April 15, 2015

NFSTRACE USER AND DEVELOPER MANUAL
Version 0.4.0

NFSTRACE User and developer manual

EPAM Systems

This manual provides basic instructions on how to use nfstrace to monitor NFS and CIFS activity and how to develop pluggable analysis modules.

Contents

1. INTRODUCTION

`nfstrace` performs live Ethernet 1 Gbps – 10 Gbps packets capturing and helps to determine NFS and CIFS procedures in raw network traffic. Furthermore, it performs filtration, dumping, compression, statistical analysis, visualization and provides the API for custom pluggable analysis modules.

`nfstrace` captures raw packets from an Ethernet interface using libpcap interface to Linux (LSF) or FreeBSD (BPF) implementations. At the moment it is assumed that libpcap delivers correct TCP and UDP packets. Assembling of IP packets from ethernet frames and IP packets defragmentation are performed in the operating system's kernel.

The application has been tested on the workstations with integrated 1 Gbps NICs (Ethernet 1000baseT/Full).

Currently `nfstrace` supports the following protocols:

Ethernet → IPv4 | IPv6 → UDP | TCP → NFSv3 | NFSv4 | NFSv4.1 | CIFSv1 | CIFSv2

1.1 PORTABILITY

The application has been developed and tested on GNU/Linux (Fedora 20, OpenSUSE 13.2, Ubuntu 14.04/14.10, CentOS 7, Arch Linux, Alt Linux 7.0.5) and FreeBSD (FreeBSD 10.1). It is written in C++11 programming language and uses standard POSIX interfaces and the following libraries: libpthread, libpcap, libstdc++.

2 USAGE

2.1 OPTIONS

`-m, --mode=live|dump|drain|stat`

    Set the running mode (see the description below) (default: `live`).

`-i, --interface=INTERFACE`

    Listen interface, it is required for live and dump modes (default: searches for the lowest numbered, configured up interface (except loopback)).

`-f, --filtration="filter"`

    Specify the packet filter in BPF syntax; for the expression syntax, see `pcapfilter(7)` (default: `"port 2049 or port 445"`).

`-s, --snaplen=1..65535`

    Set the max length of captured raw packet (bigger packets will be truncated). Can be used **only for UDP** (default: `65535`).

`-t, --timeout=milliseconds`

    Set the read timeout that will be used while capturing (default: `100`).

`-b, --bsize=MBytes`

    Set the size of the operating system capture buffer in MBytes; note that this option is crucial for capturing performance (default: `20`).

`-p, --promisc`

    Put the capturing interface into promiscuous mode (default: `true`).

`-d, --direction=in|out|inout`

    Set the direction for which packets will be captured (default: `inout`).

`-a, --analysis=PATH#opt1,opt2=val,...`

    Specify the path to an analysis module and set its options (if any).

`-I, --ifile=PATH`

    Specify the input file for stat mode, '–' means `stdin` (default: `nfstrace{filter}.pcap`).

`-O, --ofile=PATH`

    Specify the output file for dump mode, '-' means stdout (default: `nfstrace-{filter}.pcap`).

```
--log=PATH
```
Specify the log file (default: `nfstrace.log`).

```
-C, --command="shell command"
```
Execute command for each dumped file.

```
-D, --dump-size=MBytes
```
Set the size of dumping file portion, `0` means no limit (default: `0`).

```
-E, --enum=interfaces|plugins|-
```
Enumerate all available network interfaces and and/or all available plugins, then exit; please note that interfaces can't be listed unless nfstrace was built against the recent version of libpcap that supports the `pcap_findalldevs()` function (default: `none`).

```
-M, --msg-header=1..4000
```
Truncate RPC messages to this limit (specified in bytes) before passing to a pluggable analysis module (default: `512`).

```
-Q, --qcapacity=1..65535
```
Set the initial capacity of the queue with RPC messages (default: `4096`).

```
-T, --trace
```
Print collected NFSv3/NFSv4/NFSv4.1/CIFSv2 procedures, `true` if no modules were passed with `-a` option.

```
-Z, --droproot=username
```
Drop root privileges after opening the capture device.

```
-v, --verbose=0|1|2
```
Specify verbosity level (default: `1`).

```
-h, --help
```
Print help message and usage for modules passed with `-a` option, then exit.

2.2 RUNNING MODES

`nfstrace` can operate in four different modes:

- online analysis (`--mode=live`): performs online capturing, filtration and live analysis of detected NFS/CIFS procedures using a pluggable analysis module or prints out them to `stdout` (`-T` or `--trace` options);

- online dumping (`--mode=dump`): performs online traffic capturing, filtration and dumping to the output file (specified with `-O` or `--ofile` options);

- offline analysis (`--mode=stat`): performs offline filtration of the `.pcap` that contains previously captured traces and performs analysis using a pluggable analysis module or prints found NFS/CIFS procedures to `stdout` (`-T` or `-trace` options);

- offline dumping (`--mode=drain`): performs a reading of traffic from the `.pcap` file (specified with `-I` or `--ifile` options), filtration, dumping to the output `.pcap` file (specified with `-O` or `--ofile` options) and removing all the packets that are not related to NFS/CIFS procedures.

2.3 PACKETS FILTRATION

Internally `nfstrace` uses libpcap that provides a portable interface to the native system API for capturing network traffic. By so doing, filtration is performed in the operating system's kernel. `nfstrace` provides a special option (`-f` or `--filtration`) for specifying custom filters in BPF syntax.

The default BPF filter in `nfstrace` is `port 2049 or port 445`, which means that each packet that is delivered to user-space from the kernel satisfies the following conditions: it has IPv4 or IPv6 header and it has TCP and UDP header with source or destination port number equals to 2049 (default NFS port) or 445 (default CIFS port).

The reality is that this filter is very heavy and support of IPv6 is experimental, so if you want to reach faster filtration of IPv4-only traffic we suggest to use the following BPF filter: `ip and port 2049 or port 445`.

2.4 DUMP FILE FORMAT

`nfstrace` uses libpcap file format for input and output files so any external tool (e.g. Wireshark) can be used in order to inspect filtered traces.

2.5 USAGE EXAMPLES

In this sections some use cases will be explained. Every next example inherit something from the previous ones, so we suggest to read all of them from the beginning.

2.5.1 AVAILABLE OPTIONS

The following command demonstrates available options of the application and plugged analysis modules (attached with `--analysis` or `-a` options). Note that you can pass more than one module here.

```
nfstrace --help --analysis=libjson.so
```

2.5.2 ONLINE TRACING

The following command will run `nfstrace` in online analysis mode (specified with `--mode` or `-m` options) without a pluggable analysis module. It will capture NFS traffic transferred over TCP or UDP with source or destination port number equals to 2049 and will simply print them out to `stdout` (`-T` or `--trace` options). Capturing is over when `nfstrace` receives `SIGINT` (Control-C).

Note that capturing from network interface requires superuser privileges.

```
nfstrace --mode=live                  \
         --filtration="ip and port 2049" \
         -T
```

2.5.3 ONLINE ANALYSIS

The following command demonstrates running `nfstrace` in online analysis mode. Just like in the previous example it will capture NFS traffic transferred over TCP or UDP with source or destination port number equals to 2049 and then it will perform Operation Breakdown analysis using pluggable analysis module `libbreakdown.so`.

```
nfstrace --mode=live                  \
         --filtration="ip and port 2049" \
         --analysis=libbreakdown.so
```

2.5.4 ONLINE DUMPING AND OFFLINE ANALYSIS

The following example demonstrates running `nfstrace` in online dumping and offline analysis modes. At first `nfstrace` will capture NFS traffic transferred over TCP or UDP with source or destination port number equals to 2049 and will dump captured packets to `dump.pcap` file (specified with `--ofile` or `-O` options).

At the second run `nfstrace` will perform offline Operation Breakdown analysis using pluggable analysis module `libbreakdown.so`.

```
# Dump captured packets to dump.pcap
nfstrace --mode=dump                  \
         --filtration="ip and port 2049" \
         -O dump.pcap

# Analyse dump.pcap using libbreakdown.so
nfstrace --mode=stat                  \
         -I dump.pcap                 \
         --analysis=libbreakdown.so
```

2.5.5 ONLINE DUMPING, COMPRESSION AND OFFLINE ANALYSIS

The following example demonstrates running `nfstrace` in online dumping and offline analysis modes. Since dump file can easily exhaust disk space, compression makes sense.

At first `nfstrace` will capture NFS traffic transferred over TCP or UDP with source or destination port number equals to 2049 and will dump captured packets to `dump.pcap` file.

Note that compression is done by the external tool (executed in script passed with `--command` or `-C` options) and it will be executed when capturing is done. The output file can be inspected using some external tool as described in [2.4].

At the second run `nfstrace` will perform offline analysis. Again, the external tool (`bzcat` in this example) is used in order to decompress previously saved dump. `nfstrace` will read `stdin` (note the `-I -` option) and perform offline analysis using Operation Breakdown analyzer.

```
# Dump captured procedures to dump.pcap file.
# Compress output using bzip2 when capturing is over.
nfstrace --mode=dump                 \
        --filtration="ip and port 2049" \
        -O dump.pcap                 \
        -C "bzip2 -f -9"

# Extract dump.pcap from dump.pcap.bz2 to stdin.
# Read stdin and analyze data with libbreakdown.so module.
bzcat dump.pcap.bz2 | nfstrace --mode=stat         \
                               -I -                \
                               --analysis=libbreakdown.so
```

2.5.6 ONLINE DUMPING WITH FILE LIMIT, COMPRESSION AND OFFLINE ANALYSIS

This example is similar to the previous one except one thing: output dump file can be very huge and cause problems in some situations, so `nfstrace` provides the ability to split it into parts.

At first `nfstrace` will be invoked in online dumping mode. Everything is similar to the previous example except `-D` (`--dump-size`) option: it specifies the size limit in MBytes, so dump file will be split according to this value.

At the second run `nfstrace` will perform offline analysis of captured packets using Operation Breakdown analyzer.

Please note that only the first dump file has the pcap header.

```
# Dump captured procedures to the multiple files and compress them.
nfstrace --mode=dump                 \
         --filtration="ip and port 2049" \
        -O dump.pcap                 \
        -D 1                         \
        -C "bzip2 -f -9"

# get list of parts in the right order:
#    dump.pcap.bz2
#    dump.pcap-1.bz2
parts=$(ls dump.pcap*.bz2 | sort -n -t - -k 2)
```

```
# Extract main dump.pcap and parts from dump.pcap.bz2 to stdin.
# Read stdin and analyze data with libbreakdown.so module.
bzcat "$parts" | nfstrace --mode=stat                 \
                          -I -                         \
                          --analysis=libbreakdown.so
```

2.5.7 VISUALIZATION

This example demonstrates the ability to plot graphical representation of data collected by Operation Breakdown analyzer.

`nst.sh` is a shell script that collects data generated by analyzers and passes it to gnuplot script specified with `-a` option.

`breakdown_nfsv3.plt and breakdown_nfsv4.plt` are a gnuplot scripts that understand output data format of Operation Breakdown analyzer and generate `.png` files with plots.

Note that `gnuplot` must be installed.

```
# Extract dump.pcap from dump.pcap.bz2 to stdin.
# Read stdin and analyze data with libbreakdown.so module.
bzcat trace.pcap.bz2 | nfstrace -m stat -I - -a libbreakdown.so

# Generate plot according to *.dat files generated by
# libbreakdown.so analyzer.
nst.sh -a breakdown.plt -d . -p 'breakdown*.dat' -v
```

3 ANALYZERS

All pluggable modules are implemented as external shared libraries.

3.1 OPERATION BREAKDOWN ANALYZER (LIBBREAKDOWN.SO)

Operation Breakdown (OB) analyzer calculates average frequency of NFS/CIFS procedures and computes standard deviation of latency.

```
$ nfstrace -a libbreakdown.so -h
nfstrace 0.4.0 (Release) built on Linux-3.16.1-1-generic by C++ compiler GNU 4.9.1
Usage: ./nfstrace [OPTIONS]...
```

And the result of execution will look something like this:

```
LLog file: nfstrace.log
Loading module: 'libbreakdown.so' with args: []
Read packets from:
datalink: EN10MB (Ethernet)
version: 2.4
Note: It's potentially unsafe to run this program as root without dropping root
privileges.
Note: Use -Z username option for dropping root privileges when you run this program as
user with root privileges.
Processing packets. Press CTRL-C to quit and view results.
Detect session 127.0.0.1:34744 --> 127.0.1.1:2049 [TCP]
Detect session 127.0.0.1:929 --> 127.0.1.1:2049 [TCP]
Detect session 127.0.0.1:774 --> 127.0.1.1:2049 [TCP]
Detect session 127.0.0.1:854 --> 127.0.1.1:2049 [TCP]
Detect session 10.0.2.15:55529 --> 10.6.208.121:445 [TCP]
Detect session 10.0.2.15:55530 --> 10.6.208.121:445 [TCP]
###  Breakdown analyzer  ###
CIFS v1 protocol Total operations:
101. Per operation:
CREATE_DIRECTORY         0    0.00%
DELETE_DIRECTORY         0    0.00%
OPEN                     0    0.00%
CREATE                   0    0.00%
...complete output has been omitted...
Per connection info:
Session: 10.0.2.15:55529 --> 10.6.208.121:445 [TCP]
Total operations: 101. Per operation:
CREATE_DIRECTORY     Count:    0 (  0.00%) Min: 0.000 Max: 0.000 Avg: 0.000 StDev:
0.00000000
DELETE_DIRECTORY     Count:    0 (  0.00%) Min: 0.000 Max: 0.000 Avg: 0.000 StDev:
0.00000000
OPEN                 Count:    0 (  0.00%) Min: 0.000 Max: 0.000 Avg: 0.000 StDev:
0.00000000
CREATE               Count:    0 (  0.00%) Min: 0.000 Max: 0.000 Avg: 0.000 StDev:
0.00000000
CLOSE                Count:    2 (  1.98%) Min: 0.001 Max: 0.002 Avg: 0.001 StDev:
0.00057205
...complete output has been omitted...
###  Breakdown analyzer  ###
CIFS v2 protocol Total operations:
77. Per operation:
NEGOTIATE                1    1.30%
SESSION SETUP            2    2.60%
```

```
LOGOFF                    0   0.00%
TREE CONNECT              2   2.60%
...complete output has been omitted...
###  Breakdown analyzer  ###
NFS v3 protocol Total operations: 7123. Per operation:
NULL          2   0.03%
GETATTR      47   0.66%
SETATTR       5   0.07%
LOOKUP        4   0.06%
...complete output has been omitted...
###  Breakdown analyzer  ###
NFS v4.0 protocol
Total procedures: 3264. Per procedure:
NULL                2   0.06%
COMPOUND         3262  99.94%
Total operations: 9701. Per operation:
ILLEGAL             0   0.00%
ACCESS             16   0.16%
CLOSE               5   0.05%
...complete output has been omitted...
###  Breakdown analyzer  ###
NFS v4.1 protocol
Total procedures: 8127. Per procedure:
NULL                0   0.00%
COMPOUND         8127 100.00%
Total operations: 32359. Per operation:
ILLEGAL             0   0.00%
ACCESS             15   0.05%
CLOSE               5   0.02%
COMMIT             81   0.25%
...complete output has been omitted...
```

OB analyzer produces .dat file in the current directory for each detected NFS/CIFS session:

```
$ ls -a *.dat
breakdown_10.6.137.79:949 --> 10.6.7.38:2049 [TCP].dat
```

As described in [2.5.7], produced .dat files can be visualized using nst.sh and breakdown_nfsv3.plt or breakdown_nfsv4.plt (according to NFS version).

```
nst.sh -a breakdown_nfsv3.plt -d . -f 'breakdown_10.6.137.79:949 --> 10.6.7.38:2049
[TCP].dat'
```
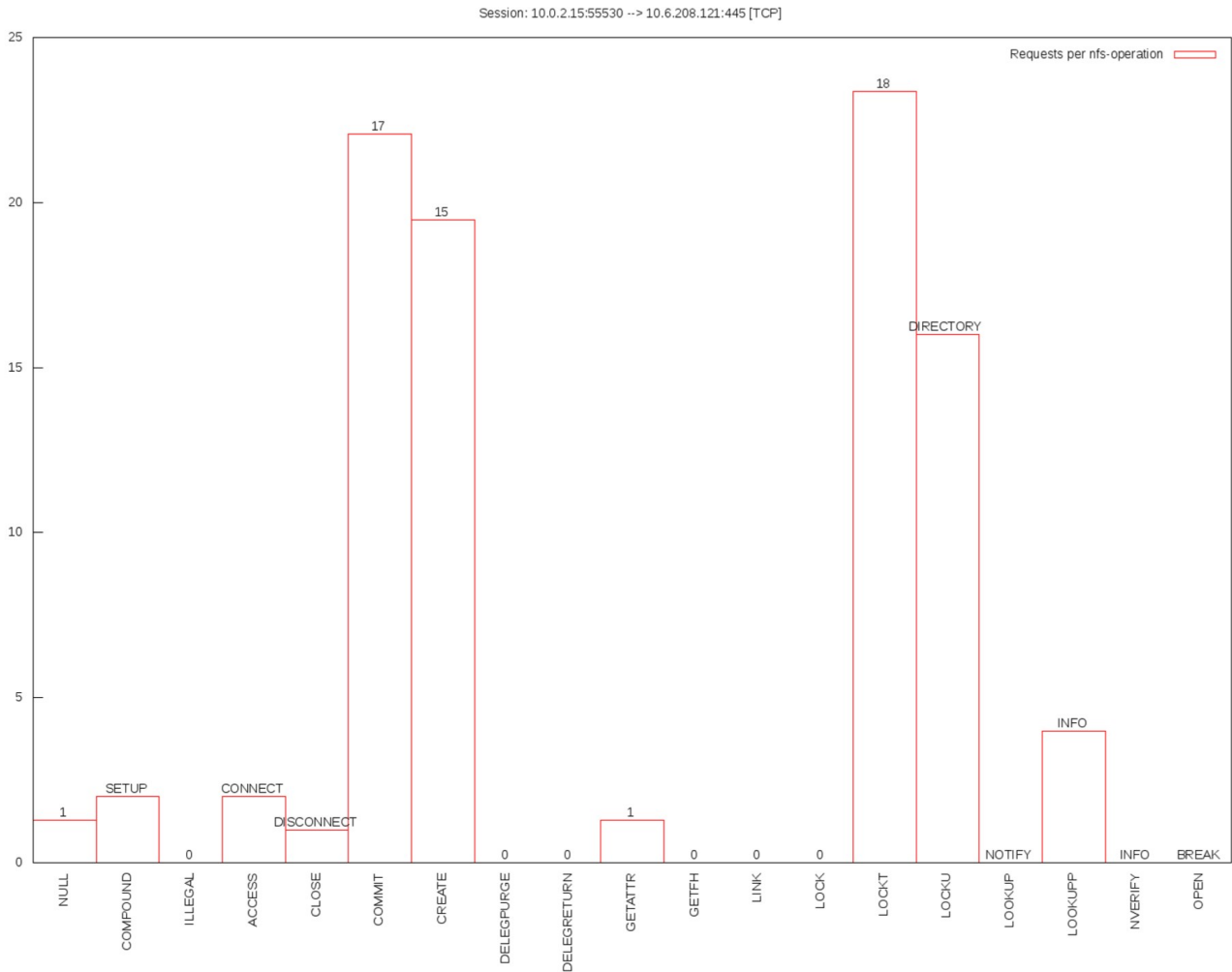
Session: 10.0.2.15:55530 --> 10.6.208.121:445 [TCP]

Figure 3.1 – Session Visualization

3.2 WATCH (LIBWATCH.SO)

Watch plugin mimics old `nfswatch` utility: it monitors NFS and CIFS traffic and displays it in terminal using `ncurses`. It supports NFSv3, NFSv4, NFSv4.1, CIFSv1 and CIFSv2.

```
--------------------------------------------------------------------------------
¦Nfstrace watch plugin. To scroll press up or down keys. Ctrl + c to exit.     ¦
¦Host name:      epbyminw0059.minsk.epam.com                                   ¦
¦Date:   3.4.2015        Time: 16:57:58                                        ¦
¦Elapsed time:           0 days; 0:1:50 times                                 ¦
--------------------------------------------------------------------------------
¦      NFS v3      NFS v4      NFS v41      CIFS v1    < CIFS v2 >             ¦
¦                                                                              ¦
¦Total:                182                                                     ¦
¦NEGOTIATE          3    1.65%                                                 ¦
¦SESSION SETUP      8    4.40%                                                 ¦
¦LOGOFF             1    0.55%                                                 ¦
¦TREE CONNECT       4    2.20%                                                 ¦
¦TREE DISCONNECT    2    1.10%                                                 ¦
¦CREATE            33   18.13%                                                 ¦
¦CLOSE             32   17.58%                                                 ¦
¦FLUSH              0    0.00%                                                 ¦
¦READ               2    1.10%                                                 ¦
¦WRITE              1    0.55%                                                 ¦
¦LOCK               0    0.00%                                                 ¦
¦IOCTL              0    0.00%                                                 ¦
¦CANCEL             0    0.00%                                                 ¦
¦ECHO              50   27.47%                                                 ¦
¦QUERY DIRECTORY   34   18.68%                                                 ¦
¦CHANGE NOTIFY      0    0.00%                                                 ¦
¦QUERY INFO         9    4.95%                                                 ¦
¦SET INFO           3    1.65%                                                 ¦
¦OPLOCK BREAK       0    0.00%                                                 ¦
¦                                                                              ¦
L--------------------------------------------------------------------------------
```

By default watch plugin will update its screen every second, you can specify another timeout in milliseconds:

```
$ nfstrace -a libwatch.so#2000
```

3.3 JSON ANALYZER (LIBJSON.SO)

JSON analyzer calculates a total amount of each supported application protocol operation. It accepts TCP-connections on particular TCP-endpoint (host:port), sends a respective JSON to the TCP-client and closes connection. Suggested to be used in **live** mode.

Available options:

> `host=HOSTNAME`
>> Network interface to listen (default: listen all interfaces)

> `port=PORT`
>> IP-port to bind to (default: `8888`)

> `workers=WORKERS`
>> Amount of worker threads (default: `10`)

> `duration=DURATION`
>> Max serving duration in milliseconds (default: `500`)

> `backlog=BACKLOG`
>> Listen backlog (default: `15`)

In order to try this analyzer out you can start `nfstrace` in on terminal:

```
$ nfstrace -i eth0 -a libjson.so#host=localhost
```

And then you can make a TCP-request to `nfstrace` in another terminal to fetch current statistics:

```
$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
{
"nfs_v3":{
"null":32,
"getattr":4582,
...
},
"nfs_v4":{
... }, ...
}Connection closed by foreign host.
```

4 IMPLEMENTATION DETAILS

This section may be interested for the developers who want to contribute or implement new pluggable analysis module.

4.1 PAYLOAD FILTRATION

Each NFSv3 procedure consists of two RPC messages:

- call – request from client to server;

- reply – reply from server with result of requested procedure.

Both RPC messages may contain data useful for analysis. Both RPC messages may contain thousands of payload bytes useless for analysis. `nfstrace` captures headers of calls and replies and then matches pairs of them to complete NFS procedures.

The `--snaplen` option sets up the amount of bytes of incoming packet for uprising from the kernel to user-space. In case of TCP transport layer this option is useless because TCP connection is a bidirectional stream of data (instead of UDP that is form of interchange up to 64k datagrams). In case of NFS over TCP `nfstrace` captures whole packets and copies them to user-space from the kernel for DPI and performing NFS statistical analysis.

Finally, `nfstrace` filtrates whole NFS traffic passed from the kernel to user-space and detects RPC/NFS message headers (up to 4 Kbytes) within gigabytes of network traffic.

Detected headers are copied into internal buffers (or dumped into a `.pcap` file) for statistical analysis.

The key principle of the filtration here is **discard payload ASAP**.

Filtration module works in a separate thread and captures packets from network interface using libpcap. It matches packets to a related session (TCP or UDP) and performs reassembling of TCP flow from a TCP segment of a packet. After that the part of a packet will be passed to `RPCFiltrator`. In case of NFSv4 the whole packet will be passed to `RPCFiltrator` because it consists of several NFSv4 operations.

There are two `RPCFiltrator` in one TCP session. Both of them know the state of the current RPC message in related TCP flow. They can detect RPC messages and perform actions on a packet: discard it or collect for analysis.

The size of the kernel capture buffer can be set with `-b` option (in MBytes). Note that this option is very crucial for capturing performance.

`wsize` and `rsize` of an NFS connection are important for filtration and performance analysis too.

4.2 PLUGGABLE ANALYSIS MODULES

`nfstrace` provides C++ api for implementing pluggable analysis modules. Header files provide definitions of `IAnalyzer` interface, NFS/CIFS data structures and functions. The `IAnalyzer` interface is a set of NFS/CIFS handlers that will be called by `Analysis` module for each NFS/CIFS procedure. All constants and definitions of types will be included with `<nfstrace/api/plugin_api.h>` header.

A pluggable analysis module must be a dynamically linked shared object and must export the following C functions:

```
const char* usage (); // return description of expected opts for create(opts)
IAnalyzer* create (const char* opts); // create and return an instance of an Analyzer
void destroy (IAnalyzer* instance); // destroy created instance of an Analyzer const
AnalyzerRequirements* requirements(); // return Analyzer's requirements
```

After the declaration of all these function there must be the following macro:

```
NST_PLUGIN_ENTRY_POINTS (&usage, &create, &destroy, &requirements)
```

`usage()` function must return a C-string with module description and required parameters for creation of an instance of analyzer, this string will be shown in the output of `--help` option.

`IAnalyzer* create(const char* opts)` must create and return an instance of the analyzer according to passed options.

`void destroy(IAnalyzer* instance)` must destroy previously created analyzer instance and perform required cleanups (e.g. close connection to a database etc.).

`const AnalyzerRequirements* requirements()` must create and return an instance of analyzer requirements. Its `silence` property is used if exclusive control over standard output is required.

All existing analyzers are implemented as pluggable analysis modules and can be attached to `nfstrace` with `-a` option.

4.3 GENERAL SCHEMA

The general schema of `nfstrace` is presented in the figure 4.1.

In this schema you can see how data flows in different modes:

- on-line analysis – green line ;

- on-line dumping – yellow line ;

- off-line dumping – blue line ;
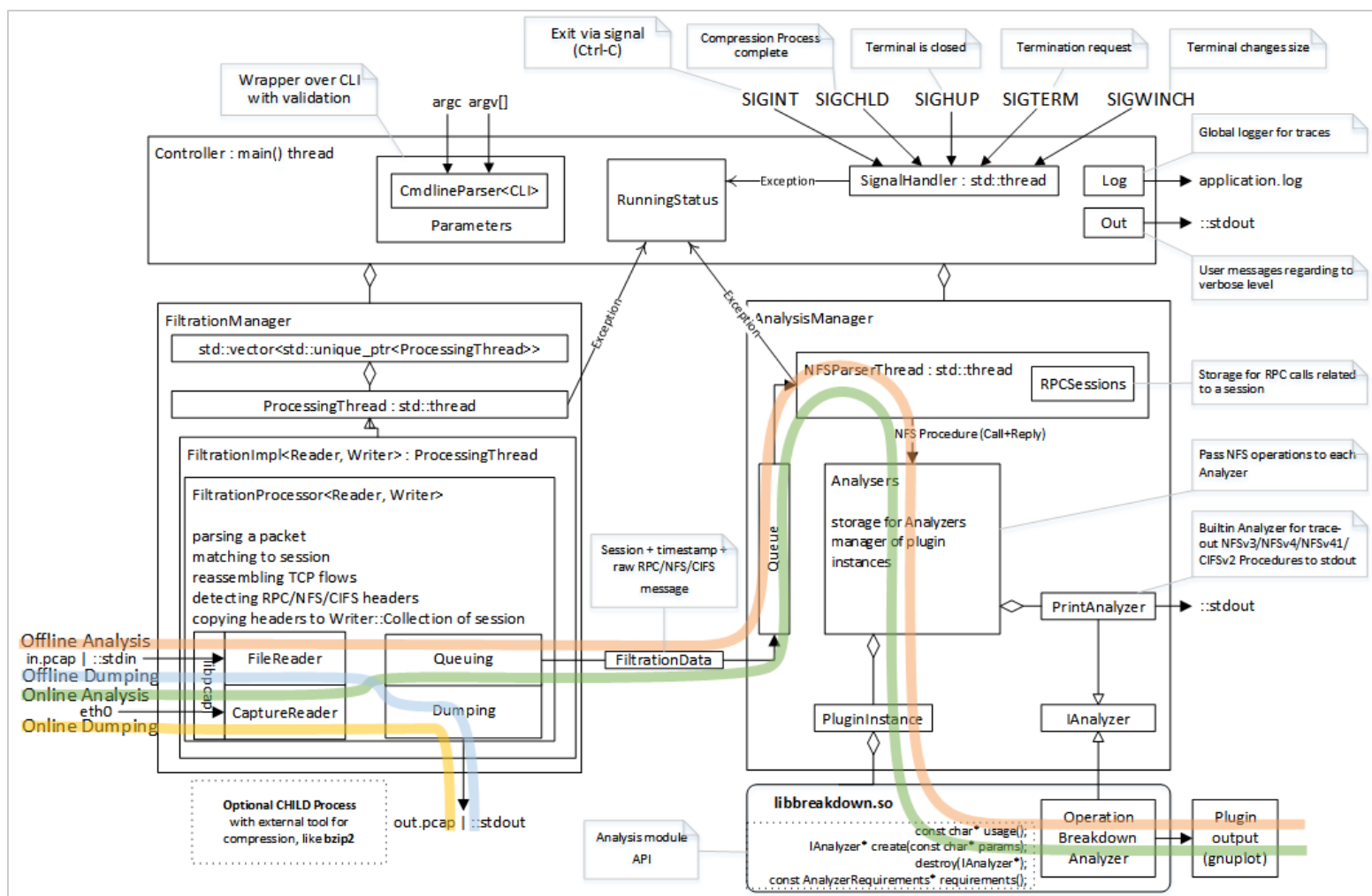
- off-line analysis – orange line .

Figure 4.1 - General schema

5 GLOSSARY

BPF

CIFS

DPI

Gnuplot

LSF

NFS

NIC

Payload

POSIX

rsize

Wireshark

Wsize