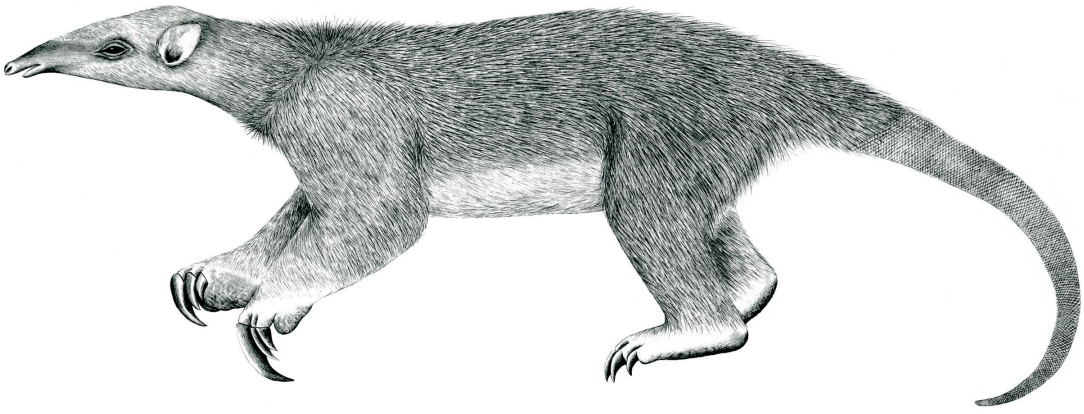
Tranalyzer2

Version 0.9.3 (Cobra)

Flow based forensic and network troubleshooting traffic analyzer



Tranalyzer Development Team

Contents

1 Introduction **1**
1.1 Getting Tranalyzer . . . 1
1.2 Dependencies . . . 1
1.3 Compilation . . . 2
1.4 Installation . . . 3
1.5 Getting Started . . . 3
1.6 Getting Help . . . 3

2 Tranalyzer2 **5**
2.1 Supported Link-Layer Header Types . . . 5
2.2 Enabling/Disabling Plugins . . . 5
2.3 Man Page . . . 7
2.4 Invoking Tranalyzer . . . 7
2.5 hashTable.h . . . 13
2.6 ioBuffer.h . . . 14
2.7 loadPlugins.h . . . 14
2.8 main.h . . . 14
2.9 networkHeaders.h . . . 16
2.10 proto/capwap.h . . . 16
2.11 proto/ethertype.h . . . 16
2.12 proto/linktype.h . . . 16
2.13 proto/lwapp.h . . . 17
2.14 packetCapture.h . . . 17
2.15 tranalyzer.h . . . 17
2.16 bin2txt.h . . . 28
2.17 gz2txt.h . . . 28
2.18 outputBuffer.h . . . 28
2.19 rbTree.h . . . 29
2.20 subnetHL.h . . . 29
2.21 t2log.h . . . 29
2.22 Tranalyzer2 Output . . . 29
2.23 Final Report . . . 30
2.24 Monitoring Modes During Runtime . . . 33
2.25 Cancellation of the Sniffing Process . . . 39

1 Introduction

Tranalyzer2 is a lightweight flow generator and packet analyzer designed for simplicity, performance and scalability. The program is written in C and built upon the *libpcap* library. It provides functionality to pre- and post-process IPv4/IPv6 data into flows and enables a trained user to see anomalies and network defects even in very large datasets. It supports analysis with special bit coded fields and generates statistics from key parameters of IPv4/IPv6 tcpdump traces either being live-captured from an Ethernet interface or one or several pcap files. The quantity of binary and text based output of Tranalyzer2 depends on enabled modules, herein denoted as **plugins**. Hence, users have the possibility to tailor the output according to their needs and developers can develop additional plugins independent of the functionality of other plugins.

1.1 Getting Tranalyzer

Tranalyzer can be downloaded from: https://tranalyzer.com/downloads.html

1.2 Dependencies

Tranalyzer2 requires the following tools and libraries:

Kali/Ubuntu:

```
sudo apt-get install autoconf autoconf-archive automake libbsd-dev libpcap-dev
    libreadline-dev libtool make meson zlib1g-dev
```

Arch/Manjaro:

```
sudo pacman -S autoconf autoconf-archive automake bash-completion gcc libpcap
    libtool make meson pkgconf zlib
```

CentOS/Fedora/Red Hat:

```
sudo dnf install autoconf autoconf-archive automake bzip2 libbsd-devel
    libpcap-devel libtool meson readline-devel zlib-devel [1]
```

Gentoo:

```
sudo emerge autoconf autoconf-archive automake bash-completion libpcap libtool meson zlib
```

openSUSE:

```
sudo zypper install autoconf autoconf-archive automake gcc libbsd-devel
    libpcap-devel libtool meson readline-devel zlib-devel
```

macOS:

```
brew install autoconf autoconf-archive automake libpcap libtool meson readline zlib [2]
```

Note that `meson` is optional, but recommended as it is much faster than the autotools (`autoconf`, `automake`, ...).

[1] If the `dnf` command could not be found, try with `yum` instead

[2] Brew is a packet manager for macOS that can be found here: https://brew.sh

1.3 Compilation

To build Tranalyzer2 and the plugins, run one of the following commands:

- Tranalyzer2 only:
  `cd "$T2HOME"; ./autogen.sh tranalyzer2`
  (alternative: `cd "$T2HOME/tranalyzer2"; ./autogen.sh`)
- A specific plugin only, e.g., `myPlugin`:
  `cd "$T2HOME"; ./autogen.sh myPlugin`
  (alternative 1: `cd "$T2PLHOME/myPlugin"; ./autogen.sh`)
  (alternative 2: `cd "$T2HOME/plugins/myPlugin"; ./autogen.sh`)
- Tranalyzer2 and a default set of plugins:
  `cd "$T2HOME"; ./autogen.sh`
- Tranalyzer2 and all the plugins in `T2HOME`:
  `cd "$T2HOME"; ./autogen.sh -a`
- Tranalyzer2 and a custom set of plugins (listed in plugins.build) (Section 1.3.1):
  `cd "$T2HOME"; ./autogen.sh -b`

where `T2HOME` points to the root folder of Tranalyzer, i.e., where the file `README.md` is located.

For finer control of which plugins to load, refer to Enabling/Disabling Plugins .

Note that if `t2_aliases` is installed, the `t2build` command can be used instead of `autogen.sh`. The command can be run from anywhere, so just replace the above commands with `t2build tranalyzer2`, `t2build myPlugin`, `t2build -a` and `t2build -b`. Run `t2build --help` for the full list of options accepted by the script.

1.3.1 Custom Build

The `-b` option of the `autogen.sh` script takes an optional file name as argument. If none is provided, then the default `plugins.build` is used. The format of the file is as follows:

- Empty lines and lines starting with a `'#'` are ignored (can be used to prevent a plugin from being built)
- One plugin name per row
- Example:

```
# Do not build the tcpStates plugin
#tcpStates

# Build the txtSink plugin
txtSink
```

A `plugins.ignore` file can also be used to prevent specific plugins from being built. A different filename can be used with the `-I` option.

1.4 Installation

The `-i` option of the `autogen.sh` script installs Tranalyzer in `/usr/local/bin` (as `tranalyzer`) and the man page in `/usr/local/man/man1`. Note that root rights are required for the installation.

Alternatively, use the file `t2_aliases` or add the following alias to your `~/.bash_aliases`:

```
alias tranalyzer="$T2HOME/tranalyzer2/src/tranalyzer"
```

where `T2HOME` points to the root folder of Tranalyzer, i.e., where the file `README.md` is located.

The man page can also be installed manually, by calling (as root):

```
mkdir -p /usr/local/man/man1 && gzip -c man/tranalyzer.1 > /usr/local/man/man1/tranalyzer.1.gz
```

1.4.1 Aliases

The file `t2_aliases` documented in `$T2HOME/scripts/doc/scripts.pdf` contains a set of aliases and functions to facilitate working with Tranalyzer. To install it, append the following code to `~/.bashrc` or `~/.bash_aliases` (make sure to replace `$T2HOME` with the actual path, e.g., `$HOME/tranalyzer2-0.9.3`):

```
if [ -f "$T2HOME/scripts/t2_aliases" ]; then
    . "$T2HOME/scripts/t2_aliases"            # Note the leading `.'
fi
```

1.5 Getting Started

Run Tranalyzer as follows:

```
tranalyzer -r file.pcap -w outfolder/outprefix
```

For a full list of options, use Tranalyzer `-h` option: `tranalyzer -h` or refer to the complete documentation.

1.6 Getting Help

1.6.1 Documentation

Tranalyzer and every plugin come with their own documentation, which can be found in the `doc` subfolder. The complete documentation of Tranalyzer2 and all the locally available plugins can be generated by running `make` in `$T2HOME/doc`. The file `t2_aliases` provides the function `t2doc` to allow easy access to the different parts of the documentation from anywhere.

1.6.2 Man Page

If the man page was installed (Section 1.4), then accessing the man page is as simple as calling

```
man tranalyzer
```

If it was not installed, then the man page can be invoked by calling

```
man $T2HOME/tranalyzer2/man/tranalyzer.1
```

1.6.3 Help

For a full list of options, use Tranalyzer `-h` option: `tranalyzer -h`

1.6.4 FAQ

Refer to the complete documentation in `$T2HOME/doc` for a list of frequently asked questions.

1.6.5 Contact

Any feedback, feature requests and questions are welcome and can be sent to the development team via email at:

andy@tranalyzer.com

2 Tranalyzer2

Tranalyzer2 is designed in a modular way. Thus, the packet flow aggregation and the flow statistics are separated. While the main program performs the header dissection and flow organization, the plugins produce specialized output such as packet statistics, mathematical transformations, signal analysis and result file generation.

2.1 Supported Link-Layer Header Types

Tranalyzer handles most PCAP link-layer header types automatically. Some specific types can be analyzed by switching on flags in `linktypes.h`. The following table summarizes the link-layer header types handled by Tranalyzer:

Linktype	Description	Flags
DLT_C_HDLC	Cisco PPP with HDLC framing	
DLT_C_HDLC_WITH_DIR	Cisco PPP with HDLC framing preceded by one byte direction	
DLT_EN10MB	IEEE 802.3 Ethernet (10Mb, 100Mb, 1000Mb and up)	
DLT_FRELAY	Frame Relay	
DLT_FRELAY_WITH_DIR	Frame Relay preceded by one byte direction	
DLT_IEEE802_11	IEEE802.11 wireless LAN	
DLT_IEEE802_11_RADIO	Radiotap link-layer information followed by an 802.11 header	
DLT_IPV4	Raw IPv4	
DLT_IPV6	Raw IPv6	
DLT_JUNIPER_ATM1	Juniper ATM1 PIC (experimental)	LINKTYPE_JUNIPER=1
DLT_JUNIPER_ETHER	Juniper Ethernet (experimental)	LINKTYPE_JUNIPER=1
DLT_JUNIPER_PPPOE	Juniper PPPoE PIC (experimental)	LINKTYPE_JUNIPER=1
DLT_LAPD	Raw LAPD	LAPD_ACTIVATE=1
DLT_LINUX_LAPD	LAPD (Q.921) frames, with a DLT_LINUX_SLL header	LAPD_ACTIVATE=1
DLT_LINUX_SLL	Linux “cooked” capture encapsulation	
DLT_NULL	BSD loopback encapsulation	
DLT_PPI	Per-Packet Information	
DLT_PPP	Point-to-Point Protocol	
DLT_PPP_SERIAL	PPP in HDLC-like framing	
DLT_PPP_WITH_DIR	PPP preceded by one byte direction	
DLT_PRISM_HEADER	Prism monitor mode information followed by an 802.11 header	
DLT_RAW	Raw IP	
DLT_SYMANTEC_FIREWALL	Symantec Enterprise Firewall	

2.2 Enabling/Disabling Plugins

The plugins are stored under *~/.tranalyzer/plugins*. This folder can be changed with the `-p` option.

By default, all the plugins found in the plugin folder are loaded. This behavior can be changed by altering the value of `USE_PLLIST` in *loadPlugins.h:35*. The valid options are:

USE_PLLIST	Description
0	disable `-b` option and load all plugins from the plugin folder (default)
1	only load plugins present in the list (whitelist)

USE_PLLIST	**Description**
2	do not load plugins present in the list (blacklist)

This following sections discuss the various ways to selectively enable/disable plugins.

2.2.1 Default

By default, all the files in the plugin folder named according to the following pattern are loaded:

```
^[0-9]{3}_[a-zA-Z0-9]+.so$
```

To disable a plugin, it must be removed from the plugin folder. A subfolder, e.g., *disabled*, can be used to store unused plugins.

2.2.2 Whitelisting Plugins

If `USE_PLLIST=1`, the whitelist (loading list) is searched under the plugins folder with the name `plugins.txt`. The name can be changed by adapting the value `PLLIST` in *loadPlugins.h:36*. If the file is stored somewhere else, Tranalyzer2 `-b` option can be used.

The format of the whitelist is as follows (empty lines and lines starting with a `'#'` are ignored):

```
# This is a comment

# This plugin is whitelisted (will be loaded)
001_protoStats.so

# This plugin is NOT whitelisted (will NOT be loaded)
#010_basicFlow.so
```

Note that if a plugin is not present in the list, it will **NOT** be loaded.

2.2.3 Blacklisting Plugins

If `USE_PLLIST=2`, the blacklist is searched under the plugins folder with the name `plugins.txt`. The name can be changed by adapting the value `PLLIST` in *loadPlugins.h:36*. If the file is stored somewhere else, Tranalyzer2 `-b` option can be used.

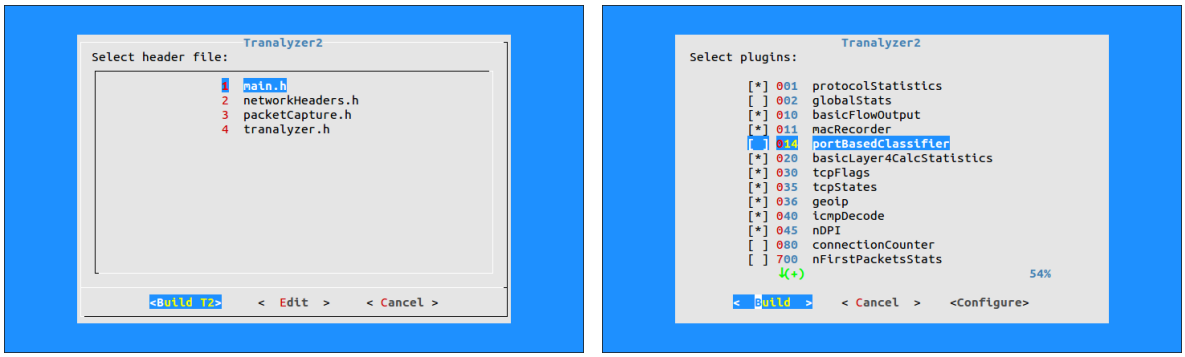
The format of the blacklist is as follows (empty lines and lines starting with a `'#'` are ignored):

```
# This is a comment

# This plugin is blacklisted (will NOT be loaded)
001_protoStats.so

# This plugin is NOT blacklisted (will be loaded)
#010_basicFlow.so
```

Note that if a plugin is not present in the list, it will be loaded.

2.2.4 Graphical Configuration and Building of T2 and Plugins

Tranalyzer2 comes with a script named `t2conf` allowing easy configuration of all the plugins through a command line based graphical menu:



Use the arrows on your keyboard to navigate up and down and between the buttons. The first window is only displayed if the `-t2` option is used. The `Edit` and `Configure` buttons will launch a text editor (`$EDITOR` or vim[3] if the environment variable is not defined). The second window can be used to activate and deactivate plugins (toggle the active/inactive state with the space key).

To access the script from anywhere, use the provided `install.sh` script, install `t2_aliases` or manually add the following alias to `~/.bash_aliases`:

```
alias t2conf="$T2HOME/scripts/t2conf/t2conf"
```

Where `$T2HOME` is the folder containing the source code of Tranalyzer2 and its plugins.

A man page for `t2conf` is also provided and can be installed with the `install.sh` script.

2.3 Man Page

If the man page was installed (Section 1.4), then accessing the man page is as simple as calling

```
man tranalyzer
```

If it was not installed, then the man page can be invoked by calling

```
man $T2HOME/tranalyzer2/man/tranalyzer.1
```

2.4 Invoking Tranalyzer

As stated earlier Tranalyzer2 either operates on Ethernet/DAG interfaces or pcap files. It may be invoked using a BPF if only certain flows are interesting. The required arguments are listed below. Note that the `-i`, `-r`, `-R` and `-D` options cannot be used at the same time.

[3] The default editor can be changed by editing the variable `DEFAULT_EDITOR` (line 7)

2.4.1 Help

For a full list of options, use the `-h` option: `tranalyzer -h`

```
Tranalyzer 0.9.2 - High performance flow based network traffic analyzer

Usage:
    tranalyzer [OPTION...] <INPUT>

Input arguments:
    -i IFACE     Listen on interface IFACE
    -r PCAP      Read packets from PCAP file or from stdin if PCAP is "-"
    -R FILE      Process every PCAP file listed in FILE
    -D EXPR[:SCHR][,STOP]
                 Process every PCAP file whose name matches EXPR, up to an
                 optional last index STOP. If STOP is omitted, then Tranalyzer
                 never stops. EXPR can be a filename, e.g., file.pcap0, or an
                 expression, such as "dump*.pcap00", where the star matches
                 anything (note the quotes to prevent the shell from
                 interpreting the expression). SCHR can be used to specify
                 the last character before the index (default: 'p')

Output arguments:
    -w PREFIX    Append PREFIX to any output file produced. If the option is
                 omitted, derive PREFIX from the input. Use '-w -' to output
                 the flow file to stdout (other files will be saved as if the
                 '-w' option had been omitted and the '-l' and '-m' options used)
    -W PREFIX[:SIZE][,START]
                 Like -w, but fragment flow files according to SIZE, producing
                 files starting with index START. SIZE can be specified in bytes
                 (default), KB ('K'), MB ('M') or GB ('G'). Scientific notation,
                 i.e., 1e5 or 1E5 (=100000), can be used as well. If a 'f' is
                 appended, e.g., 10Kf, then SIZE denotes the number of flows.
    -l           Print end report in PREFIX_log.txt instead of stdout
    -m           Print monitoring output in PREFIX_monitoring.txt instead of stdout
    -s           Packet forensics mode

Interface capture arguments:
    -S SNAPLEN   Set the snapshot length (used with -i option)
    -B BUFSIZE   Set the live Rx buffer size (used with -i option)

Optional arguments:
    -p PATH      Load plugins from PATH instead of ~/.tranalyzer/plugins
    -b FILE      Use plugin list FILE instead of plugin_folder/plugins.txt
    -e FILE      Create a PCAP file by extracting all packets belonging to
                 flow indexes listed in FILE (require pcapd plugin)
    -f FACTOR    Set hash multiplication factor
    -x ID        Sensor ID
    -c CPU       Bind tranalyzer to one core. If CPU is 0 then OS selects the
```

```
                core to bind
    -P PRIO     Set tranalyzer priority to PRIO (int) instead of 0
                (PRIO [highest, lowest]: [-20, 20] (root), [0, 20] (user))
    -M FLT      Set monitoring interval to FLT seconds
    -F FILE     Read BPF filter from FILE

Help and documentation arguments:
    -V          Show the version of the program and exit
    -h          Show help options and exit

Remaining arguments:
    BPF         Berkeley Packet Filter command, as in tcpdump
```

2.4.2 –i INTERFACE

Capture data from an Ethernet interface `INTERFACE` (requires *root* privileges). If high volume of traffic is expected, then enable internal buffering in ioBuffer.h.

```
sudo tranalyzer -i eth0
```

2.4.3 –r FILE

Capture data from a pcap file `FILE`.

```
tranalyzer -r file.pcap
```

The special file `'-'` can be used to read data from *stdin*. This can be used, e.g., to process compressed pcap files, e.g., *file.pcap.gz*, using the following command:

```
zcat file.pcap.gz | tranalyzer -r - -w out
```

2.4.4 –R FILE

Process all the pcap files listed in `FILE`. All files are being treated as one large file. The life time of a flow can extend over many files. The processing order is defined by the location of the filenames in the text file. The absolute path has to be specified. The `t2caplist` script documented in `$T2HOME/scripts/scripts.pdf` can be used to generate such a list. All lines starting with a `'#'` are considered as comments and thus ignored.

```
$ t2caplist directory > pcap_list.txt
$ tranalyzer -R pcap_list.txt
```

2.4.5 –D FILE[*][.ext]#1[:SCHR][,#2]

Process files in a directory using file start and stop index, defined by `#1` and `#2` respectively. `ext` can be anything, e.g., `.pcap`, and can be omitted. If `#2` is omitted and not in round robin mode, then Tranalyzer2 never stops and waits until the next file in the increment is available. If leading zeroes are used, `#2` defaults to $10^{\text{number\_length}} - 1$. Note that only the last occurrence of `SCHR` is considered, e.g., if `SCHR='p'`, then out.pca**p**001 will work, but out001pca**p**, will not. with the `:[SCHR]` option a new separation character can be set, superseding `SCHR` defined in tranalyzer.h.

The following variables in tranalyzer.h can be used to configure this mode:

Name	Default	Description
RROP	0	Activate round robin operations WARNING: if set to 1, then `findexer` will not work anymore
POLLTM	5	Poll timing (in seconds) for files
MFPTMOUT	0	> 0: timeout for poll > `POLLTM`, 0: no poll timeout
SCHR	'p'	Separating character for file number

For example, when using `tcpdump` to capture traffic from an interface (eth0) and produce 100MB files as follows:

```
sudo tcpdump -C 100 -i eth0 -w out.pcap
```

The following files are generated: *out.pcap, out.pcap1, out.pcap2, . . . , out.pcap10, . . .*

Then `SCHR` must be set to `'p'`, i.e., the last character before the file number (out.pca**p**NUM) and Tranalyzer must be run as follows:

```
tranalyzer -D out.pcap
```

Or to process files 10 to 100:

```
tranalyzer -D out.pcap10,100
```

Or to process files 10 to 100 in another format:

```
tranalyzer -D out10.pcap,100 -w out
```

Or to process files from 0 to $2^{32} - 1$ using regex characters:

```
tranalyzer -D "out*.pcap" -w out
```

The last command can be shortened further, the only requirement being the presence of `SCHR` (the last character before the file number) in the pattern:

```
tranalyzer -D "*p" -w out
```

Note the quotes (") which are necessary to avoid preemptive interpretation of regex characters and `SCHR` which MUST appear in the pattern. The same configuration can be used for filenames using one or more leading zeros, e.g., *out.pcap000, out.pcap001, out.pcap002, . . . , out.pcap010, . . .*

The following table summarizes the supported naming patterns and the configuration required:

Filenames	SCHR	Command
out.pca**p**, out.pca**p**1, out.pca**p**2, . . .	'p'	`tranalyzer -D out.pcap -w out`
out.pca**p**00, out.pca**p**01, out.pca**p**02, . . .	'p'	`tranalyzer -D out.pcap00 -w out`
ou**t**0.pcap, ou**t**1.pcap, ou**t**2.pcap, . . .	't'	`tranalyzer -D out0.pcap -w out`
ou**t**00.pcap, ou**t**01.pcap, ou**t**02.pcap, . . .	't'	`tranalyzer -D out00.pcap -w out`
out_24.04.2016.20h00.pca**p**, out_24.04.2016.20h00.pca**p**1, . . .	'p'	`tranalyzer -D "out*.pcap" -w out`

Filenames	SCHR	Command
out_24.04.2016.20h00.pca**p**00, out_24.04.2016.20h00.pca**p**01, ...	'p'	`tranalyzer -D "out*.pcap00" -w out`
ou**t**0.pcap, ou**t**1.pcap, ou**t**2.pcap, ...	't'	`tranalyzer -D out0.pcap:t -w out`
out.pca**p**00, out.pca**p**01, out.pca**p**02, ...	'p'	`tranalyzer -D out.pcap00:p -w out`

2.4.6 –w PREFIX

Use a `PREFIX` for all output file types. The number of files being produced vary with the number of activated plugins. The file suffixes are defined in the file tranalyzer.h (see Section 2.15.13) or in the header files for the plugins. If you forget to specify an output file, Tranalyzer will use the input interface name or the file name as file prefix and print the flows to *stdout*. Thus, Tranalyzer output can be piped into other command line tools, such as netcat in order to produce centralized logging to another host or an AWK script for further post processing without intermediate writing to a slow disk storage.

2.4.7 –W PREFIX[:SIZE][,START]

This option allows the fragmentation of flow files produced by Tranalyzer independent of the input mode. The expression before the `':'` is the output prefix, the expression after the `':'` denotes the maximal file size for each fragment and the number after the `','` denotes the start index of the first file. If omitted it defaults to `0`. The size of the files can be specified in bytes (default), KB (`'K'`), MB (`'M'`) or GB (`'G'`). Scientific notation, i.e., 1e5 or 1E5 (=100000), can be used as well. If no size is specified, the default value of 500MB, defined by `OFRWFILELN` in tranalyzer.h is used. If no size is specified, then the `':'` can be omitted. The same happens if no start index is specified. If an additional `'f'` is appended the unit is flow count. This enables the user to produce file chunks containing the same amount of flows. Some typical examples are shown below.

Command	Fragment Size	Start Index	Output Files
`tranalyzer -r nudel.pcap -W out:1.5E9,10`	1.5GB	10	out10, out11, ...
`tranalyzer -r nudel.pcap -W out:1.5e9,5`	1.5GB	5	out5, out6, ...
`tranalyzer -r nudel.pcap -W out:1.5G,1`	1.5GB	1	out1, out2, ...
`tranalyzer -r nudel.pcap -W out:5000K`	0.5MB	0	out0, out1, ...
`tranalyzer -r nudel.pcap -W out:5Kf`	5000 Flows	0	out0, out1, ...
`tranalyzer -r nudel.pcap -W out:180M`	180MB	0	out0, out1, ...
`tranalyzer -r nudel.pcap -W out:2.5G`	2.5GB	0	out0, out1, ...
`tranalyzer -r nudel.pcap -W out,6`	`OFRWFILELN`	0	out6, out7, ...
`tranalyzer -r nudel.pcap -W out`	`OFRWFILELN`	0	out0, out1, ...

2.4.8 –l

All Tranalyzer command line and report output is diverted to the log file: `PREFIX_log.txt`. Fatal error messages still appear on the command line.

2.4.9 –m

All Tranalyzer monitoring output is diverted to the monitoring file: `PREFIX_monitoring.txt`.

2.4.10 –s

Initiates the packet mode where a file with the suffix `PREFIX_packets.txt` is created. The content of the file depends on the plugins loaded. The display of the packet number (first column is controlled by `SPKTMD_PKTNO` in main.h. The payload can be displayed in hexadecimal and/or as characters by using the `SPKTMD_PCNTH` and `SPKTMD_PCNTC` respectively. The start of the payload (full packet, L2/3/4/7) to print is controlled by `SPKTMD_PCNTL`. A tab separated header description line is printed at the beginning of the packet file. The first two lines then read as follows:

```
%pktNo    time    pktIAT    duration    flowInd    flowStat    numHdrDesc    hdrDesc    vlanID
    ethType    srcMac    dstMac    srcIP4    srcPort    dstIP4    dstPort    l4Proto    ipTOS
    ipID    ipIDDiff    ipFrag    ipTTL    ipHdrChkSum    ipCalChkSum    l4HdrChkSum
    l4CalChkSum    ipFlags    pktLen    ipOptLen    ipOpts    seq    ack    seqDiff    ackDiff
    seqPktLen    ackPktLen    tcpStat    tcpFlags    tcpAnomaly    tcpWin    tcpOptLen    tcpOpts
       l7Content
...
25    1291753225.446846    0.000000    0.000000    23    0x00006000    6    eth:vlan:mpls{2}:ipv4:
    tcp    20    0x0800    00:11:22:33:44:55    66:77:88:99:aa:bb    X.Y.Z.U    62701    M.N.O.P
       80    6    0x00    0x26f6    0    0x4000    62    0x6ca6    0x6ca6    0x0247    0x0247    0
    x0040    460    0    0xb2a08909    0x90314073    0    0    0    0    0x59    0x18    0
    x0000    65535    12    0x01 0x01 0x08 0x0a 0x29 0x2d 0xc3 0x6e 0x83 0x63 0xc5 0x76    GET /
    images/I/01TnJ0+mhnL.png HTTP/1.1\r\nHost: ecx.images-amazon.com\r\nUser-Agent: Mozilla/5.0 (
    Macintosh; U; Intel Mac OS X 10.6; de; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8\r\nAccept:
    image/png,image/*;q=0.8,*/*;q=0.5\r\nAccept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3\r\
    nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\nKeep-Alive
    : 115\r\nConnection: keep-alive\r\nReferer: http://z-ecx.images-amazon.com/images/I/11
    J5cf408UL.css\r\n\r\n
...
```

2.4.11 –p FOLDER

Changes the plugin folder from standard *~/.tranalyzer/plugins* to `FOLDER`.

2.4.12 –b FILE

Changes the plugin blacklist file from *plugin_folder/plugin_blacklist.txt* to `FILE`, where `plugin_folder` is either *~/.tranalyzer/plugins* or the folder specified with the `-p` option.

2.4.13 –e FLOWINDEXFILE

Denotes the filename and path of the flow index file when the `pcapd` plugin is loaded. The path and name of the pcap file depends on `FLOWINDEXFILE`. If omitted the default names for the PCAP file are defined in *pcapd.h*. The format of the `FLOWINDEXFILE` is a list of 64 bit flow indices which define the packets to be extracted from the pcap being read by the `-r` option. In general the user should use a plain file with the format displayed below:

```
# Comments (ignored)
% Flow file info (ignored)
30
3467
656697
5596
```

For more information on the `pcapd` plugin please refer to its documentation.

2.4.14 –f HASHFACTOR

Sets and supersedes the `HASHFACTOR` constant in tranalyzer.h.

2.4.15 –x SENSORID

Each T2 can have a separate sensor ID which can be listed in a flow file in order to differentiate flows originating from several interfaces during post processing, e.g., in a DB. If not specified `T2_SENSORID` (`666`), defined in tranalyzer.h, will be the default value.

2.4.16 –c CPU

Bind Tranalyzer to core number `CPU`; if `CPU == 0` then the operating system selects the core to bind.

2.4.17 –P PRIO

Set Tranalyzer priority to `PRIO` instead of 0 (int). `PRIO` MUST belong in [-20, 20] for root and in [0, 20] for standard users. The lower the number, the higher the priority, i.e., -20 has higher priority than 20.

2.4.18 –M FLT

Set monitoring interval to `FLT` seconds.

2.4.19 –F FILE

Read BPF filter from FILE. A filter can span multiple lines and can be commented using the `'#'` character (everything following a `'#'` is ignored).

2.4.20 BPF Filter

A Berkeley Packet Filter (BPF) can be specified at any time in order to reduce the amount of flows being produced and to increase speed during life capture ops. All rules of pcap BPF apply.

2.5 hashTable.h

Name	Default	Description
`T2_HASH_FUNC`	10	Hash function to use: 0: standard 1: Murmur3 32-bits 2: Murmur3 128-bits (truncated to 64-bits) 3: xxHash 32-bits 4: xxHash 64-bits 5: XXH3 64-bits 6: XXH3 128-bits (truncated to lower 64-bits) 7: CityHash64 8: MUM-hash version 3 64-bits 9: hashlittle 32-bits 10: wyhash 64-bits

Name	Default	Description
		11: FastHash32 12: FastHash64 13: t1ha0 (Linux only) [meson build backend only] 14: t1ha2 [meson build backend only]
HASHTABLE_DEBUG	0	Print debug information
HASHTABLE_NAME_LEN	7	Maximum length of a hashTable's name

2.6 ioBuffer.h

Name	Default	Description
IO_BUFFERING	0	Enables buffering of the packets in a queue
If IO_BUFFERING == 1, the following flags are available:		
IO_BUFFER_FULL_WAIT_MS	200	Number of milliseconds to wait if queue is full
IO_BUFFER_SIZE	8192	Maximum number of packets that can be stored in the buffer (power of 2)
IO_BUFFER_MAX_MTU	2048	Maximum size of a packet (divisible by 4)

2.7 loadPlugins.h

Name	Default	Description
USE_PLLIST	1	Behavior of -b option (plugin loading list): 0: disable -b option and load all plugins from the plugin folder 1: only load plugins present in the list (whitelist) 2: do not load plugins present in the list (blacklist)

2.8 main.h

Name	Default	Description	Flags
The following four flags apply to the packet mode (-s option):			
SPKTMD_PKTNO	1	Print the packet number	
SPKTMD_PCNTC	1	Print packet payload as characters	
SPKTMD_PCNTH	0	Print packet payload as hex	
SPKTMD_PCNTL	4	Content field: 0: Print the full payload of the packet 1: Print payload from L2 2: Print payload from L3 3: Print payload from L4	

Name	Default	Description	Flags
		4: Print payload from L7	
SPKTMD_BOPS	0x00	Bit operations on content: 0x00: MSB, no bit inverse, no shift 0x01: LSB, bit inverse 0x02: Nibble swap 0x10: Shift right 0x20: shift from last byte into extra trailing byte (requires SPKTMD_BOPS & 0x10)	
SPKTMD_BSHFT_POS	5	Shift SPKTMD_BSHFT_POS-1 at 8-SPKTMD_BSHFT into following bytes	
SPKTMD_BSHFT	2	Bitshift	SPKTMD_BOPS&0x10
SPKTMD_PCNTH_PREF	"0x"	Prefix to add to every byte in packet mode as hex ("" → ab cd instead of 0xab 0xcd)	SPKTMD_PCNTH=1
SPKTMD_PCNTH_SEP	" "	Byte separator in packet mode as hex (",", → 0xab,0xcd instead of 0xab 0xcd)	SPKTMD_PCNTH=1
MIN_MAX_ESTIMATE	0	Min/Max bandwidth statistics	
MMXLAGTMS	0.1	Min Max interval [s]	MIN_MAX_ESTIMATE=1
MMXNO0	0	Suppress 0 in MIN estimation	MIN_MAX_ESTIMATE=1
The following flags control the monitoring mode:			
MONINTTHRD	1	Activate threaded interrupt handling	
MONINTBLK	0	Block interrupts in main loop during packet processing (disables MONINTTHRD)	
MONINTPSYNC	1	0: Interrupt printing, 1: pcap main loop synchronized printing	
MONINTTMPCP	0	0: real time base, 1: pcap time base	
MONINTTMPCP_ON	0	Startup monitoring. 0: off, 1: on	MONINTTMPCP=1
MONINTV	1.0	≥ 1 sec interval of monitoring output if USR2 is sent or MONINTTMPCP=0	
POLLENV	0	Change monitoring interval via env var $T2MTIME	
MONPROTMD	1	0: report protocol numbers, 1: report protocol names	
MONPROTFL	"proto.txt"	proto file	
The following flags control the DPDK multi-process mode:			
DPDK_MP	0	Use DPDK multi-process mode instead of libpcap	

The MONPROTL2 and MONPROTL3 flags can be used to configure the L2 and L3 protocols to monitor. Their default values are

- MONPROTL2: ETHERTYPE_ARP,ETHERTYPE_RARP
- MONPROTL3: L3_TCP,L3_UDP,L3_ICMP,L3_ICMP6,L3_SCTP

2.9 networkHeaders.h

Name	Default	Description	Flags
IPV6_ACTIVATE	2	0: IPv4 only 1: IPv6 only 2: Dual mode	
ETH_ACTIVATE	1	0: No Ethernet flows 1: Activate Ethernet flows generation 2: Also use Ethernet addresses for IPv4/6 flows	
LAPD_ACTIVATE	0	0: No LAPD/Q.931 flows 1: Activate LAPD/Q.931 flow generation	
LAPD_OVER_UDP	0	0: Do not try dissecting LAPD over UDP 1: Dissect LAPD over UDP (experimental)	LAPD_ACTIVATE=1
SCTP_ACTIVATE	0	1: standard flows 1: activate SCTP chunk streams → flow 2: activate SCTP association → flow 3: activate SCTP chunk & association → flow	
SCTP_STATFINDEX	0	1: findex constant for all SCTP streams in a packet 0: findex increments	SCTP_ACTIVATE=1
MULTIPKTSUP	0	Multi-packet suppression (discard duplicated packets)	IPV6_ACTIVATE=0
T2_PRI_HDRDESC	1	1: keep track of the headers traversed	
T2_HDRDESC_AGGR	1	1: aggregate repetitive headers, e.g., vlan{2}	T2_PRI_HDRDESC=1
T2_HDRDESC_LEN	128	max length of the headers description	T2_PRI_HDRDESC=1

2.10 proto/capwap.h

Name	Default	Description	Flags
CAPWAP_SWAP_FC	1	Swap CAPWAP frame control (required for Cisco)	CAPWAP=1

2.11 proto/ethertype.h

Name	Default	Description	Flags
PW_ETH_CW	1	Detect Pseudowire (PW) Ethernet Control Word (Heuristic, experimental)	

2.12 proto/linktype.h

Name	Default	Description	Flags
LINKTYPE_JUNIPER	1	Dissect PCAP with Juniper linktypes (Experimental)	

2.13 proto/lwapp.h

Name	Default	Description	Flags
LWAPP_SWAP_FC	1	Swap LWAPP frame control (required for Cisco)	LWAPP=1

2.14 packetCapture.h

The config file *packetCapture.h* provides control about the packet capture and packet structure process of Tranalyzer2. The most important fields are described below. Please note that after changing any value in define statements a rebuild is required. Note that the PACKETLENGTH switch controls the len variable in the packet structure, from where the packet length is measured from. So statistical plugins such as basicStats can have a layer dependent output. If only L7 length is needed, use the l7Len variable in the packet structure.

Name	Default	Description	Flags
PACKETLENGTH	3	0: including L2, L3 and L4 header 1: including L3 and L4 header 2: including L4 header 3: only higher layer payload (Layer 7)	
FRGIPPKTLENVIEW	1	0: IP header stays with 2nd++ fragmented packets 1: IP header stripped from 2nd++ fragmented packets	PACKETLENGTH=1
NOLAYER2	0	0: Automatic L3 header discovery 1: Manual L3 header positioning	
NOL2_L3HDROFFSET	0	Offset of L3 header	NOLAYER2=1
MAXHDRCNT	5	Maximal header count (MUST be ≥ 3)	IPV6_ACTIVATE=1
SALRM	0	1: enable sending FL_ALARM bit for pcapd	
SALRMINV	0	1: invert selection	SALRM=1

2.15 tranalyzer.h

Name	Default	Description	Flags
T2_SENSORID	666	Sensor ID (can be overwritten with t2 -x option)	
ENVCNTRL	2	Plugins configuration mode: 0: Values from header file during compilation 1: Values from header file at runtime 2: Values from environment if defined, otherwise from header file at runtime	
REPSUP	0	Activate alive mode	
PID_FNM_ACT	0	Save the PID into a file PID_FNM (default: "tranalyzer.pid")	

Name	Default	Description	Flags
PKT_CB_STATS	0	Compute stats about time spent in perPacketCallback()	
DEBUG	0	0: no debug output 1: debug output which occurs only once or very seldom 2: + debug output which occurs in special situations, but not regularly 3: + debug output which occurs regularly (every packet)	
VERBOSE	2	0: no output 1: basic pcap report 2: + full traffic statistics 3: + info about fragmentation anomalies	
MEMORY_DEBUG	0	0: no memory debug 1: detect leaks and overflows (see *utils/memdebug.h*)	
NO_PKTS_DELAY_US	1000	If no packets are available, sleep for *n* microseconds	
NON_BLOCKING_MODE	1	Non-blocking mode	
MAIN_OUTBUF_SIZE	1000000	Size of the main output buffer	
SNAPLEN	BUFSIZ	Snapshot length (live capture)	
CAPTURE_TIMEOUT	1000	Read timeout in milliseconds (live capture)	
BPF_OPTIMIZE	0	1: Optimize BPF filters	
TSTAMP_PREC	1	Timestamp precision: 0: microseconds, 1: nanoseconds	
TSTAMP_UTC	1	Time representation: 0: localtime, 1: UTC	
TSTAMP_R_UTC	0	Time report representation: 0: localtime, 1: UTC	
ALARM_MODE	0	Only output flow if an alarm-based plugin fires	
ALARM_AND	0	0: logical OR, 1: logical AND	ALARM_MODE=1
FORCE_MODE	0	Parameter induced flow termination (NetFlow mode)	
BLOCK_BUF	0	Block unnecessary buffer output when non Tranalyzer format event based plugins are active	
USE_T2BUS	0	Use t2Bus communication backend (experimental)	
PLUGIN_REPORT	1	Enable plugins to contribute to Tranalyzer end report	
DIFF_REPORT	0	0: absolute Tranalyzer command line USR1 report 1: differential report	
MACHINE_REPORT	0	USR1 report: 0: human compliant, 1: machine compliant	
REPORT_HIST	0	Store statistical report history in REPORT_HIST_FILE after shutdown and reload it when restarted	
ESOM_DEP	0	Allow plugins to globally access other plugins variables	
AYIYA	1	Process AYIYA	
GENEVE	1	Process GENEVE	
TEREDO	1	Process TEREDO	
L2TP	1	Process L2TP	
GRE	1	Process GRE	
GTP	1	Process GTP (GPRS Tunneling Protocol)	
VXLAN	1	Process VXLAN	
IPIP	1	Process IPv4/6 in IPv4/6	
ETHIP	1	Process Ethernet within IP	
CAPWAP	1	Process CAPWAP	

Name	Default	Description	Flags
LWAPP	1	Process LWAPP	
DTLS	1	Process DTLS	
FRAGMENTATION	1	Activate fragmentation processing	
FRAG_HLST_CRFT	1	Enables crafted packet processing	FRAGMENTATION=1
FRAG_ERROR_DUMP	0	Dumps flawed fragmented packet to stdout	FRAGMENTATION=1
IPVX_INTERPRET	0	Interpret bogus IPvX packets	
ANONYM_IP	0	1: No output of IP information	
ETH_STAT_MODE	0	0: use the innermost layer 2 type for the statistics 1: use the outermost layer 2 type for the statistics	
SUBNET_ON	1	Core control of subnet function for plugins	
RELTIME	0	0: absolute time 1: relative time	
FDURLIMIT	0	If > 0, force flow life span to $n \pm 1$ seconds	
FDLSFINDEX	0	Findex for flows of a superflow 0: different findex 1: same findex	FDURLIMIT=1
FLOW_TIMEOUT	182	Flow timeout after a packet is not seen after n seconds	
NOFLWCRT	1	SIGINT 1 create no flow, SIGINT 2 release all flows, end report	
ZPKTITMUPD	1	0: update if packets received 1: Zero Pkt actTime update active	
ZPKTTMO	1500	Number of loops until actTime update	ZPKTITMUPD=1
HASHFACTOR	1	default multiplication factor for HASHTABLE_BASE_SIZE (can be overwritten with -f option)	
HASH_CHAIN_FACTOR	2	default multiplication factor for HASHCHAINTABLE_BASE_SIZE	
HASH_AUTOPILOT	1	When main hash map is full: 0: terminate 1: flushes oldest NUMFLWRM flow(s)	
NUMFLWRM	1	Number of flows to flush when main hash map is full	HASH_AUTOPILOT=1

Note that the PLUGIN_FOLDER flag (".tranalyzer/plugins/") can be either set in this file or set at runtime with t2 -p option.
Although not recommended, the suffix for the generated files can also be changed by editing the PACKETS_SUFFIX ("_packets.txt"), LOG_SUFFIX ("_log.txt") and MON_SUFFIX ("_monitoring.txt") flags.

2.15.1 -D constants

the following constants influence the file name convention:

Name	Default	Description
RROP	0	round robin operations

Name	Default	Description
POLLTM	5	poll timing for files
SCHR	'p'	separating character for file number

2.15.2 alive signal

The alive signal is a derivative of the passive monitoring mode by the USR1 signal, where the report is deactivated. If REPSUP=1 then only the command defined by REPCMDAS/W is sent to the control program defined by ALVPROG as defined below:

Name	Default	Description
REPSUP	0	0: alive mode off, 1: alive mode on, monitoring report suppressed
ALVPROG	"t2alive"	name of control program
REPCMDAS	"a=`pgrep " ALVPROG "`; \ if [$a]; then kill -USR1 $a; fi"	alive and stall USR1 signal (no packets)
REPCMDAW	"a=`pgrep " ALVPROG "`; \ if [$a]; then kill -USR2 $a; fi"	alive and well USR2 signal (working)

If T2 crashes or is stopped a syslog message is issued by the t2alive daemon. Same if T2 gets started.

2.15.3 FORCE_MODE

A 1 enables the force mode which enables any plugin to force the output of flows independent of the timeout value. Hence, Cisco NetFlow similar periodic output can be produced or overflows of counters can produce a flow and restart a new one. The macro which has to be present in the t2OnLayer4() function is shown below:

```
T2_RM_FLOW(flowP);
```

Figure 1: *Force code line in the* t2OnFlowTerminate() *plugin routine*

2.15.4 ALARM_MODE

A 1 enables the alarm mode which differs from the default flow mode by the plugin based control of the Tranalyzer core flow output. It is useful for classification plugins generating alarms, thus emulating alarm based SW such as Snort, etc. The default value is 0. The plugin sets the global output suppress variable supOut=1 in the t2OnFlowTerminate() function before any output is generated. This mode also allows multiple classification plugins producing an 'AND' or an 'OR' operation if many alarm generating plugins are loaded. The variable ALARM_AND controls the logical alarm operation. The macro which has to be present in the t2OnFlowTerminate() function is shown below:

```
T2_REPORT_ALARMS(tcpWinFlowP->winThCnt);
```

Figure 2: *Alarm code line in the* t2OnFlowTerminate() *plugin routine*

2.15.5 BLOCK_BUF

if set to '1' unnecessary buffered output from all plugins is blocked when non Tranalyzer format event based plugins are active, e.g., text-based or binary output plugins are not loaded.

2.15.6 Report Modes

Tranalyzer provides a user interrupt based report and a final report. The interrupt based mode can be configured in a variety of ways being defined below.

Name	Default	Description
PLUGIN_REPORT	0	enable plugins to contribute to the tranalyzer command line end report
DIFF_REPORT	0	1: differential, 0: Absolute tranalyzer command line USR1 report
MACHINE_REPORT	0	USR1 Report 1: machine compliant; 0: human compliant

The following interrupts are being caught by Tranalyzer2:

Signal Name	Description
SIGINT	like ^C terminates new flow production[4]
SIGTERM	terminates tranalyzer
SIGUSR1	prints statistics report
SIGUSR2	toggles repetitive statistics report

2.15.7 State and statistical save mode

T2 is capable to preserve its internal statistical state and certain viable global variables, such as the findex.

Name	Default	Description
REPORT_HIST	0	Store statistical report history after shutdown, reload it upon restart
REPORT_HIST_FILE	"stat_hist.txt"	default statistical report history filename

The history file is stored by default under ./tranalyzer/plugins or under the directory defined by a -p option.

2.15.8 L2TP

A '1' activates the L2TP processing of the Tranalyzer2 core. All L2TP headers either encapsulated in MPLS or not will be processed and followed down via PPP headers to the IP header and then passed to the IP processing. The default value of the variable is '0'. Then the stack will be parsed until the first IP header is detected. So all L2TP UDP headers having source and destination port 1701 will be processed as normal UDP packets.

2.15.9 GRE

A '1' activates the L3 General Routing Encapsulation (GRE, l4proto=47) processing of the Tranalyzer2 core. All GRE headers either encapsulated in MPLS or not will be processed and followed down via PPP headers to the IP header and then passed to the IP processing. The default value of the variable is 0. Then the stack will be parsed until the first IP header is detected. If the following content is not existing or compressed the flow will contain only l4Proto=47 information.

[4] If two SIGINT interrupts are being sent in short order Tranalyzer will be terminated instantly.

2.15.10 FRAGMENTATION

A '1' activates the fragmentation processing of the Tranalyzer2 core. All packets following the header packet will be assembled in the same flow. The core and the plugin `tcpFlags` will provide special flags for fragmentation anomalies. If `FRAGMENTATION` is set to 0 only the initial fragment will be processed; all later fragments will be ignored.

2.15.11 FRAG_HLST_CRFT

A '1' enables crafted packet processing even when the lead fragment is missing or packets contain senseless flags as being used in attacks or equipment failure.

2.15.12 FRAG_ERROR_DUMP

A '1' activates the dump of packet information on the command line for time based identification of ill-fated or crafted fragments in tcpdump or Wireshark. It provides the Unix timestamp, the six tuple, IPID and fragID as outlined in figure below.

```
MsgType   msg   time   vlan   srcIP   srcPort   dstIP   dstPort   proto   fragID   fragOffset
[WRN] packetCapture: 1. frag not found @ 1291753225.449690 20 X.Y.Z.U 42968 M.N.O.P 52027 17 - 0
    x191F 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.482611 20 X.Y.Z.U 43044 M.N.O.P 1719 17 - 0
    x1922 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.492830 20 X.Y.Z.U 55841 M.N.O.P 28463 17 - 0
    x1923 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.503955 20 X.Y.Z.U 25668 M.N.O.P 8137 17 - 0
    x1924 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.551094 20 X.W.Z.T 41494 T.V.W.Z 27796 17 - 0
    x5A21 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.639627 20 A.B.C.D 38824 E.F.G.H 55133 17 - 0
    x0DAE 0x00AC
```

Figure 3: *A sample report on stdout for packets with an elusive first fragment*

WARNING: If `FRAG_HLST_CRFT == 1` then every fragmented headerless packet will be reported!

2.15.13 *_SUFFIX

This constant defines the suffix of all plugin output files. For example if you specify the output *foo.foo* (with the `-w` option), the generated file for the per-packet output will be in the default setting *foo.foo_packets*.

2.15.14 RELTIME

`RELTIME` renders all time based plugin output into relative to the beginning of the pcap or start of packet capture. In `-D` or `-R` read operation the first file defines the start time.

2.15.15 FLOW_TIMEOUT

This constant specifies the default time in seconds (182) after which a flow will be considered as terminated since the last packet is captured. Note: Plugins are able to change the timeout values of a flow. For example the `tcpStates` plugin adjusts the timeout of a flow according to the TCP state machine. A reduction of the flow timeout has an effect on the necessary flow memory defined in `HASHCHAINTABLE_SIZE`, see below.

2.15.16 FDURLIMIT

`FDURLIMIT` defines the maximum flow duration in seconds which is then forced to be released. It is a special force mode for the duration of flows and a special feature for Dalhousie University. If `FDURLIMIT > 0` then `FLOW_TIMEOUT` is overwritten if `FURLIMIT` seconds are reached.

2.15.17 HASHFACTOR

A factor to be multiplied with the `HASHTABLE_SIZE` described below. It facilitates the correct setting of the hash space. Moreover, if T2 runs out of hash it will give an upper estimate the user can choose for `HASHFACTOR`. Set it to this value, recompile and rerun T2. This constant is superseded by the `-f` option.

2.15.18 HASHTABLE_SIZE

The number of buckets in the hash table. As a separate chaining hashing method is used, this value does not denote the amount of elements the hash table is able to manage! The larger, the less likely are hash collisions. The current default value is 2^{18}. Its value should be selected at least two times larger as the value of `HASHCHAINTABLE_SIZE` discussed in the following chapter.

2.15.19 HASH_CHAIN_FACTOR

A factor to be multiplied with the `HASHCHAINTABLE_SIZE` described below.

2.15.20 HASHCHAINTABLE_SIZE

Specifies the amount of flows the main hash table is able to manage. The default value is 2^{19}, so roughly half the size of `HASHTABLE_SIZE`. T2 supplies information about the hash space in memory in: `Max number of IPv4 flows in memory: 113244 (50.220%)`. Together with the amount of traffic already processed the total value can be computed. An example is given in Figure 1.

2.15.21 HASH_AUTOPILOT

Default 1. Avoids overrun of main hash, flushes oldest flow on every flow insert if hash map is full. 0 disables hash overrun protection. If speed is an issue avoid overruns by invoking T2 with the `-f` option set to the value recommended by T2.

2.15.22 SUBNET_ON

Since the version 0.8.8 the core controls the subnet functions instead of `basicFlow` as now the aggregation mode according to countries or organization is possible.

2.15.23 utils.h

The following flags can be used to configure the subnet files and the output of the plugins:

Name	Default	Description
`SUBRNG`	0	Subnet definition: 0: CIDR only 1: Begin-End

Name	Default	Description
CNTYCTY	0	1: Add the county and city
CNTYLEN	14	length of County record
CTYLEN	14	length of City record
WHOLEN	27	length of Organization record

If any of those flags is changed, `tranalyzer` **MUST** be recompiled with `t2build -f` in order to generate a new binary subnet file, as we only load the info the user needs.

2.15.24 Format of the Subnet Files

The text format of the `subnets4.txt` and `subnets6.txt` files is defined as follows:

- A `'-'` in the first column (`prefix/mask`) denotes a non-CIDR range. In this case, Tranalyzer reads the 2nd column instead of the 1st when `SUBRNG=1` in utils.h.
- If `SUBRNG=0`, the 2nd column is ignored and only CIDR ranges are accepted.
- Country and Whois 32 bit hex code: `cccc cccc cTww wwww wwww wwww wwww wwww` where `c`: Country, `T` Tor address bit, `w`: Organization
- Autonomous System Number (ASN)
- Uncertainty of location in km
- Latitude
- Longitude
- Country
- County
- City
- Organization

An extraction of `subnets4.txt` is depicted below:
The text files `subnets4.txt` and `subnets6.txt` can be edited and manually converted, just move to `utils/subnet` directory and invoke the following command:

`./subconv subnets4.txt` and `./subconv subnets6.txt`

2.15.25 Tor Information

Since 0.8.8 Tor information is also available for IPv6. The conversion programs from the raw files to T2 format can be found under `utils/subnet/tor`. It can be controlled also by `subconv`, see options below:

```
#       5       16122020
# IPCIDR        IPrange CtryWhoCode     ASN     Uncert  Latitude        Longitude       Country
    County  City    Org
# Begin IPv4 private address space
10.0.0.0/8      10.0.0.0-10.255.255.255 0x0301c2a7      0       -1.0    666.000000      666.000000
          04      -       -       Private network
14.0.0.0/8      14.0.0.0-14.255.255.255 0x00000000      0       -1.0    666.000000      666.000000
          03      -       -       Public data networks
24.0.0.0/8      24.0.0.0-24.255.255.255 0x00000000      0       -1.0    666.000000      666.000000
          09      -       -       Cable television networks
127.0.0.0/8     127.0.0.0-127.255.255.255       0x01014fe7      0       -1.0    666.000000
    666.000000      01      -       -       Loopback
100.64.0.0/10   100.64.0.0-100.127.255.255       0x0702041f      0       -1.0    666.000000
    666.000000      20      -       -       Shared address space
169.254.0.0/16  169.254.0.0-169.254.255.255     0x02014965      0       -1.0    666.000000
    666.000000      02      -       -       Link-local
172.16.0.0/12   172.16.0.0-172.31.255.255        0x0381c2a7      0       -1.0    666.000000
    666.000000      05      -       -       Private network
192.0.0.0/24    192.0.0.0-192.0.0.255   0x0401c2a7      0       -1.0    666.000000      666.000000
          06      -       -       Private network
192.0.2.0/24    192.0.2.0-192.0.2.255   0x07823fb8      0       -1.0    666.000000      666.000000
          21      -       -       TEST-NET-1
192.88.99.0/24  192.88.99.0-192.88.99.255       0x0b011c29      0       -1.0    666.000000
    666.000000      60      -       -       IPv6 to IPv4 relay
192.168.0.0/16  192.168.0.0-192.168.255.255     0x0481c2a7      0       -1.0    666.000000
    666.000000      07      -       -       Private network
198.18.0.0/15   198.18.0.0-198.19.255.255        0x0501c2a7      0       -1.0    666.000000
    666.000000      08      -       -       Private network
198.51.100.0/24 198.51.100.0-198.51.100.255     0x08023fb9      0       -1.0    666.000000
    666.000000      22      -       -       TEST-NET-2
203.0.113.0/24  203.0.113.0-203.0.113.255       0x08823fba      0       -1.0    666.000000
    666.000000      23      -       -       TEST-NET-3
224.0.0.0/4     224.0.0.0-239.255.255.255       0x06017598      0       -1.0    666.000000
    666.000000      10      -       -       Multicast
...
240.0.0.0/4     240.0.0.0-255.255.255.254       0x0901dd04      0       -1.0    666.000000
    666.000000      24      -       -       Reserved
255.255.255.255/32      255.255.255.255-255.255.255.255 0x06804ca0      0       -1.0    666.000000
          666.000000      11      -       -       Broadcast
# End IPv4 private address space
1.0.0.0/24      1.0.0.0-1.0.0.255       0x8480205e      13335   80.000000       34.052231
    -118.243683     us      California      Los Angeles     APNIC Research and Development
1.0.1.0/24      1.0.1.0-1.0.1.255       0x260062cc      0       80.000000       26.061390
    119.306107      cn      Fujian  Fuzhou  CHINANET FUJIAN PROVINCE NETWORK
1.0.2.0/23      1.0.2.0-1.0.3.255       0x260062cc      0       80.000000       26.061390
    119.306107      cn      Fujian  Fuzhou  CHINANET FUJIAN PROVINCE NETWORK
1.0.4.0/24      1.0.4.0-1.0.4.255       0x148284d8      56203   80.000000       -37.813999
    144.963318      au      Victoria        Melbourne       Wirefreebroadband Pty Ltd
...
```

```
$ ./subconv -h
Usage:
    subconv [OPTION...] <subnets.txt>

Optional arguments:
    -4                Generate subnet file for IPv4
```

```
-6                Generate subnet file for IPv6

-t                Add Tor info to subnet file
-a                Download and add Tor info to subnet file
-c                Convert from Tor JSON info to subnet file

-h, --help        Show this help, then exit
```

So to convert the IPv4 subnet file and add existing Tor info invoke the following command:

```
./subconv -t subnets4.txt
```

2.15.26 Aggregation Mode

The aggregation mode enables the user to confine certain IP, port or protocol ranges into a single flow. The variable AGGREGATIONFLAG in tranalyzer.h defines a bit field which enables specific aggregation modes according to the six tuple values listed below.

Aggregation Flag	Value
L4PROT	0x01
DSTPORT	0x02
SRCPORT	0x04
DSTIP	0x08
SRCIP	0x10
VLANID	0x20
SUBNET	0x80

If a certain aggregation mode is enabled the following variables in tranalyzer.h define the aggregation range.

Aggregation Flag	Type	Description
SRCIP4CMSK	uint8_t	src IPv4 aggregation CIDR mask
DSTIP4CMSK	uint8_t	dst IPv4 aggregation CIDR mask
SRCIP6CMSK	uint8_t	src IPv6 aggregation CIDR mask
DSTIP6CMSK	uint8_t	dst IPv6 aggregation CIDR mask
SRCPORTLW	uint16_t	src port lower bound
SRCPORTHW	uint16_t	src port upper bound
DSTPORTLW	uint16_t	dst port lower bound
DSTPORTHW	uint16_t	dst port upper bound

If SUBNET is chosen then all flows are aggregated according to a 32-bit hex subnet mask of the organization country hex code.

```
// SUBNET mode: IP flow aggregation network masks
#define CNTRY_MSK 0xff800000
#define TOR_MSK   0x00400000
#define ORG_MSK   0x003fffff
#define NETIDMSK  (CNTRY_MSK | ORG_MSK) // netID mask
```

2.16 bin2txt.h

Name	Default	Description
IP4_FORMAT	0	IPv4 addresses representation: 0: normal, 1: normalized (padded with zeros), 2: one 32-bits hex number 3: one 32-bits unsigned number
IP6_FORMAT	0	IPv6 addresses representation: 0: compressed, 1: uncompressed, 2: one 128-bits hex number, 3: two 64-bits hex numbers
MAC_FORMAT	0	MAC addresses representation: 0: normal (edit MAC_SEP to change the separator), 1: one 64-bits hex number,
MAC_SEP	":"	Separator to use in MAC addresses: 11:22:33:44:55:66
B2T_NON_IP_STR	"-"	Representation of non-IPv4/IPv6 addresses in IP columns
HEX_CAPITAL	0	Hex output: 0: lower case; 1: upper case
TFS_EXTENDED_HEADER	0	Extended header in flow file
TFS_NC_TYPE	2	Types in header file: 0: none, 1: numbers, 2: C types
TFS_SAN_UTF8	1	Activates the UTF-8 sanitizer for strings
B2T_TIMESTR	0	Print Unix timestamps as human readable dates
HDR_CHR	"%"	start character(s) of comments
SEP_CHR	"\t"	column separator in the flow file ";", ".", "_" and "\"" should not be used
JSON_KEEP_EMPTY	0	Output empty fields
JSON_PRETTY	0	Add spaces to make the output more readable

2.17 gz2txt.h

Name	Default	Description
USE_ZLIB	1	Activate code for gzip-(de)compression

2.18 outputBuffer.h

Name	Default	Description	Flags
BUF_DATA_SHFT	0	Adds for each binary output record the length and shifts the record by *n* uint32_t words to the right (see binSink and socketSink plugin)	
OUTBUF_AUTOPILOT	1	Automatically increase the output buffer when required	
OUTBUF_MAXSIZE_F	5	Maximal factor to increase the output buffer size to	OUTBUF_AUTOPILOT=1

2.19 rbTree.h

Name	Default	Description
RBT_DEBUG	0	Enable debug output
RBT_ROTATION	0	Activate the Red-Black Tree feature of rotating an unbalanced tree

2.20 subnetHL.h

Name	Default	Description
SUBRNG	0	IP range definition: 0: CIDR only, 1: Begin-End
CNTYCTY	0	Output the county and the city
WHOADDR	0	Add whois address info
SUB_MAP	1	Use mmap to load the subnet file
CNTYLEN	14	Length of County record
CTYLEN	14	Length of City record
WHOLEN	30	Length of Organization record
ADDRLEN	30	Length of Address record
SUBNET_UNK	"-"	Representation of unknown locations

2.21 t2log.h

Name	Default	Description
T2_LOG_COLOR	1	Whether or not to color messages

2.22 Tranalyzer2 Output

As stated before, the functionality and output of Tranalyzer2 is defined by the activated plugins. Basically, there are two ways a plugin can generate output. First, it can generate its own output file and write any arbitrary content into any stream. The second way is called standard output or per-flow output. After flow termination Tranalyzer2 provides an output buffer and appends the direction of the flow to it. For example, in case of textual output, an "A" flow is normally followed by a "B" flow or if the "B" flow does not exist it is followed by the next "A" flow. Then, the output buffer is passed to the plugins providing their per-flow output. Finally the buffer is sent to the activated output plugins. This process repeats itself for the "B" flow. For detailed explanation about the functionality of the output plugins refer to the section plugins.

2.22.1 Hierarchical Ordering of Numerical or Text Output

Tranalyzer2 provides a hierarchical ordering of each output. Each plugin controls the:

- volume of its output
- number of values or bins

- hierarchical ordering of the data
- repetition of data substructures

Thus, complex structures such as lists or matrices can be presented in a single line.
The following sample of text output shows the hierarchical ordering for four data outputs, separated by tabulators:

```
A    0.3    2.0_3.4_2.1    2;4;2;1    (1_2_9)_(1_3_1)_(7_5_3)_(2_3_7)
```

The `A` indicates the direction of the flow; in this case it is the initial flow. The next number denotes a singular descriptive statistical result. Output number two consists of three values separated by "_" characters. Output number three consists of one value, that can be repeated, indicated by the character ";". Output number four is a more complex example: It consists of four values containing three subvalues indicated by the braces. This could be interpreted as a matrix of size 4×3.

2.23 Final Report

Standard configuration of Tranalyzer2 produces a statistical report to *stdout* about timing, packets, protocol encapsulation type, average bandwidth, dump length, etc. A sample report including some current protocol relevant warnings is depicted in the figure below. Warnings are not fatal hence are listed at the end of the statistical report when Tranalyzer2 terminates naturally. The *Average total Bandwidth* estimation refers to the processed bandwidth during the data acquisition process. It is only equivalent to the actual bandwidth if the total packet length including all encapsulations is not truncated and all traffic is IP. The *Average IP Traffic Bandwidth* is an estimate comprising all IP traffic actually present on the wire. Plugins can report extra information when `PLUGIN_REPORT` is activated. This report can be saved in a file, by using one of the following command:

```
tranalyzer -r file.pcap -w out -l (See Section 2.4.8)
 tranalyzer -r file.pcap -w out | tee out_stdout.txt
   tranalyzer -r file.pcap -w out > out_stdout.txt
```

Both commands will create a file `out_stdout.txt` containing the report. The only difference between those two commands is that the first one still outputs the report to stdout.

Fatal errors regarding the invocation, configuration and operation of Tranalyzer2 are printed to *stderr* after the plugins are loaded, thus before the processing is activated, see the *Hash table error* example in Listing 1. These errors terminate Tranalyzer2 immediately and are located before the final statistical report as being indicated by the *"Shutting down..."* key phrase. If the final report is to be used in a following script a pipe can be appended and certain lines can be filtered using grep or awk.

```
$ ./tranalyzer -r ~/data/knoedel.pcap -w ~/results/
================================================================================
Tranalyzer 0.8.9 (Anteater), Tarantula. PID: 3426
================================================================================
[INF] Creating flows for L2, IPv4, IPv6
Active plugins:
    01: protoStats, 0.8.9
    02: basicFlow, 0.8.9
    03: macRecorder, 0.8.9
    04: portClassifier, 0.8.9
    05: basicStats, 0.8.9
    06: tcpFlags, 0.8.9
    07: tcpStates, 0.8.9
    08: icmpDecode, 0.8.9
    09: dnsDecode, 0.8.9
```

```
    10: httpSniffer, 0.8.9
    11: connStat, 0.8.9
    12: txtSink, 0.8.9
[INF] IPv4 Ver: 5, Rev: 16122020, Range Mode: 0, subnet ranges loaded: 406027 (406.03 K)
[INF] IPv6 Ver: 5, Rev: 17122020, Range Mode: 0, subnet ranges loaded: 50974 (50.97 K)
Processing file: /home/wurst/knoedel.pcap
Link layer type: Ethernet [EN10MB/1]
Dump start: 1291753225.446732 sec (Tue 07 Dec 2010 20:20:25 GMT)
[WRN] snapL2Length: 1550 - snapL3Length: 1484 - IP length in header: 1492
[WRN] Hash Autopilot: main HashMap full: flushing 1 oldest flow(s)
[INF] Hash Autopilot: Fix: Invoke Tranalyzer with '-f 5'
Dump stop : 1291753452.373884 sec (Tue 07 Dec 2010 20:24:12 GMT)
Total dump duration: 226.927152 sec (3m 46s)
Finished processing. Elapsed time: 171.835756 sec (2m 51s)
Finished unloading flow memory. Time: 182.101116 sec (3m 2s)
Percentage completed: 100.00%
Number of processed packets: 53982409 (53.98 M)
Number of processed bytes: 42085954664 (42.09 G)
Number of raw bytes: 42101578296 (42.10 G)
Number of pad bytes: 25023 (25.02 K)
Number of pcap bytes: 42949673232 (42.95 G)
Number of IPv4 packets: 53768017 (53.77 M) [99.60%]
Number of IPv6 packets: 214107 (214.11 K) [0.40%]
Number of A packets: 31005806 (31.01 M) [57.44%]
Number of B packets: 22976603 (22.98 M) [42.56%]
Number of A bytes: 14211017503 (14.21 G) [33.77%]
Number of B bytes: 27874937161 (27.87 G) [66.23%]
Average A packet load: 458.33
Average B packet load: 1213.19 (1.21 K)
--------------------------------------------------------------------------------
macRecorder: MAC pairs per flow: min: 1, max: 3, average: 1.00
basicStats: Biggest L2 talker: xx:xx:xx:xx:xx:xx: 68 [0.00%] packets
basicStats: Biggest L2 talker: xx:xx:xx:xx:xx:xx: 85480 (85.48 K) [0.00%] bytes
basicStats: Biggest L3 talker: v.v.v.v (GB): 154922 (154.92 K) [0.29%] packets
basicStats: Biggest L3 talker: w.w.w.w (GB): 236308004 (236.31 M) [0.56%] bytes
tcpFlags: Aggregated ipFlags: 0x3d6e
tcpFlags: Aggregated tcpAnomaly: 0xfe07
tcpFlags: Number of TCP scans attempted, successful: 88849 (88.85 K), 140408 (140.41 K) [158.03%]
tcpFlags: Number of TCP SYN retries, seq retries: 65369 (65.37 K), 15783 (15.78 K)
tcpFlags: Number WinSz below 1: 162725 (162.72 K) [0.38%]
tcpFlags: Number of MPTCP packets: 6 [0.00%]
tcpFlags: Aggregated MPTCP types: 0x0008 and flags: 0x00
tcpStates: Aggregated tcpStates anomalies: 0xdf
icmpDecode: Aggregated icmpStat: 0x31
icmpDecode: Number of ICMP echo request packets: 14979 (14.98 K) [12.00%]
icmpDecode: Number of ICMP echo reply packets: 2690 (2.69 K) [2.16%]
icmpDecode: ICMP echo reply / request ratio: 0.18
icmpDecode: Number of ICMPv6 echo request packets: 1440 (1.44 K) [9.63%]
icmpDecode: Number of ICMPv6 echo reply packets: 951 [6.36%]
icmpDecode: ICMPv6 echo reply / request ratio: 0.66
dnsDecode: Number of DNS packets: 237597 (237.60 K) [0.44%]
dnsDecode: Number of DNS Q packets: 125260 (125.26 K) [52.72%]
dnsDecode: Number of DNS R packets: 112337 (112.34 K) [47.28%]
dnsDecode: Aggregated dnsStat: 0xf72f
httpSniffer: Number of HTTP packets: 36623492 (36.62 M) [67.84%]
httpSniffer: Number of HTTP GET  requests: 426825 (426.82 K) [1.17%]
httpSniffer: Number of HTTP POST requests: 39134 (39.13 K) [0.11%]
httpSniffer: HTTP GET/POST ratio: 10.91
httpSniffer: Aggregated httpStat     : 0x003f
httpSniffer: Aggregated httpAFlags   : 0x5143
```

```
httpSniffer: Aggregated httpCFlags   : 0x007a
httpSniffer: Aggregated httpHeadMimes: 0x80ef
httpSniffer: Number of files img_vid_aud_msg_txt_app_unk: 192890_5262_401_95_149772_91634_1958
connStat: Number of unique source IPs: 275311 (275.31 K)
connStat: Number of unique destination IPs: 301003 (301.00 K)
connStat: Number of unique source/destination IPs connections: 1242 (1.24 K)
connStat: Max unique number of source IP / destination port connections: 1521 (1.52 K)
connStat: IP prtcon/sdcon, prtcon/scon: 1.224638, 0.005525
connStat: Source IP with max connections: X.Y.Z.U (HU): 1515 (1.51 K) connections
connStat: Destination IP with max connections: L.M.N.O (FI): 3241 (3.24 K) connections
--------------------------------------------------------------------------------
Headers count: min: 4, max: 13, average: 7.10
Max VLAN header count: 1
Max MPLS header count: 2
Number of LLC packets: 285 [0.00%]
Number of GRE packets: 285 [0.00%]
Number of Teredo packets: 213876 (213.88 K) [0.40%]
Number of AYIYA packets: 231 [0.00%]
Number of IGMP packets: 401 [0.00%]
Number of ICMP packets: 124800 (124.80 K) [0.23%]
Number of ICMPv6 packets: 14946 (14.95 K) [0.03%]
Number of TCP packets: 43273341 (43.27 M) [80.16%]
Number of TCP bytes: 36129271238 (36.13 G) [85.85%]
Number of UDP packets: 10311931 (10.31 M) [19.10%]
Number of UDP bytes: 5832263590 (5.83 G) [13.86%]
Number of IPv4 fragmented packets: 19155 (19.16 K) [0.04%]
Number of IPv6 fragmented packets: 9950 (9.95 K) [4.65%]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of processed   flows: 1438758 (1.44 M)
Number of processed A flows: 1209354 (1.21 M) [84.06%]
Number of processed B flows: 229404 (229.40 K) [15.94%]
Number of request     flows: 930585 (930.59 K) [64.68%]
Number of reply       flows: 508173 (508.17 K) [35.32%]
Total   A/B    flow asymmetry: 0.68
Total req/rply flow asymmetry: 0.29
Number of processed   packets/flows: 37.52
Number of processed A packets/flows: 25.64
Number of processed B packets/flows: 100.16
Number of processed total packets/s: 237884.31 (237.88 K)
Number of processed A+B   packets/s: 237884.31 (237.88 K)
Number of processed A     packets/s: 136633.30 (136.63 K)
Number of processed   B   packets/s: 101251.01 (101.25 K)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of average processed flows/s: 6340.18 (6.34 K)
Average full raw bandwidth: 1484232320 b/s (1.48 Gb/s)
Average snapped bandwidth : 1483681536 b/s (1.48 Gb/s)
Average full bandwidth : 1483899136 b/s (1.48 Gb/s)
Max number of flows in memory: 262144 (262.14 K) [100.00%]
Number of flows terminated by autopilot: 815749 (815.75 K) [56.70%]
Memory usage: 2.49 GB [3.70%]
Aggregate flow status: 0x0c00bcfad298fb04
[WRN] L3 SnapLength < Length in IP header
[WRN] L4 header snapped
[WRN] Consecutive duplicate IP ID
[WRN] IPv4/6 payload length > framing length
[WRN] IPv4/6 fragmentation header packet missing
[WRN] IPv4/6 packet fragmentation sequence not finished
[INF] Ethernet flows
[INF] IPv4 flows
[INF] IPv6 flows
```

```
[INF] VLAN encapsulation
[INF] IPv4/6 fragmentation
[INF] MPLS encapsulation
[INF] L2TP encapsulation
[INF] PPP/HDLC encapsulation
[INF] GRE encapsulation
[INF] AYIYA tunnel
[INF] Teredo tunnel
[INF] CAPWAP/LWAPP tunnel
[INF] IPsec AH
[INF] IPsec ESP
[INF] SSDP/UPnP
[INF] SIP/RTP
```

Listing 1: *A sample Tranalyzer2 final report including encapsulation warning, Hash Autopilot engagement when hash table full*

T2 runs in IPv4 mode, but warns the user that there is IPv6 encapsulated. Note that the new Hash Autopilot warns you when the main hash map is full. T2 then removes the oldest flow and continues processing your pcap. To avoid that, run T2 again, but this time, use the `-f 5` option as indicated in the warning message:

```
[INF] Hash Autopilot: Fix: Invoke Tranalyzer with '-f 5'
   $ t2 -r ~/wurst/data/knoedel.pcap -w ~/results -f 5
```

or just let it run to the finish.

2.24 Monitoring Modes During Runtime

:If debugging is deactivated or the verbose level is zero (see Section 2.15), Tranalyzer2 prints no status information or end report. Since 0.8.8 the control of T2 can be achieved without the knowledge of the PID via `t2stat`:

```
t2stat -h
Usage:
    t2stat [OPTION...]

Optional arguments:
    INTERVAL     Send a signal to Tranalyzer every INTERVAL seconds
    -SIGNAME     Send SIGNAME signal instead of USR1
    -s           Run the command as root (with sudo)
    -p           Print Tranalyzer PID(s) and exit
    -l           List Tranalyzer PID(s), commands, running time and exit
    -i           Interactively cycle through all Tranalyzer processes

Help and documentation arguments:
    -h           Show help options and exit
```

Here are some examples

Command	Description
`t2stat`	T2 sends configured monitoring report to stdout
`t2stat 5`	T2 sends configured monitoring report to stdout every 5 seconds
`t2stat -USR2`	T2 toggles between on demand and continuous monitoring operation
`t2stat -SIGINT`	stop flow creation (like `^C` in the shell)
`t2stat -TERM`	terminate T2 immediately (Like two times `^C` in the shell)

The script `t2stat` has the same function as `kill -USR1 PID`. An example of a typical signal requested report (`MACHINE_REPORT=0`) is shown in Listing 2.

```
                                    @      @
                                     |    |
================================vVv==(a    a)==vVv================================
====================================\    /====================================
=====================================\  /=====================================
                                      oo
USR1 A type report: Tranalyzer 0.8.9 (Anteater), Tarantula. PID: 3434
PCAP time: 1291753338.831347 sec (Tue 07 Dec 2010 20:22:18 GMT)
PCAP duration: 113.384615 sec (1m 53s)
Time: 1613737828.648459 sec (Fri 19 Feb 2021 13:30:28 CET)
Elapsed time: 66.478134 sec (1m 6s)
Processing file: /home/wurst/knoedel.pcap
Total bytes to process: 42949673232 (42.95 G)
Percentage completed: 50.60%
Total bytes processed so far: 21730424832 (21.73 G)
Remaining time: 64.914335 sec (1m 4s)
ETF: 1613737893.562794 sec (Fri 19 Feb 2021 13:31:33 CET)
Number of processed packets: 27154806 (27.15 M)
Number of processed bytes: 21295948035 (21.30 G)
Number of raw bytes: 21303536835 (21.30 G)
Number of pad bytes: 12158 (12.16 K)
Number of IPv4 packets: 27051228 (27.05 M) [99.62%]
Number of IPv6 packets: 103450 (103.45 K) [0.38%]
Number of A packets: 15590982 (15.59 M) [57.42%]
Number of B packets: 11563824 (11.56 M) [42.58%]
Number of A bytes: 7188873480 (7.19 G) [33.76%]
Number of B bytes: 14107074555 (14.11 G) [66.24%]
Average A packet load: 461.09
Average B packet load: 1219.93 (1.22 K)
--------------------------------------------------------------------------------
tcpFlags: Number of TCP scans attempted, successful: 29501 (29.50 K), 51309 (51.31 K) [173.92%]
tcpFlags: Number of TCP SYN retries, seq retries: 32048 (32.05 K), 7843 (7.84 K)
icmpDecode: Aggregated icmpStat: 0x21
icmpDecode: Number of ICMP echo request packets: 8244 (8.24 K) [11.58%]
icmpDecode: Number of ICMP echo reply packets: 1948 (1.95 K) [2.74%]
dnsDecode: Number of DNS packets: 122201 (122.20 K) [0.45%]
dnsDecode: Number of DNS Q packets: 64102 (64.10 K) [52.46%]
dnsDecode: Number of DNS R packets: 58099 (58.10 K) [47.54%]
dnsDecode: Aggregated dnsStat: 0x0201
httpSniffer: Number of HTTP packets: 18450263 (18.45 M) [67.94%]
connStat: Number of unique source IPs: 136530 (136.53 K)
connStat: Number of unique destination IPs: 151775 (151.78 K)
connStat: Number of unique source/destination IPs connections: 1456 (1.46 K)
connStat: Max unique number of source IP / destination port connections: 2019 (2.02 K)
connStat: IP prtcon/sdcon, prtcon/scon: 1.386676, 0.014788
connStat: Source IP with max connections: x.x.x.x (CH): 1165 (1.17 K) connections
connStat: Destination IP with max connections: y.y.y.y (CH): 5028 (5.03 K) connections
--------------------------------------------------------------------------------
Headers count: min: 4, max: 13, average: 7.10
Max VLAN header count: 1
Max MPLS header count: 2
Number of LLC packets: 128 [0.00%]
Number of GRE packets: 128 [0.00%]
Number of Teredo packets: 103318 (103.32 K) [0.38%]
Number of AYIYA packets: 132 [0.00%]
Number of IGMP packets: 168 [0.00%]
Number of ICMP packets: 63517 (63.52 K) [0.23%]
```

```
Number of ICMPv6 packets: 7691 (7.69 K) [0.03%]
Number of TCP packets: 21820890 (21.82 M) [80.36%]
Number of TCP bytes: 18347271411 (18.35 G) [86.15%]
Number of UDP packets: 5137175 (5.14 M) [18.92%]
Number of UDP bytes: 2890294860 (2.89 G) [13.57%]
Number of IPv4 fragmented packets: 8295 (8.29 K) [0.03%]
Number of IPv6 fragmented packets: 3238 (3.24 K) [3.13%]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of processed   flows: 707800 (707.80 K)
Number of processed A flows: 593821 (593.82 K) [83.90%]
Number of processed B flows: 113979 (113.98 K) [16.10%]
Number of request     flows: 582056 (582.06 K) [82.23%]
Number of reply       flows: 125744 (125.74 K) [17.77%]
Total   A/B    flow asymmetry: 0.68
Total req/rply flow asymmetry: 0.64
Number of processed   packets/flows: 38.37
Number of processed A packets/flows: 26.26
Number of processed B packets/flows: 101.46
Number of processed total packets/s: 239492.87 (239.49 K)
Number of processed A+B   packets/s: 239492.87 (239.49 K)
Number of processed A     packets/s: 137505.27 (137.50 K)
Number of processed   B   packets/s: 101987.60 (101.99 K)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of average processed flows/s: 6242.47 (6.24 K)
Average full raw bandwidth: 1503099136 b/s (1.50 Gb/s)
Average snapped bandwidth : 1502563456 b/s (1.50 Gb/s)
Average full bandwidth : 1502764416 b/s (1.50 Gb/s)
Fill size of main hash map: 582178 [44.42%]
Max number of flows in memory: 582178 (582.18 K) [44.42%]
Memory usage: 5.61 GB [8.32%]
Aggregate flow status: 0x0c00b872d298fb04
[WRN] L3 SnapLength < Length in IP header
[WRN] Consecutive duplicate IP ID
[WRN] IPv4/6 payload length > framing length
[WRN] IPv4/6 fragmentation header packet missing
[INF] Ethernet flows
[INF] IPv4 flows
[INF] IPv6 flows
[INF] VLAN encapsulation
[INF] IPv4/6 fragmentation
[INF] MPLS encapsulation
[INF] L2TP encapsulation
[INF] PPP/HDLC encapsulation
[INF] GRE encapsulation
[INF] AYIYA tunnel
[INF] Teredo tunnel
[INF] CAPWAP/LWAPP tunnel
[INF] IPsec AH
[INF] IPsec ESP
[INF] SSDP/UPnP
[INF] SIP/RTP
================================================================================
```

Listing 2: *A sample Tranalyzer2 human readable report aggregate mode*

Note at the beginning `USR1 A type` denotes that there was a signal received and all reporting is aggregated from the beginning of T2 operation, almost like the end report. Note, that the reaction to signals depends to internal core configuration, discussed in a later chapter below. For the time being we stick with the default. If you require now a report every 30s then type `t2stat 30`. Then a signal is sent to T2 every 30s, hence a remote control. Nevertheless, he can do it

also by himself, more later. Note that the report does not only tell you all about packet statistics but also the

1. pcap duration
2. Elapsed time
3. Total bytes to process
4. Percentage completed
5. Total bytes processed so far
6. Remaining time
7. Estimated Time Finish (ETF)

Especially the ETF is very valuable for large or multiple large pcap operations, e.g., multiple 10TB files. So the Anteater tells you when to come back from your coffee break.

Listing 3 illustrates the output of the header line and subsequent data lines generated when `MACHINE_REPORT=1`.

```
%repTyp   time                sensorID  dur           memUsageKB  fillSzHashMap  numFlows  ...
USR1MR_A  1022171702.125000   666       0.308953000   31686       2152           2152      ...
USR1MR_A  1022171703.027000   666       1.308855000   33914       3919           3919      ...
USR1MR_A  1022171704.334000   666       2.309162000   34750       5031           5031      ...
USR1MR_A  1022171705.030000   666       3.308858000   35258       5847           5847      ...
```

Listing 3: *A sample Tranalyzer2 machine report aggregate mode*

2.24.1 Configuration for Monitoring Mode on interface real time mode

To enable monitoring mode, configure Tranalyzer as follows:

main.h		tranalyzer.h	
`#define MONINTTMPCP`	`0`	`#define DIFF_REPORT`	`1`
`#define MONINTTHRD`	`1`	`#define MACHINE_REPORT`	`1`

The following plugins contribute to the output:

- arpDecode
- basicStats
- cdpDecode
- connStat
- dnsDecode
- ftpDecode
- httpSniffer
- icmpDecode
- lldpDecode
- modbus
- mqttDecode
- ntpDecode
- radiusDecode
- sshDecode
- stpDecode
- t2PSkel
- tcpFlags
- vrrpDecode

The generated output is illustrated in Figure 3. The columns are as follows:

1. repType
2. sensorID
3. procID (`t2 -i` option and `DPDK_MP=1` only)
4. time
5. duration
6. pktsRec (`t2 -i` option only)
7. pktsDrp (`t2 -i` option only)
8. ifDrp (`t2 -i` option and `DPDK_MP=0` only)
9. pktsErr (`t2 -i` option and `DPDK_MP=1` only)
10. bytesRec (`t2 -i` option and `DPDK_MP=1` only)
11. memUsageKB
12. fillSzHashMap
13. numFlows
14. numAFlows
15. numBFlows
16. numPkts
17. numAPkts
18. numBPkts
19. numL2Pkts
20. numV4Pkts
21. numV6Pkts
22. numVxPkts
23. numBytes
24. numABytes
25. numBBytes
26. numFrgV4Pkts
27. numFrgV6Pkts
28. numAlarms
29. rawBandwidth
30. globalWarn
31. Layer 2 protocols stats (see `MONPROTL2` in *main.h*):
    - 0x0806Pkts (ARP)
    - 0x0806Bytes (ARP)
    - 0x8035Pkts (RARP)
    - 0x8035Bytes (RARP)
32. Layer 3 protocols stats (see `MONPROTL3` in *main.h*):
    - TCPPkts
    - TCPBytes
    - UDPPkts
    - UDPBytes
    - ICMPPkts
    - ICMPBytes
    - ICMPv6Pkts
    - ICMPv6Bytes
    - SCTPPkts
    - SCTPBytes

2.24.2 Monitoring Mode to Syslog

In order to send monitoring info to a syslog server T2 must be configured in machine mode as indicated above. Then the output has to be piped into the following script:

```
t2 -D ... -w ... | gawk -F"\t" `{ print "<25> ", strftime("%b %d %T"), "Monitoring: " $0 }' | \
        nc -u w.x.y.z 514
```

Netcat will send it to the syslog server at address `w.x.y.z`. Specific columns from the monitoring output can be selected in the awk script.

2.24.3 RRD Graphing of Monitoring Output

The monitoring output can be stored in a RRD database using the `t2rrd` script. To start creating a RRD database, launch Tranalyzer2 (in monitoring mode) as follows:

```
t2 -r file.pcap -l | t2rrd -m
```

Or for monitoring from a live interface:

```
st2 -i eth0 -l | t2rrd -m
```

Plots for the various fields can then be generated using the same `t2rrd` script:

```
t2rrd -i 30s numAFlows numBFlows
```

To specify intervals, use `s` (seconds), `m` (minutes), `h` (hour), `d` (day), `w` (week), `mo` (month), `y` (year). For example, to plot the data from the last two weeks, use `-i 2w` or `-s -2w`.

An example graph is depicted in Figure 4.

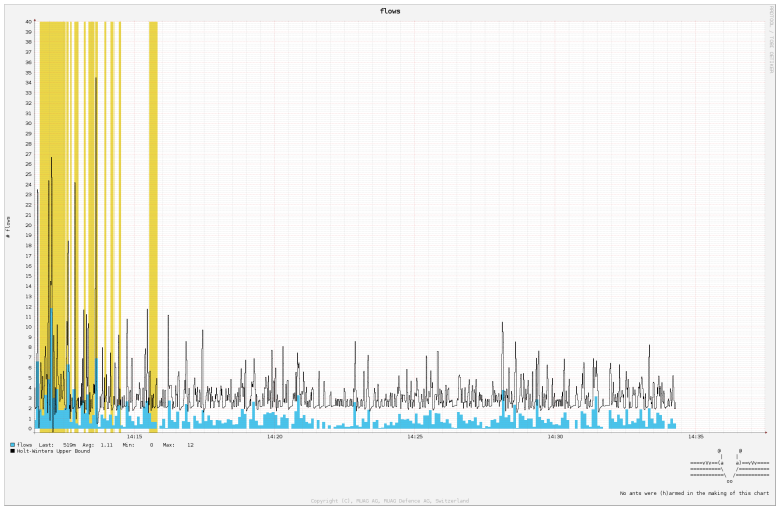


Figure 4: *T2 monitoring using RRD*

2.25 Cancellation of the Sniffing Process

Processing of a pcap file stops upon end of file. In case of live capture from an interface Tranalyzer2 stops upon `CTRL+C` interrupt or a `kill -9 PID` signal. The disconnection of the interface cable will stop Tranalyzer2 also after a timeout of 182 seconds. The console based `CTRL+C` interrupt does not immediately terminate the program to avoid corrupted entries in the output files. It stops creating additional flows and finishes only currently active flows. Note that waiting the termination of active flow depends on the activity or the lifetime of a connection and can take a very long time. In order to mitigate that problem the user can issue the `CTRL+C` for `GI_TERM_THRESHOLD` times to immediately terminate the program.