Appendix A

User manual

In this appendix, we will detail how to compile and launch the collaborative IDS from the terminal since it does not have at the moment a GUI. We will also list the most important and modified files, what they contain and the changes that can be done in order to tweak the parameters to be more suitable to the user.

A.1 Requirements and compilation

The implementation of the IDS can be found in the following Bitbucket repository `https://bitbucket.org/Mickael-Hamaide/sinf2990-distributed-ids` in the *ids* directory. The program requires the **glib** library in order to have an implementation of an hash table, the **libpcap** library to sniff packets on a designated interface.

In the *ids* folder, a script *compile.sh* is available to compile the Chord library, the sniffer, the server or all. The usage is as follow:

```
./compile.sh [chord, sniffer, server, all]
```

with the arguments in [...] being exclusive.

A.2 Terminal command and logs

The command to start the program is described as :

```
./sniffer [-cs] [-i interface] [-h host] [-p port]
```

- -c : Chord correlation unit is used
- -s : central server is used [default]
- -i : analyze the traffic on interface [default=eth0]
- -h : host is contacted to join the network [optional for chord and mandatory for central server]
- -p : port on which host is contacted [mandatory for central server, not used for Chord network as hosts always run on port 8000]

The user is alerted by the program through the log files. On each of the devices, the program is logging to the file "log-$I" where $I is the lowest positive integer found where the file does not exist. In these logs, there are different possible tags which are defined in the *ids/cChord/log.h*. All globally suspicious addresses detected are under the tag *ALERT* with the sentence: "divided thresh of **x** by 2" where **x** is the suspicious address. For now, the user is also alerted directly on the standard output of the program.

A.3 File structure

Hereby is the directory tree containing the most useful files :

```
ids
├── csniffer
│   ├── sniffer.c
│   ├── sniffer.h
│   ├── recordbuilder.c
│   ├── recordbuilder.h
│   ├── producer.c
│   ├── chord.cpp
│   ├── locks.c
│   ├── struct.h
│   └── BST.c
├── centralServer
│   ├── clientsrc
│   │   └── client.c
│   └── serversrc
│       └── server.c
└── cChord
    ├── engine
    │   ├── p2p
    │   │   └── protocol
    │   │       └── chord
    │   │           ├── ChordNode.cpp
    │   │           ├── ChordNode.h
    │   │           └── AbstractNode.cpp
    │   └── helper
    │       ├── thread
    │       │   └── Stabilization.h
    │       └── mongoose_http_server
    │           └── chord_callbacks.cpp
    └── htablecache.c
```

A.3.1 Sniffer

sniffer.c contains all the capture options.

sniffer.h BUFSIZE is the snaplen used for the capture of each packet.

recordbuilder.c contains the implementation of the detection unit.

recordbuilder.h contains the thresholds for syn floods and distributed syn floods. Moreover, it contains PERIOD which is the sleeping time of the analyzing thread and TIMEOUT which is the maximum inactive time of a flow to be in the structure.

producer.c is responsible of the buffer implementation.

chord.cpp is used to communicate and provide an easier API to send to the chord network.

locks.c contains the different mechanisms to synchronize the detection and correlation unit. This file should be modified in order to add another correlation unit.

struct.h is used to store all kind of structures of the IDS.

BST.c is the implementation of an unbalanced binary search tree. This tree is the one used to store locally detected attackers.

A.3.2 Central server

client.c obviously contains all the code of the client.

server.c in the same way, it is the implementation of the server. It contains the backlogsize of the listening socket, the size of the buffer in each thread containing the pending messages received by a client, and also the threshold concerning the number of client necessary to alert all hosts of a suspicious address.

A.3.3 Chord

ChordNode.cpp contains all methods modified or added to the library related to a specific node, meaning the replication, the successor list and the hash table storing addresses.

ChordNode.h contains the global threshold and different time related variables.

Stabilization.h is the stabilization thread.

htablecache.c contains the API to store the suspicious-notifier pair in the hash table.

chord_callbacks.cpp contains all the callbacks to answer to clients queries.

A.4 Router configuration

As stated multiple times in this thesis, there is a need for duplication of the packets at the ingress router. This is possible by modifying the rules of Netfilter [35] used for packet filtering in the linux kernel, and particularly in OpenWRT. These rules can be modified by the command line tool *iptables*:

```
iptables -t mangle -A PREROUTING ! -s <address> ! -d <address> \
         -j TEE --gateway <address>
```

This command ensures a duplication of all packets to *address* except the ones from or already destined to *address*.

Appendix B

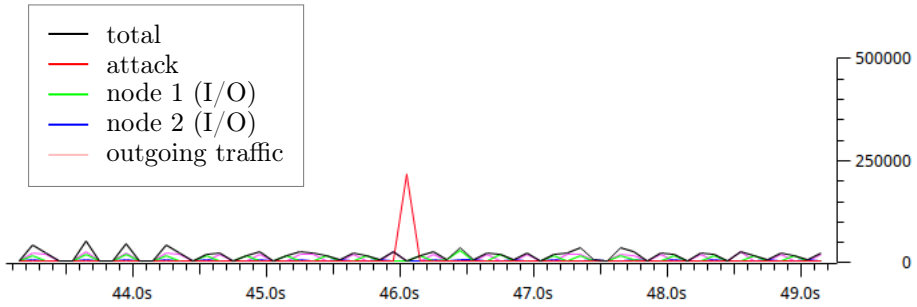
Additional test results

B.1 Chord overhead

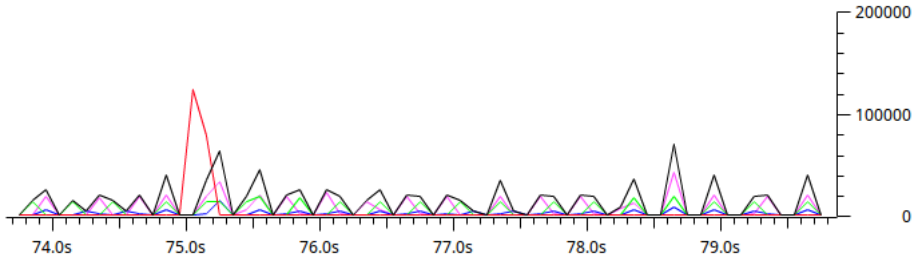
Fig. B.0 depicts all the graphics of the tests of the overhead with a Chord correlation unit with different number of attackers. The overhead is depicted in bytes/tick with a 0.1s tick and using the colors as explained in section 7.5.1.

B.2 Central server overhead

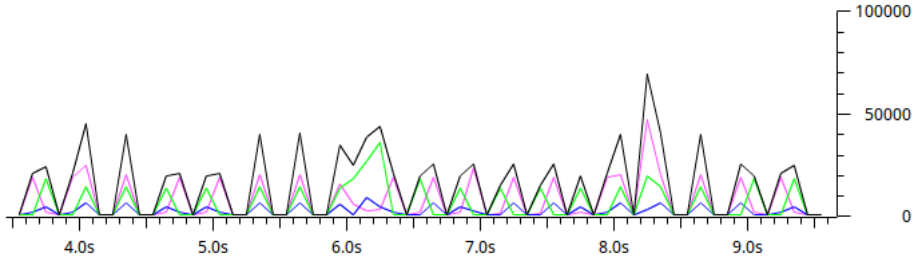
The results of the tests concerning the overhead due to the centralized correlation unit are displayed in this fig. B.0. The overhead is depicted in bytes/tick with a 0.01s tick and using the colors as explained in section 7.6.1.



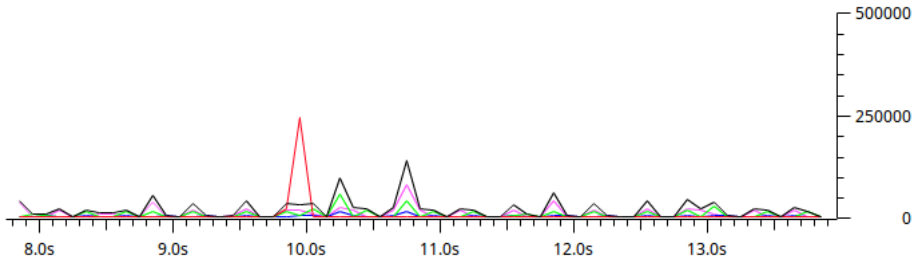
(a) 1 attacker



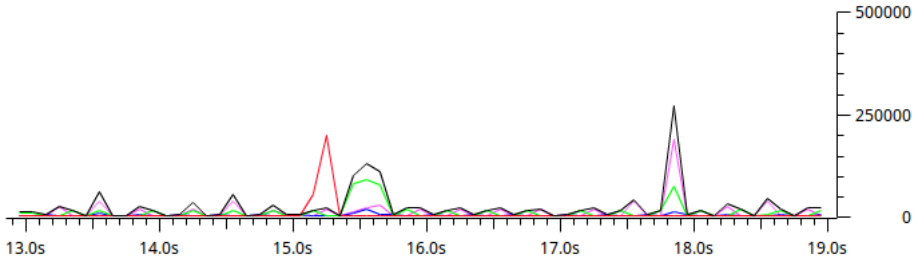
(b) 2 attackers



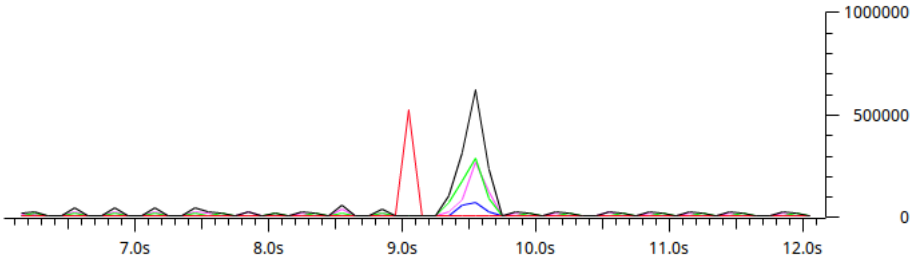
(c) 5 attackers without the attack



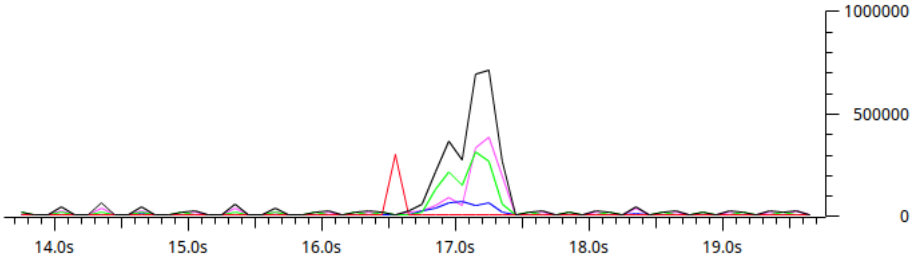
(d) 10 attackers



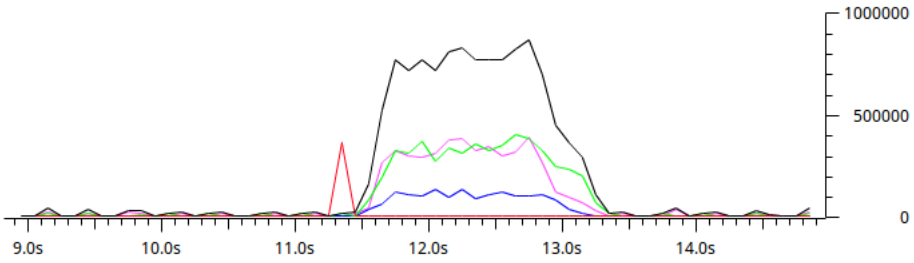
(e) 20 attackers



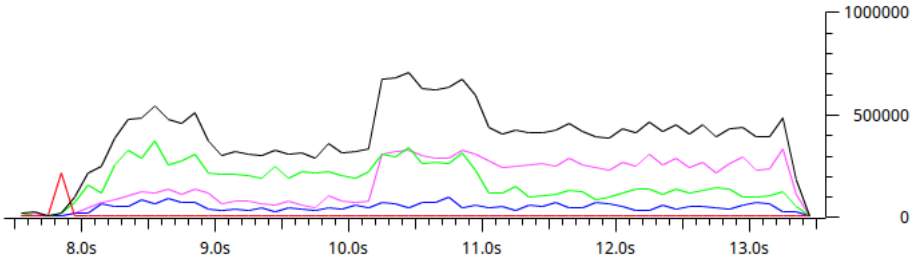
(f) 50 attackers



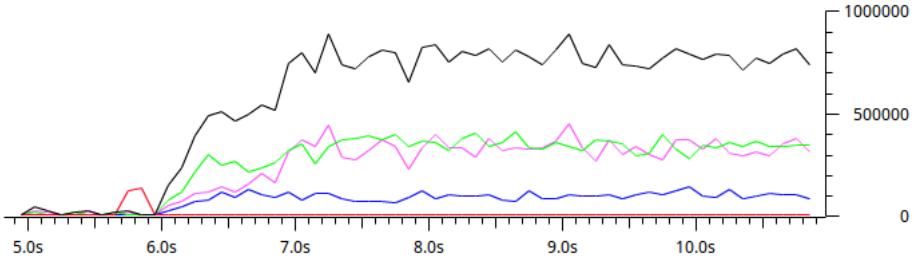
(g) 100 attackers



(h) 500 attackers

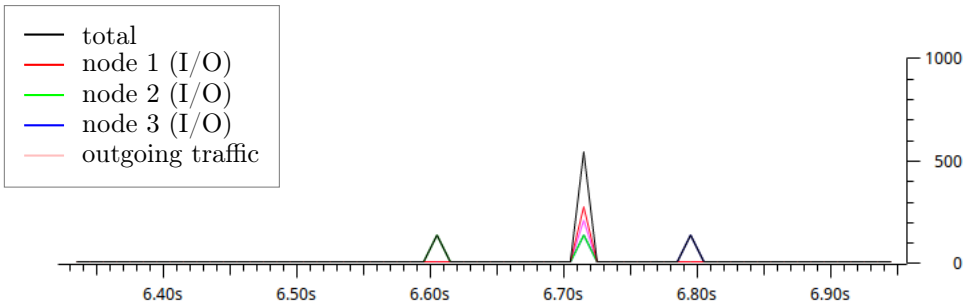


(i) 1000 attackers

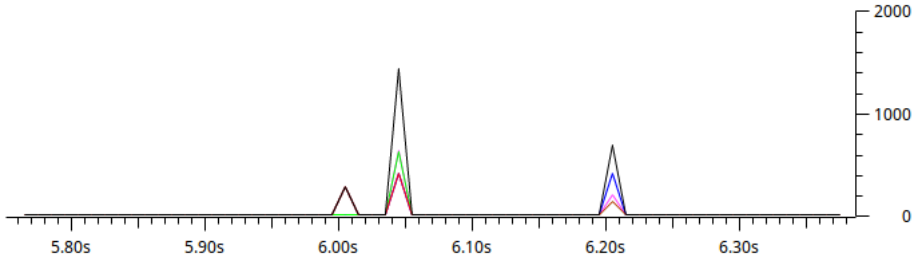


(j) 5000 attackers

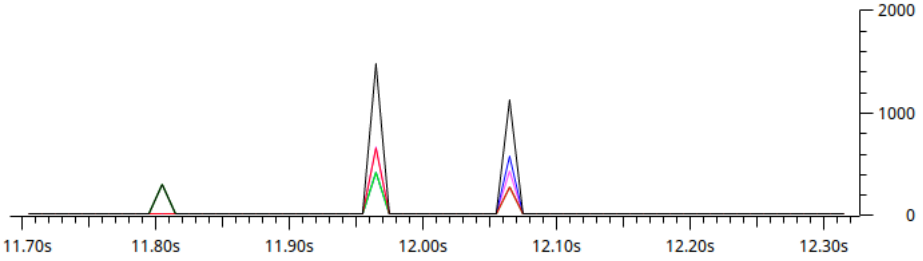
Figure B.0: Overhead with Chord



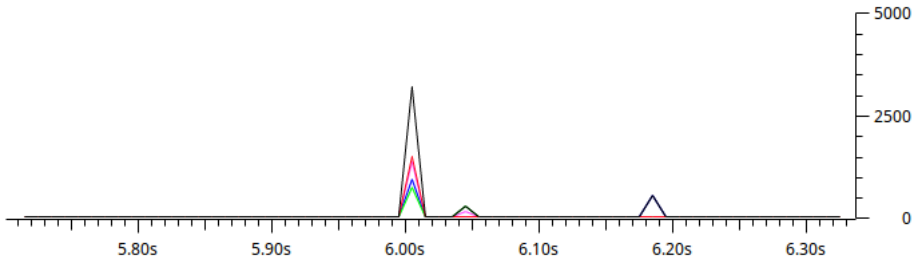
(a) 1 attacker



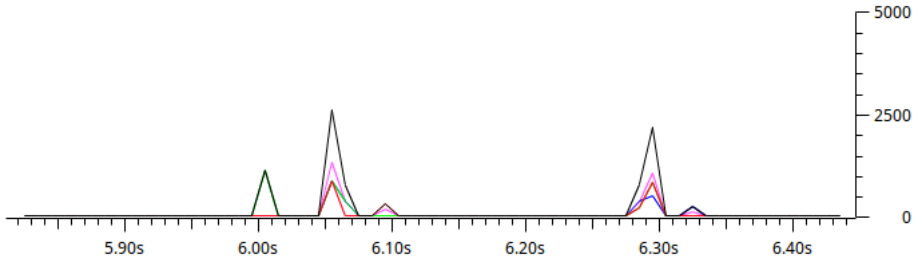
(b) 5 attackers



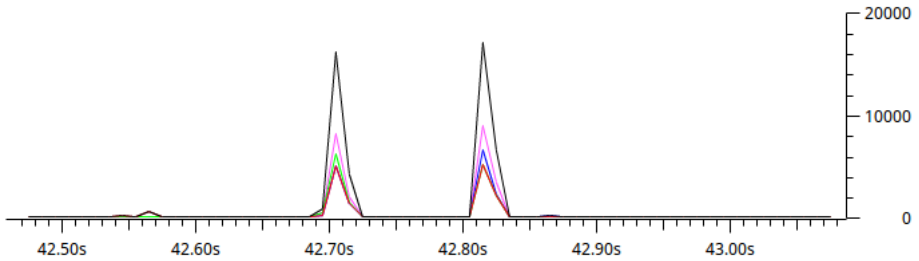
(c) 10 attackers



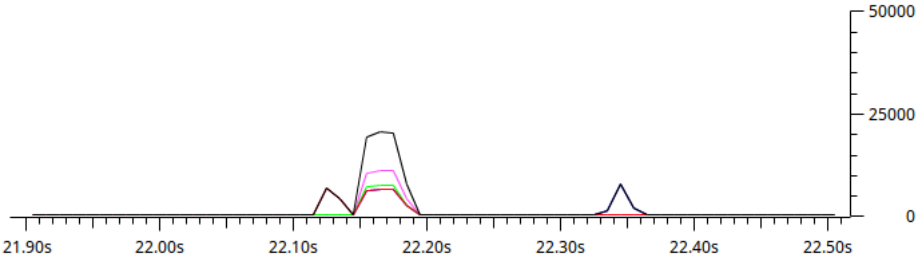
(d) 50 attackers



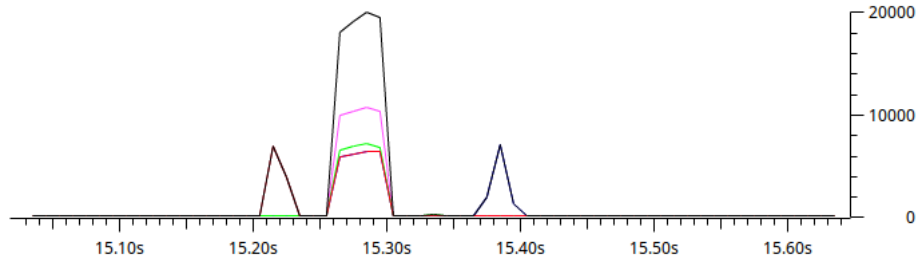
(e) 100 attackers



(f) 500 attackers



(g) 1000 attackers



(h) 1000 attackers (2)

Figure B.0: Overhead with centralized correlation unit

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**