several seconds out of synchronization (e.g., due to network delay to geographically remote hosts, or "telephone slew" between human operators). The quality of the synchronization is limited only by the ability of the time protocol (e.g., Network Time Protocol) to synchronize time, and by operating-system factors such as scheduling granularity and delays.

For example:

```
on 15
```

starts at the next fifteen-second interval (`x:00, x:15, x:30, or x:45`), while

```
on 1:00
```

starts at the beginning of the next minute.

For the very patient:

```
on 1:00:00
```

starts at the beginning of the next hour.

After the **on** clause has been satisfied, the *association-spec* and *tg-action-list* in the following lines are executed. Note that the **on** clause is designed to provide session-level synchronization between the TG processes, and does not cause traffic to be generated. Traffic is not generated until a *tg-action* clause has been parsed, as discussed in Subsection 2.4.3.

2.4.2 Association Specification

An *association-spec* is a binding between a traffic source and a traffic sink. An *association-spec* may take one of the following two forms:

protocol local-address **server [** *quality-of-service* **]**

or

protocol remote-address **[** *quality-of-service* **]**

Traffic Source/Sink Modes. The TG program can serve both as a traffic source and a traffic sink. The word **server** in the first form of the *association-spec* clause is a keyword that indicates that the traffic generator should act as a traffic sink. Once in the **server** mode, the TG program will wait indefinitely for incoming connections or traffic and will act upon the received messages as appropriate.

When the **server** keyword is used, the *local-address* field specifies the local address where TG will accept traffic. The address must be for the local host, as the TG binds to the named port. Note that the IP address may be specified as `0.0.0.0`.

Protocols. The *protocol* argument is a string (such as **`tcp or udp`**) that selects the appropriate transport-layer protocol to carry the test traffic. The current implementation supports only TCP and UDP protocols; however, the internal protocol table may easily be expanded to accommodate new protocols as they are developed.

Addressing. The current implementation does not resolve host names. All addresses must be supplied in the standard Internet dotted address form (e.g., 128.18.4.100.1234). Note that the port number is appended to the address. For example, the address `128.18.4.100.1234` specifies a rendezvous point of port `1234` at the host with the address `128.18.4.100.`

Traffic streams are multiplexed by protocol port addresses. In many UNIX implementations, port numbers less than 1024 are privileged and can only be accessed by programs with root access privileges. It is recommended that port numbers between 1024 and 2^{16} - 1 be used. If a port is already being used, select a different port number and reexecute the program.

The *remote-address* field selects the destination endpoint of the association; e.g., the destination of a datagram or a peer in a connection. The address can be an IP multicast address. Note that addresses used in the association specification include the port number. The example below sets up the TG program so that subsequent UDP transmissions are sent to the process running on `128.18.4.29` and listening at port `2345.`

```
udp 128.18.4.29.2345
```

Quality of Service. The **`quality-of-service`** (QOS) field includes the following entries, specified in any order:

`average bandwidth` *number*
`peak bandwidth` *number*
`average delay` *number*
`peak delay` *number*
`average loss` *number*
`peak loss` *number*
`interval` *number*
`mtu` *number*
`rcvwin` *number*

sndwin *number*
tos *number*
ttl *number*

These options were designed to serve as subscription/connection setup profiles. In the current TCP and UDP protocol implementation, these QOS options may be omitted. Eventually, these options may be used or ignored as the protocol's connection setup function sees fit. Currently, **rcvwin** and **sndwin** control the number of bytes in the receive and send window of TCP and UDP. **TTL** controls the scoping of the multicast traffic. **TOS** sets the IP ToS field: now known as the Differentiated Services field plus bits 6 and 7. All other options are ignored, though the numbers are parsed and inserted in the protocol data structure.

2.4.3 TG Action List

Traffic is generated via *tg-action-list* clauses. The *tg-action-list* is a list of *tg-action* elements, each of which consists of

[at *time-literal* **]** *tg-action*

The optional **at** clause specifies a time relative to the start time that the associated *tg-action* will execute. The *time-literal* is a time specified in the H:M:S colon format. If the **at** clause is omitted, the action will commence upon completion of the previous action.

A *tg-action* consists of either

setup

by itself or

wait [*time-literal* **]**

by itself or

log *datetime-format*

by itself or

arrival *distribution* **length** *distribution*

in that order. These entries may be followed by a list of any or all of the following, specified in any order:

data *number*
packet *number*
time *time-literal*
seed *number*

The **setup** clause forces an association setup to occur. If no **setup** clause is present, an implicit setup will occur at the time specified by the **on** start-time clause. It is illegal to have more than one setup clause; if a setup clause is present, it must precede all *tg-action* clauses that generate traffic.

The **wait** clause causes the traffic generator to pause, as specified by the time-literal argument, or by the succeeding **at** clause (or forever, if there is no time-literal argument and this is the last *tg-action*; this is useful for servers). The **log** clause opens a new log file. The *datetime-format* is the desired strftime format specification* (for example, **log "%D"**). The new filename is

output-file.<**strftimeresult**>

where *output-file* is the filename specified when tg was invoked and <**strftimeresult>** is the string returned from **strftime** with all forward slashes replaced by underscores. If the name of the *output-file* contains a suffix, the suffix is appended after the string from **strftime**.

The **arrival** clause specifies the interarrival time distribution, and the **length** clause specifies the packet length distribution.. The **data** clause limits the total amount of data to be sent in bytes (mimicking, for example, a file transfer operation). Similarly, the **packet** clause specifies the number of packets to send while the **time** clause limits the total amount of time spent transmitting. The **seed** clause sets the random-number generator seed (before generating random variables for the arrival and length clauses).

Where possible, the program attempts to deduce implicit time clauses and **at** clauses from those of surrounding *tg-action-list* clauses. In the following example, deduction is not possible:

```
arrival exponential 0.030 length 120
data 1000000
arrival exponential 0.060 length 240
```

*see the **strftime** man page for format specifications

The program cannot know in advance how long it will take to send one million bytes of data through the network.

2.4.4 Statistical Distributions

Statistical distributions can be associated with packet interarrival and packet length random variables. The interarrival distribution is specified with the keyword **`arrival`**, while the packet length distribution is preceded by the keyword **`length`**. Currently, four types of distributions are supported—additional distributions are planned and will be added when implemented. Four keywords are used to specify the distribution that can be used with a *tg-action* clause.

`constant` *value* or *value*
`uniform` *max* or **`uniform`** *min max*
`exponential` *mean* or **`exponential`** *mean min max*
`markov2` *number distribution number distribution*

The **`constant`** distribution always returns the number specified in the supplied parameter; and if desired, the **`constant`** keyword may be omitted entirely. The **`uniform`** distribution requires a number specifying the maximum value of the open interval [0, *max*) from which the random number is drawn. If the left endpoint is also specified, a random number is selected from the open interval [*min*, *max*). The **`exponential`** distribution is the classic distribution with the specified mean. As with the **`uniform`** distribution, it is possible to restrict the values that are returned from the exponential distribution to only those within the open interval [*min*, *max*); the exponential distribution is unchanged otherwise. Specifying an upper limit for the exponential distribution will force only reasonably sized packets to be issued; otherwise there is a small probability that an infinite size packet may be generated and will result in an error being reported. The **`markov2`** keyword specifies a two-state markov distribution: the first number gives the mean time in state 1; the second number gives the mean time in state 2. Each state is associated with its own distribution; these distributions could themselves be markov2 distributions, if desired. As shown above, the italicized parameter *distribution* is a placeholder for a keyword from the set: **`constant, uniform, exponential, and markov2`**.

2.5 EXAMPLES

The following simple example illustrates the specification entries for a TCP traffic source.

```
on 0:15
tcp 128.18.4.97.2345
at 5 setup
at 6 arrival exponential 0.1 length exponential 576
seed 321423 time 10
```

The **`on 0:15`** command instructs the TG program to wait until the next 15-second time boundary approaches before initiating any protocol activity. Since the keyword **`server`** is absent in the *association-spec* line, TG will initiate a TCP traffic connection to the node address `128.18.4.97, port 2345`. The **`setup`** command, which is activated 5 seconds after synchronizing on the closest 15-second boundary, initiates a connection request to the traffic sink node. One second later, traffic will be sent across the connection with an exponential interarrival distribution with a mean of 0.1 seconds. The TCP segment size is also exponentially distributed with a mean of 576 bytes. The last line sets the random number seed and limits the test duration to 10 seconds.

A TG must serve as a traffic sink, before TCP traffic can be exchanged with the traffic source. The specification entries for a traffic sink are simpler, and are as follows:

```
on 0:15 tcp 128.18.4.97.2345 server
at 1.1 wait
```

Note that the **`server`** keyword is present, and that the TG enters into the TG **`wait`** state, 1.1 seconds after the start time synchronization boundary.

Other examples of TG scripts can be found in the **`./examples`** directory supplied with the software distribution.

3 LOG FILE

The TG program records all notable events in a log file for post-test analysis. With the exception of the header descriptor, the TG log file is encoded in binary form to conserve storage space. Some binary fields are encoded with a variable-length code for additional space savings. Such fields are stored as a sequence of bytes: seven bits per byte in little-endian format. The sign bit is set to one, to indicate that the field continues into the next byte, or cleared to zero to indicate that this is the last byte forming a number. Note that the value of the number is unaffected if the last byte is 0x00. This allows a trailing 0x00 to be used to flag the number as being special in some way.

In general, the supplied **`dcat`** program should be used to expand a TG log file into readable text form. The **`dcat`** filter is distributed with the release in the **`./bin`** and **`./src/dcat`** directories. The syntax for invoking the **`dcat`** program is:

`dcat[-a]`

The **`-a`** option indicates the timestamps should be reported as absolute rather than relative. This allows time-correlation of data from multiple log files.

3.1 LOG FILE HEADER

An ASCII header prefaces each log file and provides general information about the test session that generated the log data. The information stored in the header of the log file includes the following:

- Header file version
- Version of the TG program used
- User who created the log file and the machine name where TG was executed
- Machine configuration
- Name of the TG specification file
- Start time of the test

- Name of the protocol under test
- Address family indicator.

The ASCII header portion of the log file may be viewed directly by using the UNIX filter **head** to extract the first several lines of the file. The ASCII header is delimited by the strings <Begin TG Header> and <End TG Header>. Note that the first character on the next new line, following the ASCII header, contains a binary TG log record.

3.2 LOG FILE RECORD FORMAT

The **dcat** program converts TG log records into a readable ASCII format and prints its output to stdout. The output from the **dcat** program can be fed into **awk, perl,** or other similar string processing programs. **Gengraph.pl** is provided as an example (See “Visualization Support” on page 16.) **Dcat** output lines have the following format:

<Event Timestamp> <Event Type> <Address> <Event Data>

An example of the output from the **dcat** program is shown below.

Table 1

Time	Type	Address	ID	Length
0.003678	Setup			
10.061812	Accept	128.18.6.90.1626	Association	4
11.041100	Transmit	128.18.6.90.2346	0	1460
11.041244	Receive	128.18.6.90.1626	0	1460
16.006682	Teardown			

Event Timestamp. The event timestamp field is printed as the number of seconds since the start of the test run; that is, the time since the **on** clause was satisfied unless the **[-a]** option was given. Note that the event timestamp is not specified relative to the **at** clause. Although the timestamp field is printed with microsecond resolution, the actual precision of the timestamp depends upon the granularity returned by the **gettimeofday** system call.

Event Types. TG event types include the majority of the common events that can be observed from the UNIX user-process level. The event types include the following:

- **`Receive`**—indicates a packet was received.
- **`Transmit`**—indicates a packet has been transmitted.
- **`Setup`**—indicates protocol initialization is complete.
- **`Accept`**—indicates a connection has been established.
- **`Teardown`**—indicates a connection has been torn down.
- **`Error`**—indicates an error in the program was detected.

Addresses. The address field specifies the TG peer that generated the event. Depending upon the event type, the address may indicate the source of a received packet or the destination of a transmitted packet. Addresses take the form of an Internet address with a port number appended to it.

Event Data. Event data are present only for certain event types. For **`Receive`** and **`Transmit`** events, a datagram/segment identification number is printed. The length of the received datagram/segment is also printed. For **`Accept`** events, the address of the connection peer is printed, together with the association number. For **`Setup`** and **`Teardown`** events, no other data is provided.

An identification number is used to reference a datagram or a segment within a stream. For stream protocols, the identification field specifies the position of the first received byte in the stream: consequently, the identification field added to the length field gives the total number of bytes transferred at the event time.

An **Error** event type is also provided to record errors as they occur during the execution of the program. Depending upon when the error takes place, an address may or may not be recorded in the log file. The error codes are described in **log.h** and are reprinted below.

Table 2

Table 3

```
#define LOGERR_INTFMT       1   /* Script format error */
#define LOGERR_MEM          2   /* Out of memory */
#define LOGERR_2SETUP       3   /* Two connections were established */
#define LOGERR_GETTIME      4   /* gettimeofday() failed */
#define LOGERR_SELECT       5   /* select() failed */
#define LOGERR_FCNTL        6   /* fcntl() failed */
#define LOGERR_GETPEER      7   /* getpeername() failed */
```

3.3 INTERNAL LOG FILE FORMAT

For those wishing to write their own filters or data manipulation programs in order to process a log file directly, the log file is organized as a sequence of records, with each record having as a minimum the record type, control, and timestamp fields. The form of the binary log file is similar to the tables printed by the **dcat** program. The source code in the **dcat** filter should provide a good starting point. The format is as follows.

```
< Receive > < Control > < Time > < Address > < Id > < Size > [ Errno ]
< Transmit > < Control > < Time > < Address > < Id > < Size > [ Errno ]
< Setup > < Control> < Time > [ Errno ]
< Teardown > < Control> < Time > [ Errno ]
< Accept > < Control > < Time > <Address > < Association > [ Errno ]
< Error > < Control > < Time > < Address > < Error Type >
```

Consult the **tg** and **dcat** source code for the elaboration of each field. As detailed above, the [**Errno**] field holds the UNIX error file number and will very likely not be present in most records.

4 Visualization Support

The TG distribution includes a perl script called **gengraph.pl** in the **./bin** directory. This script takes the TG binary log files and produces information suitable for viewing by either **xgraph**, **xplot**, or **gnuplot. Dcat** is used as part of this process and must be in the user's path. The syntax for invoking this script is:

gengraph.pl [**-h** | **--help]** [**-c** *client-file*] [**-s** *server-file*] [**-o** *output-file*]

[**-f xplot** | **xgraph** | **gnuplot**] [**-SQ**] [**-DR**] [**-S**] [**-R**] [**-IS**] [**-IR**] [**-D**]

[-AD][**-J**] [**-AJ**] [**-L**]

The *client-file* is the name of the binary log file produced by the client script while *server-file* is the name of the binary log file produced by the server script. If the **-c** and **-s** option are not specified the default filenames are **client.log** and **server.log**, respectively. The *output-file* is the name of the file produced by **gengraph.pl** and is used by the appropriate graphing tool.* The default filename is **stats.xpl**. The graphing tool is specified by the **-f** option. If no graphing tool is specified, the default tool is **xplot**. The options following the graphing specification indicate the data the user is interested in graphing. The kinds of information, which can be graphed, include the following:

- **-SQ** Sequence Numbers (UDP) or Accumulated Byte count (TCP)
- **-DR** Dropped Packets (UDP only)
- **-S** Send Rate
- **-R** Receive Rate
- **-IS** | **-IR** Interarrival times of datagrams as seen by the sender or the receiver
- **-D** Delay
- **-AD** Average Delay
- **-J** Jitter

*For **Gnuplot**, we produce two files: one named *output-file***.dem** and the other *output-file***.dat.**

- **`-AJ`** Average Jitter
- **`-L`** Packet Length

If the time scales and units are appropriate, different parameters of interest can also be combined into a single graph by specifying more than one data parameter in the command line. If no parameters of interest are specified, a sequence number plot is created.

For further information see the file **`gengraph.pdf`** in the **`./doc`** directory of the TG distribution.

5 CONCLUDING COMMENTS

The TG software is provided “as is” without express or implied warranties. Neither SRI International (SRI) nor USC /ISI will be held responsible for loss of data or inaccuracies resulting from the use of this program. A mailing list has been set up for TG users. Please send any questions or bug reports to tg@postel.org. To join the mailing list, send a message to tg-request@postel.org. If any modifications are made to TG, we request you notify the mailing list so they can be incorporated into the general distribution, as appropriate. The distribution is available at **`http://www.postel.org/services.html`**.