Tstat::Howto

TCP STatistic and Analysis Tool

Contents

1 Installation

This document provides basic information for the installation, configuration and usage of Tstat and the Bayesian framework for Skype traffic identification. A more general description of the program as well as other documentation can be found in the Tstat homepage http://tstat.tlc.polito.it

1.1 Requirements

1.1.1 Operating System

Tstat has been tested tested on `Linux 2.2.x`, `2.4.x` and `2.6.x` kernels, using `RedHat 6.x-9.x`, and `Fedora Core x` systems. It should work under other `UNIX` dialects, such as `FreeBSD`, `NetBSD 1.3` and `MAC OS X` (although we do not have any of those platforms for testing purposes). If you able to run Tstat on other OSs, we will be happy to include them in the list.

1.1.2 System Libraries

Tstat requires, by itself, a few library that should already be installed on your system, such as `libpcap` (available from http://www.tcpdump.org) and the DAG drivers (available from http://www.endace.com), in case you use such hardware. With these libraries, you are ready to capture and process the traffic flowing in your LAN.

Since Tstat uses pthread to improve the performance in case of real time analysis, your system must support POSIX threads as well if you want to profit of this feature. However, keep in mind that threaded execution is only an optional feature, and is necessary only for online traffic analysis, so that this is not a strict requirement: for this reason, threading is disabled by default.

Finally, to use the RRD functionalities, you also need to have a working installation of RRDtool (available from http://oss.oetiker.ch/rrdtool/).

1.2 Quick Install

Assuming that you want version `2.x.y`:

```
wget http://tstat.polito.it/download/tstat-2.x.y.tgz
tar -xzvf tstat-2.x.y.tgz
cd tstat-2.x.y
./configure
make
make install (with root privileges)
```

This commands install a executable file named `tstat` in `/usr/local/bin`.

1.3 Complete control in Build

The most important elements in the Tstat's package are:

```
tstat/
tstat-conf/
```

```
libtstat/
include/
libtstat-demo/
doc/
doc/HOWTO
README AUTHORS NEWS INSTALL ChangeLog
```

The `tstat` directory contains the source code of Tstat which is also the default building target. Beside Tstat anyway, can be compiled the `Libtstat`, a shared library which allows to an external program to access to the traffic analysis functions of Tstat. In the `include` directory is placed the header file of the library instead in the `libtstat-demo` directory there is a simple program of example that use the Libtstat (see Libtstat library for more information about the Libtstat API).

The building of the Libtstat library is disabled by default but is provided a configuration option to control this feature

```
./configure --enable-libtstat    # build tstat, libtstat and libtstat-demo
./configure                      # build only tstat
```

Tstat's source code uses some preprocess definition to enable/disable some features, like for example the DAG support which is disabled by default. These definitions are declared in the `tstat/Makefile.conf` each with a specific description about its purpose so it should be easy change to behaviour in the building process commenting/uncommenting some lines.

> NB: remember to run `autoreconf` from the root of the package every time a change in these file is perfomed!!!

The building of Libtstat is separated from the building of tstat so `libtstat/Makefile.conf` file define the set of option specific for the Libtstat, instead `tstat/Makefile.conf` is specific for tstat.

In the directory `tstat-conf` are placed some examples of configuration files needed by Tstat, for example the set of local addresses specified using -N option or the set of protocols to dump w.r.t. the internal DPI specified with -T option.

In the directory `doc` the are some plain text files that describes the format of logs files generated by the analysis and this howto document in different formats. `README`, `AUTHORS`, `INSTALL`, `NEWS` and `ChangeLog` instead are plain files that describes some general information abount the package like the authors, the last new features of the tools ecc...

1.4 Complete control in Install

The default `prefix` for installation is `/usr/local` so Tstat executable in installed in `/usr/local/bin` and Libtstat is installed in `/usr/local/lib`. Anyway a different `prefix` can be specified at configuration time

```
./configure --prefix=/absolute/path/where/install/tstat
```

Libtstat-demo is only a demostration tool so is builded only a local executable that is not installed.

Libstat is provided with `pkg-config` support so a `libtstat.pc` is installed in `/usr/lib/pkg-config` and typing

```
pkg-config --cflags --libs libtstat
```

it should appears an output like

```
-I/usr/local/include  -L/usr/local/lib -lm -lpthread -lpcap -lrrd
```

that indicates the `CFLAGS` and `LIBS` options used in the building process.

2 Usage

There are few things to know to run Tstat: first, you are required to have a knowledge of the network that you want to monitor. Second, there are the few options that are described in this section.

2.1 Synopsis

Tstat primary usage is as a command-line tool; the synopsis of the command is the following:

```
Usage:
        tstat  [-htuvwpg] [-d[-d]] [-N file] [-s dir]
              [-B bayes.conf] [-T dump.conf] [-S] [-L]
              [-H ?|file ]
              [-l] [-i interface]
              [-f filterfile] <file1 file2>


Options:
        -h: print this help and exit
        -t: print ticks showing the trace analysis progress
        -u: do not trace UDP packets
        -v: print version and exit
        -w: print [lots] of warning
        -c: concatenate the input files
            (input files should already be in the correct order)
        -p: enable multi-threaded engine (useful for live capture)
        -d: increase debug level (repeat to increase debug level)
        -N file: specify the file name which contains the
                 description of the internal networks.
                 This file must contain the subnets that will be
                 considered as 'internal' during the analysis
                 Each subnet must be specified using network IP address
                 on the first line and NETMASK on the next line:
                    130.192.0.0
                 255.255.0.0
                 193.204.134.0
                 255.255.255.0


        -s dir: puts the trace analysis results into directory
               tree dir (otherwise will be <file>.out)
        -H ?: print internal histograms names and definitions
        -H file: Read histogram configuration from file
```

```
                file describes which histograms tstat should collect
                'include histo_name' includes a single histogram
                'include_matching string' includes all histograms
                whose name includes the string
                special names are:
                'ALL' to include all histograms
                'ADX' to include address hits histogram
                for example, to include all TCP related
                and the address hits histograms, file should be:
                include ADX
                include_matching tcp

        -g: Enable global histo engine
        -S: No histo engine: do not create histograms files
        -L: No log engine: do not create log_* files
        -1: Use old (v1) log_mm format
        -B Bayes_Dir: enable Bayesian traffic classification
                  configuration files from Bayes_Dir
        -T dump.conf: configurazion file for the dump engine
        -l: enable live capture using libpcap
        -i interface: specifies the interface to be used to capture traffic
        -f fiterfile: specifies the libpcap filter file. Syntax as in tcpdump

        file: trace file to be analyzed
              Use 'stdin' to read from standard input.

Optional RRD options:
    -r path: path to use to create/update the RRDtool database
    -R conf: specify the configuration file for integration with
          RRDtool. See README.RRDtool for further informetion.

Optional DPMI options:
    -D conf: DPMI configuration file

Note:
        When tstat is called with no arguments (on the command line),
        it will first check if a file <tstat.conf> is provided in the
        same directory where the execution started.
        In the latter case, arguments will be read from <tstat.conf>
        rather than from the command line

Supported Input File Formats:
        tcpdump          tcpdump format -- Public domain program from LBL
        snoop            Sun Snoop format -- Distributed with Solaris
        etherpeek        etherpeek format -- Mac sniffer program
        netmetrix        Net Metrix format -- Commercial program from HP
        ns               ns format - Network simulator ns2 from LBL
        netscout         NetScout Manager format
        erf              Endace Extensible Record format
        DPMI             Distributed Passive Measurement Interface (DPMI) format
        tcpdump live     Live capture using pcap/tcpdump library
```

2.2 Live Capture vs Trace Analysis

Tstat can sniff and analyze traffic on-line through the use of either the `libpcap` library or Endace DAG cards. The use of Tstat is very easy, especially if you have experiences with `tcpdump`, although `tcpdump`'s knowledge is not required to profitably use Tstat. Moreover, advanced users will enjoy the ability of on-line processing of traffic captured with DAG cards.

As a minimal configuration, you must describe your network to Tstat. Indeed, in order to distinguish forward and backward paths, Tstat needs to know which host IP addresses can be considered as "internal" to the monitored network. In our case, Politecnico di Torino internal addresses are `130.192.0.0/16` and `193.204.134.0/24`, so the network description `net.conf` looks as following:

```
bash> cat net.conf
130.192.0.0    <-- network
255.255.0.0    <-- netmask
193.204.134.0
255.255.255.0
```

We can now run Tstat to capture the traffic flowing across our network, with the following command, which must be run as `root` (as you need to capture packets by putting the Ethernet interface in promiscuous mode). The simplest command is the following:

```
./tstat -l -N net.conf
```

Beside live-capture, it is possible to run Tstat on a previously collected trace file, where the trace format can be any of the following:

```
Supported Input File Formats:
        tcpdump          tcpdump -- Public domain program from LBL
        snoop            Sun Snoop -- Distributed with Solaris
        etherpeek        etherpeek -- Mac sniffer program
        netmetrix        Net Metrix -- Commercial program from HP
        ns               ns -- network simulator from LBL
        netscout         NetScout Manager format
        erf              Endace Extensible Record Format
        DPMI             Distributed Passive Measurement Interface (DPMI) format
        tcpdump live     Live capture using pcap/tcpdump library
```

Tstat will try to read trace files given as input, and to automatically identify the correct dump format. Trace files can be compressed or uncompressed, and Tstat will automatically detect the compression tool used (supported formats are `compress, gzip, bzip2, 7z`).

Without loss of generality, we assume to use the first of the above formats. The calling syntax is similar to the previous one, with the exception of the absence of the live-capture switch `-l` and the presence of the name(s) of the file(s) that have to be processed. For example, the following command can be used to analyze a trace file named `LAN.dump.gz`. Results of the analysis will be stored in a subdirectory named `trace1`; as before, `net.conf` contains the subnet description that will be considered as "internal" during the analysis.

```
./tstat -s trace1 -N net.conf LAN.dump.gz
```

2.3 More Control

We can control the interface that we want to sniff from as well as the output directory name as follows:

```
./tstat -i eth1 -l -s test -Nnet.conf
```

Moreover, we can also pipe Tstat input using the special keyword `stdin` as input, as in the following command (piping ns2 output to Tstat is left as an exercise for the reader):

```
tcpdump -s 80 -i eth0 -w - ip | ./tstat -Nnet.conf -spiped stdin
```

In this case, Tstat is fed by `tcpdump`'s output, and the latter has been instructed to capture packets on the eth0 device, collecting the first 80 bytes (to track uniquely packet headers) of IP packets only, and send the output to `stdout`. Moreover, since Tstat understands the `libpcap` syntax, filters can be stored in text files, as in the following command sequence:

```
echo "vlan and ip and host 10.0.0.1" > tcpdump.conf
./tstat  -i eth0 -l -f tcpdump.conf -N net.conf -s filtered
```

3 Output

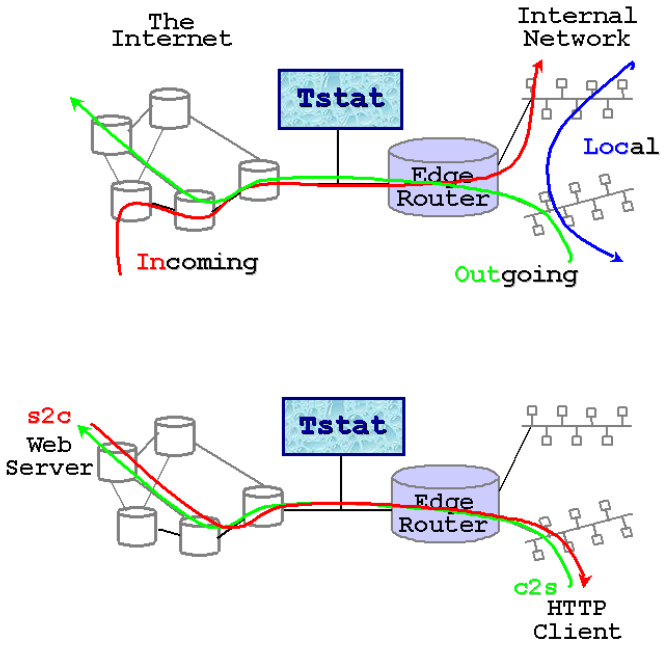
3.1 Output Classification

Recall that Tstat assumes that traces are collected on a bidirectional link, such that both data and control packets belonging to the same flow are observed; with the help of the figures below, we will explain the different classification of measurements used by Tstat.

Tstat identifies hosts based on their IP address. Given the description of the internal hosts through the `-N` command line option, Tstat distinguishes among *incoming*, *outgoing* and *local* measurements. Packets whose destination is an internal host and whose source is an external host will contribute to *incoming* measurements (red arrow in the top figure), whereas packets going in the opposite direction will contribute to *outgoing* measurements (green arrow in the top figure). Finally, in some cases it is possible that Tstat observes packets whose source and destination host belong to the internal host set: in such cases, measurements will be classified as *local* (blue arrow in the top figure). Notice that packets whose source and destination IP addresses belong to the external host set will be discarded. For example, consider a setup in which Tstat is attached to a snoop port of a LAN switch. Then Tstat will be fed by i) *outgoing* packets going to the default gateway, ii) *incoming* packets coming from the default gateway, iii) *local* packets.

Note that if you either do not know or do not want to distinguish between internal, external and local hosts, you may enable the `-DLOG_UNKNOWN` directive when compiling Tstat. Tstat will then be less strict, but results may be difficult to be correctly interpreted.

Considering instead the *role* of the host that sent the packet, statistic are collected distinguishing between *clients* (green arrow in the bottom figure) and *servers* (red arrow in the top figure), i.e., host that opens a connection and and host that replies to connection request. Recall that while TCP connections are well defined, UDP (and RTP/RTCP) connection definition is more fuzzy. In this latter case, Tstat will consider as client the source IP address of



the host that sent the first packet of that flow, while the server will be the host identified by the destination IP address of the same packet.

Therefore, when applicable, Tstat will keep track of measurements referring to the same measured quantity by *appending* a specific tag to the filename:

`_out`
: outgoing: from an internal host to an external host

`_in`
: incoming: from an external host to an internal host

`_loc`
: local between two internal hosts

`_c2s`
: going from the Client to the Server

`_s2c`
: going from the Server to the Client

3.2 Output Structure

Tstat collects several network-layer as well as transport-layer measurements, which are described in full details in http://tstat.polito.it/measure.shtml. As output, Tstat produces three different types of measurement collections, which will be described in the current section:

Histograms

store the *distribution* of a given quantity within a time interval.

Round Robin Database

stores a configurable subset of the same quantities through the RRD interface.

Log files

store a complete transport-layer *log* of all the parameters measured.

3.3 Output Types

This section details the different *types* of measurement collections generated by Tstat; for detailed informations on the specific *metrics* that Tstat is able to gather, please refer to the Tstat website http://tstat.polito.it/measure.shtml.

3.3.1 Histograms

Histograms are generated periodically: Tstat collects all the measurement data during a given measurement interval defined by the `MAX_TIME_STEP` parameter, which is hard-coded in the `param.h` file to 5 minutes. Please, note that changing the `MAX_TIME_STEP` parameter may affect RRD creation as well. For example, considering the IP packet length, Tstat updates, for each observed IP packet, the counter of the number of observed packets with a particular length. At the end of the measurement period, Tstat then saves the values stored in the histogram, resets all the values, and then restarts the samples collection.

Considering the last example of previous section, we run:

```
./tstat -s trace1 -N net.conf 23_00_28_Jun_2000.dump.gz
```

The output generated by tstat consists of a directory tree like the following:

```
trace1
‘-- 23_00_28_Jun_2000.out
    |-- 000
    |   |-- addresses
    |   |-- flow_number
    |   |-- ip_len_in
    |   ...
    |   |-- udp_port_flow_dst
    |   ‘-- udp_tot_time
    |-- 001
    |   |-- addresses
    |   |-- flow_number
    |   |-- ip_len_in
    |   ...
    |   |-- udp_port_flow_dst
    |   ‘-- udp_tot_time
    ...
    |-- LAST
    |   |-- addresses
    |   |-- flow_number
```

```
|   |-- ip_len_in
|   ...
|   |-- udp_port_flow_dst
|   `-- udp_tot_time
|-- log_rtp_complete
|-- log_tcp_complete
`-- log_tcp_nocomplete
```

- Main database

  The topmost directory is created according to the command line option `-s`, which in this case is set to `trace1`. This is intended to be the main database directory.

- Trace Start Time

  A subdirectory named from the timestamp of the first tracked packet is created using the `"%H_%M_%d_%b_%Y.out"` (or, in a more human readable format, `hour_minute_day_Month_year.out/`) notation. When running in live mode (c<-l> option), a new directory with the name of the current tracked packet Timestamp will be created every `DIRS*MAX_TIME_STEP` time. The parameter `DIRS` is defined in the file `param.h` as well. By default it is set to 12, so that a new dir will be approximatively created every hour of live measurement.

- Collection Interval

  Subdirectories with increasing numbers will be created for each measurement period with the format `nnn/`; histograms collecting measurement results will be created in these directories; note that the histograms referring to the last *partial* time period will be stored in the LAST subdirectory. The option `-g` adds also the subdirectory GLOBAL containing the gloabl histogrmas for the whole measurement period.

- Histogram data

  Each of these `nnn/` directories contain several histograms, one for each of the measured parameters, relative to the nnn-th `MAX_TIME_STEP` long time interval; notice that the tags `_in`, `_out`, `_loc`, `_c2s` and `_s2c` are appended to indicate the classification of the observed stream.

  Histogram data are saved using simple ASCII files. The format is simple: the first line contains a description of the measured quantity, while the second line contains the parameters of the histograms (minimum and maximum values, and size of each bins). The list of all the counter index and values is then dumped. To limit the file size, the corresponding entry is omitted if the counter is zero. For example, the histogram of the packet length `ip_len_in` looks like:

```
#IP packet length - incoming packets
#min=0 bin_size=4 max=1600
28 7
36 277
40 11760
44 3463
...
```

Simple Post Processing tools are available to automatically manage the histogram database.

3.3.2 RRD

The RRD output consists of a series of binary files stored in the RRD format. Tstat forces a particular *naming notation* of such files, which follows the configuration rules described later in Sec. 4.2.

The RRD can then be queried with the standard RRDtool commands, such as `rrdcreate`, `rrdupdate`, `rrdgraph`, `rrddump`, `rrdfetch`, `rrdtune`, `rrdlast`, `rrdxport`, to whose manual pages we refer the reader for further informations.

3.3.3 Logs

Tstat creates three transport-layer log-files: `log_tcp_complete`, `log_tcp_nocomplete` and `log_rtp_complete`. Log files are placed in the main database directory.

TCP flows can be either completed or not depending whether Tstat observed the 3-way handshaking or not; in the first case, all the measured indexes relatively to each flow will be collected in the `log_tcp_complete`; in the latter case, flows are considered as garbage and stored in `log_tcp_nocomplete`;

Similarly, a complete log keeping track of each UDP flow measured indexes is maintained in the `log_udp_complete` file. Being UDP basically a connectionless protocol, it is impossible to distinguish among `complete` and `nocomplete` flows in this case.

Furthermore the following log filesa are created: `log_mm_complete` for multimedia flows (i.e. RTP, RTCP, etc), `log_chat_complete` for chating protocols (i.e. MSN, jabber, etc) and `log_skype_complete` for skype traffic.

Description of the file format of the log files can be found in http://tstat.polito.it/measure.shtml.

3.3.4 Traces

Tstat internally has a `Deep Packet Inspector - DPI` which is able to identify traffic communications at application level looking the composition of the payload of packets. Using the `-T` command line option, specifing a configuration file, is possibile to split the input traffic (readed from a trace or from an network card) in traces with the respect to the classification operated by the DPI.

The syntax of the dump configuration file is really simple:

```
> cat tstat-2.x.y/tstat-conf/dump.conf
[udp]           # indicate the set of UDP protocols
rtp
rtcp
edk
kad
kadu
gnutella
bittorrent
dc
kazaa
pplive
sopcast
tvants
unknown         # all the traffic that the DPI don't recognize
complete        # all the traffic
```

The previous list indicate the complete list of protocols of which it can be generated a trace, and, as it can be seen, this feature is for now available only for UDP traffic. In the `tstat-conf` directory of the package there is complete example of this file that can be modified commenting/uncommenting lines to enable/disable the tracking of specific protocols.

The traces generated are placed in a subdirectory named `traces` inside the output directory and the traffic is splitted in traces with windows of 1 hour. This means that, for example, if we start the dump at 9:00 am

```
traces/complete.pcap0       #traffic from 9:00  to 10:00
traces/complete.pcap1       #traffic from 10:00 to 11:00
...
```

3.4 Post Processing

This section could be a separate HOWTO, since this argument cannot be treated exhaustively. Perl, Awk, Ruby *Your-Favorite-Scripting-Language* scripts are definitively best candidates to post-process `log_*` files.

In the Tstat download page, you can find `plot_time.pl` and `plot_cum.pl`, two Perl scripts that may be useful to produce either i) time or ii) aggregated plots over different time spans. They directly access the histogram database created by Tstat. Please, refer to http://tstat.polito.it/software.php#postprocess.

RRD files can be manipulated to obtain *indirect* metrics through the RPN manipulations mechanism provided by RRDtool.

3.5 Storage Considerations

To give the user a rough idea of the size of the output, let us consider a 6 hours long, 1.6GB packet-level trace containing 21M packets, sniffed with `tcpdump` that was used throughout this tutorial. Tstat identified and analyzed about 729K flows, of which about 495K were TCP flows, trashing 20K of them into `log_nocomplete`. Referring to the Sec. 3.2 above shown, we can express the following observations:

Histogram

As previously described, in order to take into account the flow directions, several histograms are dumped for the same variable `var_{in,out,loc,c2s,s2c}`. Currently, about 60 measurement indexes, described in http://tstat.tlc.polito.it/measure.shtml, are logged, for a total of 180 files. Each of the `000/`, `001/` ... `LAST/` directories is about 500KB-1MB depending on the network traffic and on the file system block allocation mechanism.

Therefore, as a rule of thumb, you can count about 1MB of storage due to histograms every 5 minutes of traffic (independently of the amount of actual traffic load during the 5 mins...). This can be useful in order to set the periodic dump timer to the desired trade off among time granularity versus storage size required.

RRD

The `rrd/` directory is, per construction, of fixed size: this should not be a surprise, since this is the goal of RRD. Therefore, the size of the database does not depend on

the amount of network traffic processed, but rather on the RRD configuration. For the standard configuration supplied with Tstat, which is also the one used in our Web server, the whole database occupy only 6MB and consists of about 250 files.

Logs

The total size of the log files amount to about 200MB, which gives a 8x reduction factor w.r.t. the packet-level trace; or, the storage cost of each flow is about 400 bytes.

Note that the `log_*` can be further compressed, using `gzip` to less than 50MB, which gives a further 4x size gain; however, for a matter of performance, is preferable to compress the log files *offline*.

Finally, consider that on a common PC architecture (specifically, Intel P4 2.40GHz equipped with 2GB of RAM and 7200rpm hard-disk), the whole trace elaboration took only 4 minutes; thus, the analysis rate is roughly 85Kpkts/sec or 3Kflows/sec.

4 RRD Module

4.1 RRDtool Installation

In order to get Tstat RRD module working, you will need to install RRDtool first (refer to the homepage of RRDtool http://oss.oetiker.ch/rrdtool/ to accomplish this step). Then, make sure to specify that you want native RRD support in Tstat by modifying the `Makefile` accordingly, and (re)compile Tstat: you will have to uncomment the following lines in `Makefile`, and to double-check that the RRDtool version and path are coherent with your system settings.

```
DEFINES    += -DHAVE_RRDTOOL
RRD_VER     = 1.2.9
RRD_LDLIBS  = -lrrd
RRD_LDFLAGS = -L/usr/lib/ -L/usr/rrdtool/lib/  -L/usr/rrdtool-${RRD_VER}/lib/
RRD_INCS    = -I/usr/rrdtool/include/ -I/usr/rrdtool-${RRD_VER}/include
```

If someone is willing to integrate the RRD identification and configuration directly via the `configure`, we would appreciate it!

4.2 RRD Configuration

Tstat RRD configuration is very easy, being centralized in a single text-file, which allows to specify at runtime what measurements should be monitored. The operating frequencies for the RRD sampling (i.e., the parameters for the temporal averages) are hard-coded into `rrdtool.h` and are chosen to mimic MRTG behavior. Again, take care that modifying the `MAX_TIME_STEP` parameter may affect the RRD management as well.

The RRD configuration file, specified through the command line option `-R`, should contain one line for each of the Tstat parameters that have to be integrated into a Round Robin Database. Each line allows to specify which statistical properties of the variable has to be tracked, as follows:

```
tstat_var1 avg min max stdev var val:a,b,c,d idx:e,f,g,h,other prc:i,j,k
```

where avg,min,max,stdev,var,idx,prc,other are keywords, whereas a,b,c,d,i,j,k are floating point numbers and e,f,g,h integer values; note that the list of indexes (e.g., TCP ports), values (e.g., packet size) and percentiles are comma separated. The name of the variables are Tstat internal ones: they can be seen by executing ./tstat -H, Alternatively, you can directly look into the 000/ ... LAST/ directories or or at http://tstat.polito.it/measure.shtml

Valid configuration lines are, e.g.:

```
#
# inspect IP packet length average, specific values and distribution
#
ip_len_in   avg prc:50,90,95,99 idx:40,1500,other


#
# TCP well known ports
#
# 20    FTP-DATA
# 21    FTP
# 22    SSH
# 23    telnet
# 25    SMTP
# 80    HTTP
# ...
#
tcp_port_dst_in       idx:20,21,22,23,25,80,other

#
# good approximation of the distribution of the RTT,
# taking into account only the incoming path contribution
#
tcp_rtt_avg_in  prc:0.1,1,5,10,25,50,75,90,95,99,99.9
```

where, evidently, the lines starting with a # sign are treated as comments. Our Web server is currently running with the configuration available at http://tstat.polito.it/download/rrd.conf.

For each specified quantity defined in the rrd.conf file, a corresponding file will be created. For example, consider that the generic configuration line:

```
tstat_var avg min max stdev var val:a,b,c,d idx:e,f,g,h,other prc:i,j,k
```

will produce the following files (17 in total):

```
tstat_var.{avg,min,max,stdev,var}.rrd
tstat_var.val{a,b,c,d}.rrd
tstat_var.idx{e,f,g,h,oth}.rrd
tstat_var.prc{i,j,k}.rrd
```

4.3 Tstat RRD and the Web

From the Tstat web site, you can download the most up-to-date version of tstat_rrd.cgi, which is the CGI script that renders the Web interface. Here is some basic tips to get it working; if you wonder how to write your own graph templates, then you are probably skilled enough to get it on your own :)

4.3.1 Database Structure

The CGI scripts allow to browse on the fly the RRD database structure. The `rrd_data` directory is the root of the tree, where each directory contains either i) other directories (i.e., is a box) or ii) a RRD-database, in which case the node is a leaf and will be shown in the interface. In case that a directory is a plain box, it may optionally contain some files (specifically {`HEADER`,`FOOTER`,`README`}.{`html`,`txt`}) that will be rendered by `tstat_rrd.cgi`. By default, the cgi script tries to load the html version; otherwise, it tries to displays "<pre> 'cat FILE' </pre>" if such a FILE exists; finally, it will display a default message held in $default{README} hard coded in the script.

Here is an example of the rrd_data directory which holds part of the RDD database accessible from the tstat web page.

```
rrd_data/
 |-- Example
 |-- GARR
 |   |-- garr-live
 |   `-- garr-old
 `-- Polito
     |-- 2000
     |   |-- Apr
     |   |-- Jun
     |   |-- Jun,post155
     |   `-- May
     |-- 2001
     |   |-- Feb
     |   `-- Jan
     |-- 2005
     |   |-- Apr
     |   `-- Feb
     `-- Current
```

4.3.2 Web Configuration

The web configuration really depends on your web server configuration. Few dependencies are required, most notably, the RRD Perl library from the RRDtool installation.

It is advisable to store the Tstat RRD files everywhere you want, and then create a symbolic link named `rrd_data` that points to it (i.e., to the root of the rrd database tree). Similarly for the directory where the rendered images should be stored (defaults to `cgi-bin/rrd_images`) and can be a symbolic link as well. The names of these symbolic links can be redefined in the configuration section of `tstat_rrd.cgi` if needed:

```
#   ____________________________
#  /                            \
# /    configuration  __________/
# \__________________/.:nonsns:.
#
# specify path to the root of the rrd database tree
# by default, I assume there is a symbolic link in cgi-bin/
# named rrd_data
$RRD_DATA = 'rrd_data';
```

```
# same thing for image directory
# in my case, var/www/cgi-bin/rrd_images is
# a symbolic link to "/var/www/html/rrd_images";
# from the html browser's perspective
$IMG_DIR = "rrd_images";
```

5 DPMI Module

To the experienced DPMI user, it can turn very useful to think of Tstat in terms of a DPMI consumer, thus suitable for live usage. Basically, two configuration files need to be provided in this case.

5.1 Tstat Configuration for DPMI

Especially for this purpose, Tstat can be executed without any argument on the command line, provided that a file named `tstat.conf` exists in the same path where the `tstat` command has been executed. Note that the filename MUST be in this case `tstat.conf`

In the latter case, arguments will be read from `tstat.conf` rather than from the command line, which makes Web-based execution easier – it just requires the creation of a text file.

Typically, the content of the file will be one of the two following cases. When only the RRD module need to be turned on, which is specially suitable for the persistent monitoring of a network link:

```
-D dpmi.conf -S -R rrd.conf -r data.rrd
```

Or, in the case where more detailed transport layer logs and histograms are to be generated, such as for shorter ad-hoc experiment:

```
-D dpmi.conf -s data
```

Note that the `dpmi.conf` filename, which is the object of the next section, is customizable.

5.2 DPMI Configuration for Tstat

This file is used by Tstat in order to properly set-up the DPMI library and, possibly, its filters. There are only two keywords that are interpreted by Tstat, and *the whole* content of this file is passed to the DPMI's `createfiler` library call. Tstat-keywords are prepended by the `tstat:` prefix, to solve any ambiguity, and are related to the type of stream and measurement direction. More specifically,

`tstat:(file|(tcp|udp|eth)[:port])`

> Specify whether a tracefile or a network socket (and in this case, which port) is the source of DPMI traffic. Note that in the case where a tracefile is used, there is no real need to specify this, since the format recognition happens automatically; thus, the `tstat:file` keyword is provided for completeness. Conversely, options such as `tstat:eth` and `tstat:tcp:32449` are necessary in order for network sokets to properly be setup.

`tstat:MP:CI`

This option is used to define the traffic directionality, specifing what network card interface (CI) and the measurement point (MP) are related to *incoming* traffic from external sources. Referring to the DPMI library internals:

```
CI <-> char nic[8];
MP <-> char mampid[8];
```

In order to provide a safe failback or a missing configuration, unless otherwise specified, the first received frame is assumed to be "incoming", thus arbitrarily determinining the incoming CI:MP couple.

6 Bayesian Classification of Skype Traffic

This section deals with the task of configuring and tuning the Bayesian framework integrated within Tstat.

6.1 A Simple Example

Assuming that you have configured the Bayesian classifier (a proper configuration sample for the Bayesian detection of Skype traffic is provided in the `skype` directory), traffic identification can be carried on either on-line or off-line: in both cases, a per-flow logfile will be created as output.

```
./tstat -l -B skype -N net.conf
```

```
./tstat -B skype -N net.conf tracefile.dump
```

6.2 Framework Configuration

The Bayesian framework is configured through a directory name, containing several configuration files. A default configuration is provided under the `tstat-conf` directory: there are two main files that allow to finely tune the framework, as described in details below. These files specify the parameter settings for different Naive Bayes Classifiers (NBC), corresponding to different features of the flows. As features, the framework consider the **packet size** (`pktsize.conf`) and **average inter-packet gap** (`avgipg.conf`).

Each of these files has the format specified in the example below, where lines starting with a dash sign `#` are interpreted as comments.

```
#   ____________________________
#  /                            \
# /    BayesConf      __________/
# \__________________/.:nonsns:.
#
#================================================
# feature name
```

```
#-----------------------------------------------
# Known Skype features:
#    PKTSIZE
#    MAXDELPKTSIZE
#    AVGIPG
#    PKTRATE
#    BITRATE
#
FEATURE      AVGIPG
#
#===============================================
# default flags
#-----------------------------------------------
# USE_LOG        1
# NORMALIZE      1
# AUTO_OTHER     0
#
WINDOW_SIZE      1
CLASS_LEN        250
MIN_THRESHOLD    1e-25
AVG_THRESHOLD   -3.5
WIN_THRESHOLD   -3
#
#===============================================
# class definition
#-----------------------------------------------
# syntax
#    DISCRETE  class P{class}
#    GAUSSIAN  class P{class} mu sigma
#    GAUSSIAN+ class P{class} N (w1,m1,s1) .. (wN,mN,sN)
#
# note: P{class} may be "="
#
GAUSSIAN mode1 = 10 2
GAUSSIAN mode2 = 20 2
GAUSSIAN mode3 = 30 2
GAUSSIAN mode4 = 40 2
GAUSSIAN mode5 = 50 2
GAUSSIAN mode6 = 60 2
```

First of all, the feature name is defined, then some optional parameters are set (such as the window size, the optional use of logarithms to avoid underflows in Bayesian belief computation, the use of normalization, etc.).

Minimum threshold avoids underflows in Bayesian belief computation, the max mean belief is compared against the average threshold to decide whether the flow can be considered skype or non-skype and the window threshold is used to count the number of individual samples (rather than their mean) that passes a more restrictive test (indeed the window threshold is greater than the average threshold).

Finally, different classes for the given feature are defined, which correspond to different "modes". Such classes may be represented as gaussian (or superposition of gaussian) distributions, centered over a given mean and with a given standard deviation, or may be arbitrarily specified (discrete distribution). The class probability is equally shared among

the classes by the keyword = as in the example above. As we will see, changing the order of the class definition affects the meaning of some of the logfile fields values.

7 Libtstat library

As described in the Install section of this document, to enable the building of Libtstat library is needed to provide a configure option

```
./configure --enable-libtstat
```

7.1 Link the Libtstat library

When the library is installed in the system using `make install` the following messages are printed on the console

```
----------------------------------------------------------------------
Libraries have been installed in:
/usr/local/lib

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
- add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
  during execution
- add LIBDIR to the 'LD_RUN_PATH' environment variable
  during linking
- use the '-Wl,--rpath -Wl,LIBDIR' linker flag
- have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------
```

This indicate where the library has been installed and how to link that to some program. The most simple thing to do, is to use the native libtool support for automake, that is, assuming that `program_name` is the name of the executable of the tool to generate, it is needed to add the following lines to `Makefile.am` of the tool:

```
program_name_LDADD = -ltstat -lpcap -lpthread -lm
program_name_LDFLAGS = -Wl,--rpath -Wl,<libtstat_dir>
```

This allow a fined control on the directory where the library has been installed. Anyway, if it has been installed in a standard library location (as `/usr/lib`), instead of the previous lines, it can be added

```
AC_CHECK_LIB([tstat], [tstat_next_pckt],, AC_MSG_ERROR([missing 'tstat' library]))
```

in `configure.ac` of the current project. This automatically look for the presence of a function tstat_next_pckt() in a system library named `libtstat`. In case of error of error print a message stopping the configuration process, instead in case of success, are automatically added all the linking options needed to build the program (see the autotools files in `libtstat-demo` for a complete example).

7.2 Libtstat API

int tstat_init (char *config_fname)

`config_fname` is a file name containing a set of Tstat options, one for each line

```
>cat tstat-conf/tstat.conf
#-s outdir      # output directory
-N net.all      # network config file
#-B bayesdir    # directory of the bayes config files
#-d             # debug
```

If NULL is provided, the library use `./tstat.conf` as filename.

void tstat_new_logdir (char *filename, struct timeval *pckt_time)

This function has to be called before the process of the first packet and allow to generate the output directory using this hierarchy:

```
<filename>.out
    |__<pckt_time>.out
```

int tstat_next_pckt (struct timeval *pckt_time, void *ip_hdr, void *last_ip_byte, int tlen)

This function enable the processing of a new packet. `pckt_time` is the timestamp of the packet, `ip_hdr` is a pointer to the first ip byte, `last_ip_byte` is a pointer to the last ip byte, and `tlen` is the number of total bytes (captured).

tstat_report *tstat_close (tstat_report *report)

This function flush to file all the pending statistics and fill a tstat_report structure with some general results.

void tstat_print_report (tstat_report *report, FILE *file)

This function print a formatted report to file using tstat_report data.

8 Author Informations

Marco Mellia, Assistant Professor.
<marco.mellia@polito.it>

Dario Rossi, PostDoc Researcher.
<dario.rossi@polito.it>

Dario Bonfiglio, Researcher.
<bonfiglio@serverlipar.polito.it>

Telecommunication Networks Group (TNG)
DELEN, Politecnico di Torino
http://www.tlc-networks.polito.it

9 Acknowledgment

Many people contributed to the development of Tstat. Tstat would never have seen the light had not `TCPTrace` being invented. Many thanks to Shawn Ostermann and to the Ohio University for their great program.

Many Master and PhD students took part in the development and debugging of Tstat. Naming all of them would be impossible. We would then like to thank Luca Muscariello for the entropy generated in the TCP anomalies identification, and Prof. Marco Ajmone Marsan and Prof. Fabio Neri who gave us the moral and scientific support to continue investing in Tstat.

10 License

Copyright (c) 2001 Politecnico di Torino. All rights reserved.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.