# A Hybrid NIPS/NIDS for terabit networks

Fahimeh Alizadeh, Rawi Ramdhan

Mar. 10 2013

**Abstract**

Mitigating unwanted traffic on the Internet is a technical challenge. The large amount of traffic and the lack of hardware capable of inspecting this forms a problem. This paper discusses the limitations of a scalable system with commodity hardware that is capable of mitigating unwanted traffic on the Internet. A hybrid system is built that inspects traffic offline and in case of unwanted traffic triggers an event. This event ensures that traffic, from or to hosts, belonging to the unwanted traffic are inspected in real time. In order to disperse the large amount of traffic to single hosts that inspect this traffic, an OpenFlow switch was used. This paper shows that the load balancing via the FloodLight software and the OpenFlow protocol does not work. Therefor the scalabilty of this architecture could not be tested. However, information regarding traffic type dispersion on the Internet is provided and the traffic redirection methods are discussed. In conclusion we can say that Floodlight in combination with OpenFlow can not be used as a load balance mechanism. The method described to mitigate attacks on the Internet might still work but have to be tested in a scalable environment.

# Contents

# 1 Introduction

The Internet has provided the society with lots of benefits, but childporn [1], virusses [2] and compromised certificates [3] are just a few examples of the downside of the Internet. The Netherlands has several agencies, like the Nederlands Forensisch Instituut (NCSC), to mitigate this kind of Internet traffic. Recently the minister of safety and security stated that NCSC should build 24/7 detection networks [5]. No information was given on how this should look like, but is it even possible? Can we monitor the Internet? And if we can, can we prevent these things from happening?

Techniques to detect and prevent malicious Internet traffic exist, namely, Intrusion Detection system (IDS) and Intrusion Prevention system (IPS). These systems can operate for individual hosts but also for multiple hosts. The latter is referred to as Network Intrusion Detection System and Network Intrusion Prevention Systems (NIDS and NIPS).

Not only commercial products (see appendix B) but also open-source solutions are available (like BRO IDS [21] and Snort [22]). Most of them have limitations regarding throughput and scalability. Non of the commercial system scan scan at speeds needed to scan the Internet at a Internet Exchange (IX). Even though scalable solutions exist, they can scan up to 80Gb/s [8], a fraction of the data transmitted on a IX. One open-source product exists that is scalable, BRO Cluster. But is this scalable enough to scan the entire internet?

## 1.1 Related work

A lot of research has been done in this area. Optimization via regular expressions [15], logic [16], offload hardware [17] and clustering [18] have been researched. The latter describes a scalable NIDS. Our research proposes an architecture based on their NIDS, expanded with a NIPS.

## 1.2 Research question

The aim of this research is to understand the technical implications and bandwidth limitations when building a scalable intrusion detection and prevention system with commodity hardware for networks with multiple terabits of traffic per second.

In order to do this, the following questions have to be answered. (i) What algorithm for dispersing traffic over multiple intrusion detection systems has the highest bandwidth processing capacity, and can OpenFlow [13] be used

for this? (ii) Is it possible to precalculate the performance of an IDS with a given set of variables? (iii) Can BRO be used as an IPS?

## 1.3 Outline

Chapter two will contain the proposed architecture. This chapter will explain the possible methods to generate, disperse and inspect traffic. These methods will be implemented, tested and described in chapter 3 (Experimental Setups). This will result in conclusions and future work stated in chapter 4.

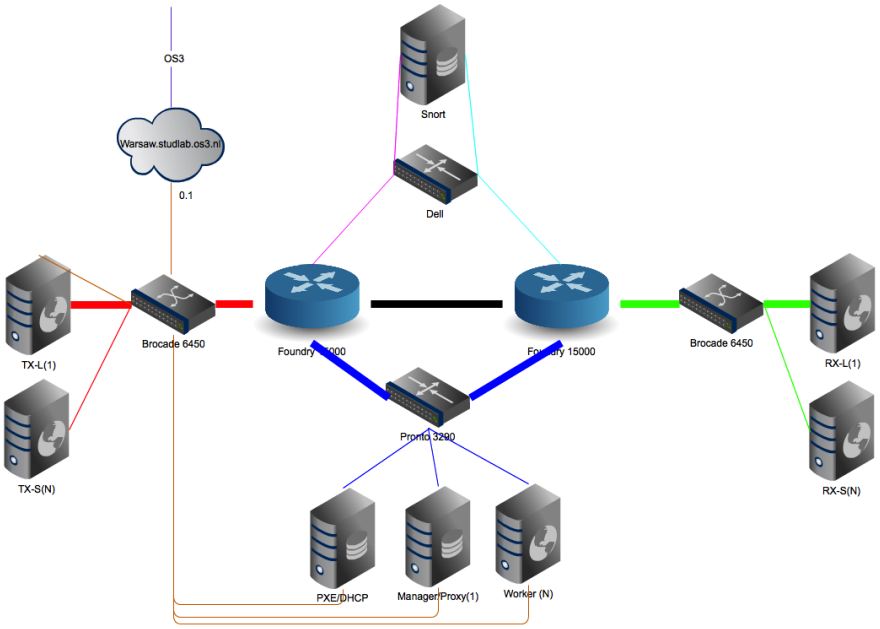


Figure 1: Architecture

# 2 Proposed Architecture

Figure 1 shows a graphical representation of the proposed architecture. The architecture is segmented in five areas; Sending, Receiving, Forwarding, Detecting and Preventing. The sending area (denoted in the figure as hosts starting with TX), will generate traffic and send it to the forwarding area. The forwarding area (two Foundry 15000 switches) will forward the data to the receiving side (hosts starting with RX). Every forwarded packet will also be copied to the inspection area. The inspection area consists of multiple hosts; PXE, Manager and Workers. This section will detect anomalies offline and inform the forwarding area to redirect traffic for those to the prevention area. Traffic redirected to the prevention area (SNORT host) will be first inspected in real time and then forwarded or dropped.

## 2.1 Traffic patterns on the Internet

Among other things this research describes the bandwidth limitations for a NIDS. In this research bandwidth is described as the amount of bits traversing a certain point in the network per second. However, this bandwidth can consist of a variable amount of packets. For example 1 Gigabit Ethernet (GbE) can contain anything from 81,274 to 1,488,096 packets per second [4].

All these packets can consist of different protocols and payloads. It is not possible to perform tests on a real Internet Exchange, therefore it is necessary to create a simulated environment with comparable characteristics. In order to test the NIDS with different types of bandwidth, full control over packet creation is needed.

We know the average throughput of different Internet exchange (IX) points [7] is 135.35 Gb/s (see appendix section C). IXs provide limited information on the type of traffic they forward. They either do not inspect traffic or are unwilling to provide this information. Multiple IXs have been contacted, AMS-IX provided the information on the amount of packets forwarded (figure: 2).

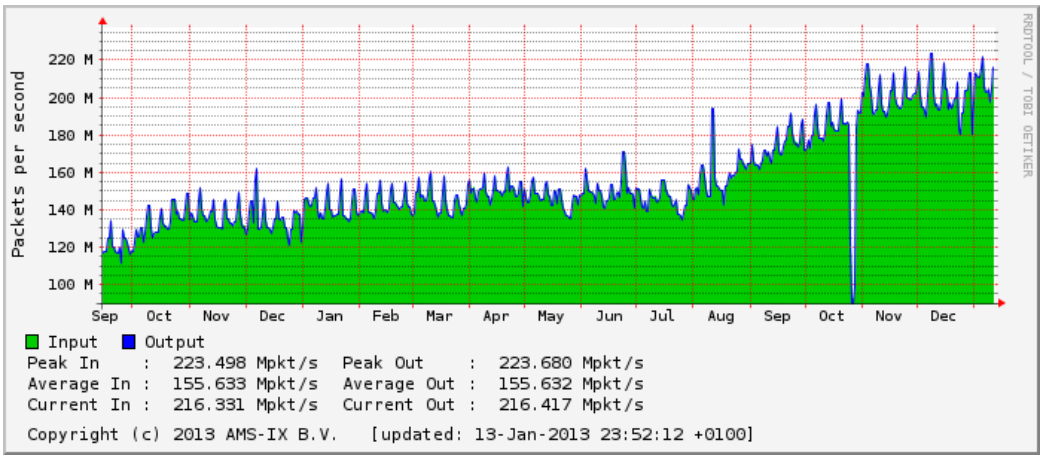


Figure 2: Troughput AMS-IX

Because of its size (the number of members [9], the volume of traffic [10]) and geographically distributed peers [11], are AMS-IX network traffic patterns a good model for Internet traffic.

AMS-IX provides public information about the frame length distribution of their packets (see appendix section A, figure: 7 [12]). They do provide other statistics [14], but no information is provided regarding protocols or payload.

In order to get more detailed information on the type of Internet traffic forwarded on the Internet, the ISP Surfnet[1] has been contacted. Surfnet, provides information regarding TCP, UDP and ICMP distribution of their traffic (see appendix section A, figure: 8). TCP has sessions, keeping track of sessions may be a tedious job for a NIDS and therefore resource and time consuming.

[1]Dutch ISP for research institutes, among others

## 2.2 Packet Generation

Traffic can be generated in multiple ways. By using client/server software like wget - Apache (HTTP) or dig - bind (DNS). An other option is to use packet generators such as Ostinato[2] or D-ITG[3]. With client/server software it is difficult to control the communication speed. Creation of the packets is not the only limiting factor but also packet processing and disk I/O. Packet generator provides full control over the speed and payload. Packets can be created in real-time where speed is limited by the CPU [6]. However, extra care has to be taken to create live like packet flows.

The generated traffic can be send in multiple ways, directly or replaying recorded packets. Sending packets directly is slower than replaying recorded packets. The generated packets have to pass trough the entire TCP/IP stack in order to be sent out to the network interface card (NIC). When replaying recorded packets the TCP/IP stack is bypassed which results in higher performance. However, session based flows are hard to synchronize this way.

## 2.3 Detection

There are a lot of commercial NIDSs and NIPSs available (see appendix section B). These systems detect events via two methods; Signature recognition, anomaly detection. The latter performs a statistical analysis on the data and triggers an event if abnormal patterns are detected. This method is useful to detect attacks that owe their effectiveness to a sequence of packets. Signature based recognition requires a "signature" of the unwanted data, every packet is compared to all the signatures.
In case of an IX, multiple Gb/s of traffic is sent. A part of this traffic may be malicious. Due to NAT in IPv4 this traffic does not have to be a single source but is detected as such. The proposed architecture is designed to prevent traffic up to 1 Gb/s from a single source. The total traffic stream can consist of multiple sources sending traffic up to 10 Gb/s.
There are different attack types, they try to compromise; Confidentiality, Integrity, Availability or Control [19]. An attack method to compromise availability is the a Denial of Service (DOS) attack. These attacks slog up resources and can be detected via anomaly detection [20]. Viruses for example, can be detected via signature based recognition. This requires deep packet inspecion (DPI) and is therefore less capable of handling high speed traffic. This research does not aim at a specific attack method but searches

[2]http://code.google.com/p/ostinato/
[3]http://traffic.comics.unina.it/software/ITG/

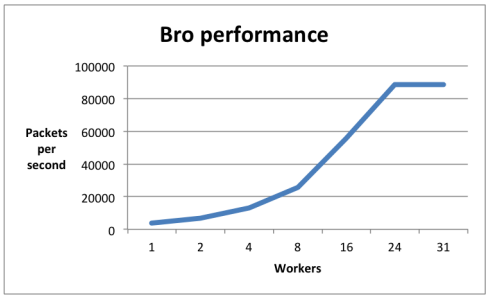


Figure 3: Packets inspected by Bro Cluster in seconds

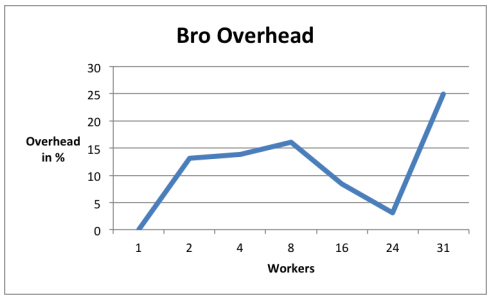


Figure 4: Percentage of overhead per worker in the Bro Cluster

for a scalable, signature based NIDS. A signature based NIDS can detect different attacks, as long as it has a signature for that attack.
The Bro cluster [21] is a free and open source NIDS that supports Signature recognition and anomaly detection. It is a scalable solution with limited overhead (Figure 4). Figure 3 shows the performance of Bro cluster. This test was executed with a single proxy, causing a bottleneck if more than 24 workers [23] are used. All signatures and anomalies can be programmed via the Bro programming language. This makes Bro a versatile NIDS capable of detecting a wide range of attacks.

The Bro cluster consists of three different node types; worker, proxy and a manager. The workers and proxies can be scaled up to multiple systems.

**The Worker** inspects traffic and generates events for possible attacks or suspicious patterns. These events are detected according to policies. These policies are defined using the Bro scripting language. Bro Cluster comes with a set of scripts to detect unwanted traffic. These events are sent to the manager. The performance of a worker is mostly limited to its CPU speed and RAM [18].

**The Manager** receives messages from the Workers and creates logs out of this. Its main purpose is to correlate the messages and logs. Getting results from one worker is not enough, it may be the case that one worker detects events from one source but the other does not. It is the manager's task to decide if that source is suspicious and if so, to contact the forwarding zone to redirect the traffic to the prevention zone.

**The Proxy** will establish the inter-worker communication. The need for proxies comes from the traffic distribution among workers. If the TCP-

sessions are not assigned to individual workers, then their status information has to be shared. The proxy will manage this situation. Every proxy is connected to a number of workers and all proxies together form a proxy ring which will result in connection between each pair of workers.

The scalable design of Bro Cluster results in a higher throughput [23]. The design does not scale linearly, the overhead is shown in figure 4.

## 2.4 Prevention

The detection of events might be useful. However, there are some situations in which detection is not enough and appropriate action must be taken. In the case of a NIPS, the action is to drop packets.
Current methods for intrusion prevention system work as standalone devices. This causes considerable limitations regarding scalability, performance, fault tolerance and real-time prevention speed. Even though the design (See figure 1) contains only one NIPS (Snort) instance, multiple can be added.

However, all will operate on a stand alone basis and will not communicate with each other. Therefore load distribution has to be done on IP basis by the switches in the forwarding zone. This ensures each session is redirected to the same IPS.
Snort is one of the most widely used intrusion prevention systems which is open source and free[4]. The intrusion prevention process is done in multiple layers:

1. **Packet decoding**: First the packets will be collected from different network interfaces.

2. **Preprocessing**: In this step, a fraction of incoming traffic will be selected. The selection is based on packet header information and some anomalies in packet header will be detected. Also defragmentation will be done to prepare the incoming packets for the next layer.

3. **Detection**: The payload inspection will occur in this phase. Snort supports two methods to detect and prevent events: Rule-based and Signature matching. The difference between rules and signatures is how they are defined. Rules refer to the weakness of the environment and by applying rules it is tried to cover the vulnerability of the environment which already motivates the attackers. The signatures refer to regular expressions matching incoming packet payloads which are defined based on the attacks done to the system. Rules and signatures will be applied on the result of the former step.

[4]http://www.sourcefire.com/security-technologies/open-source/snort

**Signature matching** Both Bro Cluster and Snort, provide the signature definition feature. In order to detect the same events, it is needed to define the same pattern matching signatures for them.

In the case of one standalone NIPS, managing all the connections coming from one source (The incoming traffic rate is the same as the NIPS process speed), combination of NIPS and NIDS would perform properly. By inc‘reasing the number of Snort systems, the NIPS zone will be able to sniff on multiple sources but the approach is to detect first and then prevent, so early intrusions will be only detected not prevented.

## 2.5 Load balancer

One of the ways to distribute load on the workers is via a load balancer. The load balancer will disperse traffic via different mechanisms to the workers. The fact that each NIDS has limitations in inspecting incoming packets, brings up the need for using a load balancer. The incoming 10Gb/s traffic can be split among several 1Gb/s workers that communicate their status and form the detecion area. The load balancing approach could be source-independent or source-dependent.

**Source-dependent load balancing** considers the packet header information and takes that into account. The distribution could be per-source, per-destination or per-connection. For example, formula 1 shows how the incoming packets will be distributed per connection among $n$ number of workers.

$$[Hash(src\_ip + dst\_ip + src\_port + dst\_port)]\%n \tag{1}$$

Then according to the load balancing method, the number of packet header fields involved in the former formula can be less or more. In order to choose the best load balancing method, it is ideal to have prior knowledge about the incoming traffic. For instance, if the traffic is evenly distributed among destinations, per-destination method fits the best with the environment.

**Source-independent load balancing** distributes the incoming traffic without the consideration of packet header information. There are several methods that accomplish this, a brief explanation on a few of them;

- Random selection: The load balancer will choose a random physical port from its outgoing ports set.
- Round-robin: The load balancer keeps the order of outgoing ports and chooses the outgoing port according to its sequence number.
- Weighted and dynamic round-robin: Two former methods do not take the incoming traffic information and workers status into account. According to this method, the load balancer keeps the order of outgoing ports but in a case that the worker is overloaded, the load balancer will skip that worker. Each worker is monitored continuously and according to its performance level, the traffic is sent to that worker.

There are two types of load balancers: software and hardware. Either they use source-dependent or source-independent mechanisms. Although hardware load balancers work faster but they do cost more. In this research we work with a Pronto 3290 switch as an openflow enabled switch.

### 2.5.1 OpenFlow switch

OpenFlow is an open standard for network devices. It separates the forwarding plain from the control plain. One server (the controller) takes the responsibility of the control plain and instructs the network devices via the OpenFlow protocol.

The OpenFlow switch keeps one flow table and each flow entry contains packet characteristics and the corresponding action (drop or forward the packet). When a packet arrives at an OpenFlow switch, the switch looks at its flow table. If it can not find the corresponding flow entry, it will redirect the packet to the controller. The controller will drop or redirect the packet and will notify the switch. The switch will take the same action for similar packets from that moment on. This whole process will occur if the switch is on its reactive method which provides dynamic flow entries but in the case of proactive method, all required flows will be defined in the switch directly via the controller before the traffic is sent through the switch. If the incoming packets do not match the defined flows, they will be dropped.

### 2.5.2 Floodlight controller

There are some open source options for the controller, such as NOX[5], Floodlight[6] and Flowscale[7].
Floodlight is a Java based Apache-licensed application which is tested in real networks and is being supported by a developer community. Floodlight consists of modules, the main module "FloodlightProvider" manages communication with switches and creates OpenFlow messages for receiving devices. It also interprets the arrived OpenFlow messages for taking appropriate action.
Unfortunately at the moment there is no straight-forward solution to make an OpenFlow enabled switch work as a load balancer. In this project, different methods are developed and tested to make this happen, which are explained in the following section.

[5]http://www.noxrepo.org
[6]http://floodlight.openflowhub.org
[7]http://www.openflowhub.org/display/FlowScale/FlowScale+Home

# 3 Experimental setups

## 3.1 Pronto switch

There is no plug-and-play solution to use the Pronto switch as a loadbalancer. Therefore, some ideas to implement this functionality have been researched.

**Load balancer module** From December 12, 2012, the floodlight version in GitHub contains a new module called "load balancer". This module is meant to work on ICMP, TCP and UDP flows. But it is still in development phase and its compatibility is not tested[8]. The idea behind this module is to define one virtual IP address (VIP) and assign a pool to that. The hosts responsible for the VIP are added to the pool. The hosts pushed in the pool will respond to the requests sent for that VIP.

The following shows the commands that are used in the controller server to create one virtual IP address which will respond to PING echo-requests:

```
# curl -X POST -d '{"id":"1", "name":"vip1", "protocol":"icmp",
    "address":"192.168.100.100", "port":"8"}' http://localhost
   :8080/quantum/v1.0/vips
# curl -X POST -d '{"id":"1", "name":"pool1", "protocol":"icmp
   ", "vip\_id":"1"}' http://localhost:8080/quantum/v1.0/pools
# curl -X POST -d '{"id":"1", "address":"192.168.100.2", "port
   ":"8", "pool\_id":"1"}' http://localhost:8080/quantum/v1.0/
   members
# curl -X POST -d '{"id":"2", "address":"192.168.100.10", "port
   ":"8", "pool\_id":"1"}' http://localhost:8080/quantum/v1.0/
   members
```

This module is still under development and its functionality is not fully guaranteed. This configuration does not work for the proposed architecture. When one host sends an ICMP echo request to the virtual address "192.168.100.100", the ICMP requests remain without any response. Also TCP and UDP protocol pools are created. The VIP is set to the IP address of the receiver and all Bro workers are pushed in the corresponding pools. They do not operate as it is explained in this module's description.

**Unknown unicast** means the switch replicates the incoming traffic to all outgoing ports. In this case, all workers receive the same traffic and they are responsible to take their part of the traffic. Unknown unicast needs more

[8] http://www.openflowhub.org/display/floodlightcontroller/Load+Balancer

complicated workers, capable of packet filtering. In comparison to a case that each worker receives its part of the incoming traffic, this method will operate in lower performance levels because each worker has to check all the packets and take its part. For this project it is not possible to use this method because the traffic is brought to the device in 10Gb/s and none of the workers can handle that rate. Hence the workers can receive the traffic up to 1Gb/s, the data loss and scalability issue would be challenging in this case.

Even if the incoming and outgoing ports were transferring in the same rate, there would be a problem with this method. Using unknown unicast, it can not be guaranteed that the amount of traffic put on each outgoing port is the same. This will result an unreliable situation which all workers do not reach their highest efficiency.

**StaticFlowEntryPusher module** Using this module, the flows are inserted manually. The Forwarding module is also disabled to prevent any packet transfer without matching the inserted flows. It is tested if the OpenFlow switch performs as a load balancer by modifying the inserted static flows and traffic distribution. Using this module, two methods are proposed which are explained:

1. **Real time flow management**: According to this mechanism, at first all the flows between the workers, the proxies and the manager are set and will remain unchanged. On the other side, flows are defined periodically from input physical ports set to the workers set. This flow definition process has to be done in an infinite loop and specific time span. As it is shown in figure 5, the gray lines indicate the fixed connections between components in the Bro Cluster. The dotted lines are the ones will be deleted and created to pass incoming traffic to the workers.

   There are multiple ways to define flows on the OpenFlow switch. One of them is to use the "curl" command. The following commands show two example flows from port 1 to port 2. The first flow only replicates the incoming traffic to port 1 into port 2 and does not check extra conditions. Second flow only replicates the packets with destination port number 80.

```
# curl -d '{"switch": "00:00:01:00:42:04:30:02", "name":"
   flow-1-2_v1","cookie":"0","ingress-port":"1","active":"
   true", "actions":"output=2"}' http://127.0.0.1:8080/wm/
   staticflowentrypusher/json
```

```
# curl -d '{"switch": "00:00:01:00:42:04:30:02", "name":"
   flow-1-2_v2","cookie":"0","ether-type":"0x0800","
   protocol":"6","dst-port":"80","ingress-port":"1","
   active":"true", "actions":"output=2"}' http
   ://127.0.0.1:8080/wm/staticflowentrypusher/json
```

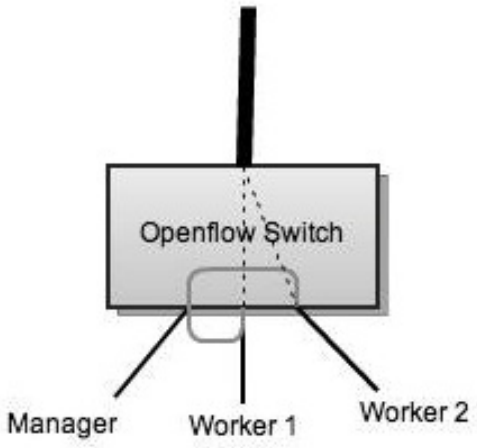


Figure 5: Realtime flow management

Before implementing this method, it was tried to set flows in a way that they contain real-time changing variables. In the flow definition, there is a field called "actions" which will be applied when it satisfies the condition. The egress port number is set in this field. To distribute the incoming traffic among the workers, it would be ideal if the egress port number could be set in real time. For example formula 2 distributes the incoming traffic per-destination ($n$ shows number of workers). The module StaticFlowEntryPusher does not provide dynamic flows. The flows have to be defined first then they will apply the corresponding actions.

$$[Hash(dst\_ip + dst\_port)]\%n \tag{2}$$

Therefore the only option for this project is to add and remove the flows as it is explained before. One python program is implemented for this purpose and is run in the controller server (See appendix section D). There are two 10Gb/s ports on the switch which are meant to transmit the traffic into the Bro Cluster. The workers are connected via 1Gb/s ports. The switch first removes the flows which their ingress-port is one of the 10Gb/s ports and then chooses the next worker according to the

| Expected gap duration | Real gap duration | | | | | |
|---|---|---|---|---|---|---|
| 1000 | 1030.199 | 1030.301 | 1030.694 | 1032.045 | 1039.184 | 1039.535 |
| 500 | 531.727 | 533.120 | 542.854 | 551.444 | 552.387 | 560.277 |
| 100 | 103.688 | 126.954 | 127.751 | 139.977 | 141.676 | 148.764 |
| 1 | 9.185 | 27.210 | 28.257 | 29.052 | 36.094 | 41.430 |

Table 1: Gap values with different time spans in miliseconds

load balancer algorithm(such as random or round-robin) and creates flows for that.

To run this 2-step process (delete the flows, create the new ones), the time span has to be tuned. The traffic is generated containing 20000 packets per seconds. Tests are run with different time span to check the required accuracy. Table 1 shows the gaps that occurred in one worker with different time spans. Each row shows the exact duration when a flow is deleted for that worker. First column indicates the expected gap duration in milliseconds. Columns 2 to 7 show tested gap duration in milliseconds. For example, row 1 shows the gap values for 1-second time span. It is not expected to see exactly 1 second gap. Although the time span is set in python program but the program will still need to run some other instructions which makes the gap a bit longer. But ideally it is expected to see the same value for each gap in one row which is not happening. As an example 1030.199 and 1039.184 are two different gap values from row 1 and their difference accuracy is about 10 milliseconds.

The 1-millisecond test states a very clear answer for the question: "Can real-time flow management be used as a load balancing mechanism?". As it is shown in the table, the gap values in this case follow a chaotic behavior. They increase as the time passes and get further from the expected value which is 1 millisecond.

The time span for the OpenFlow switch has to be in milliseconds to disperse incoming 10Gb/s traffic among 1Gb/s receiving ports. But as it has been shown, it is not possible with current OpenFlow technology.

2. **Port based flows**: It is necessary to create a method for the OpenFlow switch to make it work as a dynamic decision-maker load balancer. In order to do this and also to provide a suitable representative for Internet traffic, the random distribution could be in the incoming traffic or in the created flows.

The idea behind this method is to add all the flows before traffic transmission is started. The flows differ in their source port, destination port, ingress port and egress port. The port numbers are chosen from different sets. The ingress and egress ports show the port numbers on the switch. The ingress port is one of the two 10Gb/s ports and the egress port shows 1Gb/s ports connected to the workers. The source port and destination port number is any number from 1 to 65536.

In the case that source and destination port numbers are distributed randomly among flows, the OpenFlow switch works properly as a load balancer. For this purpose, the number of flows created in the switch has to be 65536. But there is a limitation in the number of flows can be inserted in the switch. There is a Python program written for this project to create 65536 flows in the switch and distribute the source and the destination port numbers among defined flows. In the pronto switch around 2000 flows can be inserted which is not enough for 65536 flows needed.

## 3.2 Traffic Type Impact

To test the impact of different traffic types on the IDS, multiple traffic flows have been generated. Because load could not be dispersed via the OpenFlow switch, impact has been tested on one worker. Five different types of traffic have been transmitted. Small and large TCP packets, small and large UDP packets and HTTP sessions. The packets are created with Ostinato[9], captured via TCPDump[10] and transmitted via TCPReplay[11]. This provides full control over the packet headers and increased throughput. All packets created via Ostinato contain the same payload (all zero). The TCP packets sent via Ostinato are dropped on the receiving side, no response is sent. The frame length described in table includes the Frame Check Sequence (FCS). The amount of traffic (in Mb/s) sent in table 2 is measured on the interface of the switch (Foundry in figure 1. For the HTTP session this also includes the reply. The sessions are created via Apache Benchmark (AB) and Apache server. AB sends a thousand HTTP-Get requests per second to the Apache server, which responds with the default "It works" page. This entire session is inspected by Bro.

Data represented in table 2 is captured via Capstat[12], run on the manager.

[9] http://code.google.com/p/ostinato/

[10] http://www.tcpdump.org/

[11] http://tcpreplay.synfin.net/

[12] `http://bro-ids.org/download/README.capstats.html`

| Traffic Type | Mb/s Sent | Mb/s Captured | KPPS Captured |
|---|---|---|---|
| UDP Packet with 64 Bytes | 602 | 234 | 636 |
| TCP Packet with 64 Bytes | 340 | 212 | 575 |
| UDP Packet with 1518 Bytes | 7301 | 973 | 81 |
| TCP Packet with 1518 Bytes | 6152 | 973 | 81 |
| HTTP Session | 342 | 236 | 298 |

Table 2: Capture speeds by Bro node

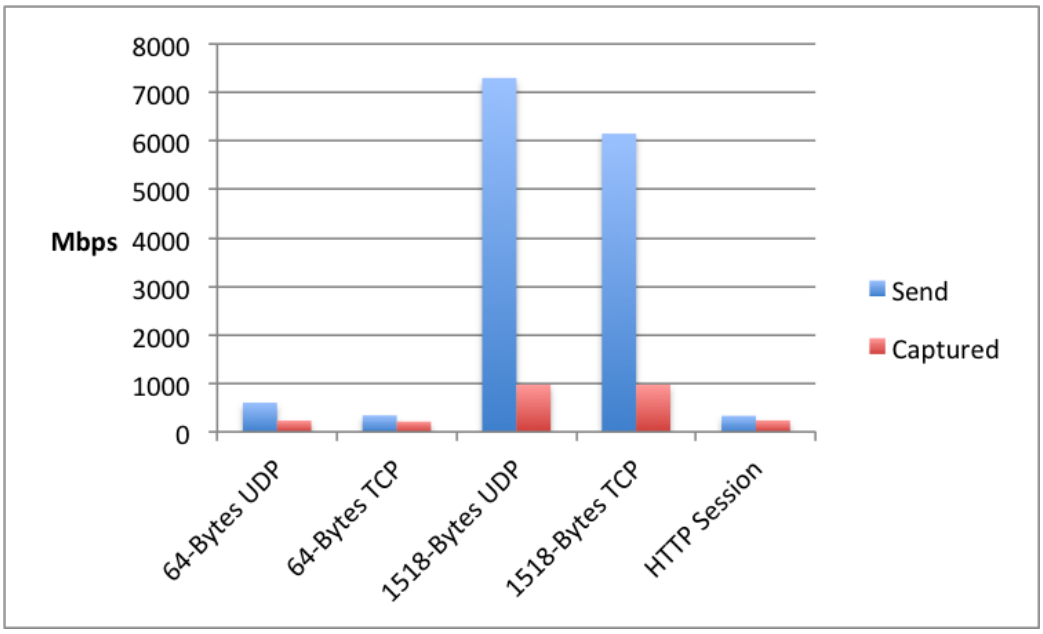


Figure 6: Traffic dispersion

The Bro host has a 1000BASE-T NIC. This explains the maximum captured traffic for large packets (1518 bytes), tests with a 10GBASE-T NIC should provide information on the maximum capture speed for large packets. For small packets (64 bytes) the maximum capture speed is 220 Mb/s. There is no significant difference in speed, for small packets regarding TCP or UDP.

The packets generated do not contain any payload that causes an action. Tests have shown that a worker can send a maximum of 1000 events per second where a script is called within 1 second. The script is called via the event triggered by Bro.

```
redef Notice::policy += {
       [$pred(n: Notice::Info) = {
              if ( n$note == TCP::TEST )
                     {
                     local cmd = fmt("/usr/local/bro/share/bro
                        /site/acl.sh");
                     piped_exec(cmd, fmt("%s", n$id$orig_h));
                     }
              return F;
        },
        $action=Notice::ACTION_NONE]
};
```

The Notice policy will define the action to apply after one signature is detected in the Bro cluster. As it is explained before, for each signature detection there must be a mechanism to redirect the packets from that specific source to the prevention area.

This scripts overrides the Notice policy. When the notice policy is triggered the file acl.sh is executed. This file contains a telnet script that adds a static route for the IP-address causing the notice. Then the packets coming from the source identified with that IP address will not be copied any more and will be sent directly to the prevention area.

# 4 Conclusion

This research is set out to understand the technical implications and bandwidth limitations by building a scalable intrusion detection and prevention system with commodity hardware for networks with multiple terabits of traffic per second.

A method has been used that provides full control over packet content and ensures high traffic throughput. In order to disperse this traffic over multiple inspection nodes, different load balancing algorithms and techniques have been researched. The combination of Openflow and Floodlight does not provide any working method for loadbalancing at this moment.

The impact of TCP and UDP traffic has been measured on a single inspection node. There is no significant difference in TCP or UDP traffic. The maximum amount of bandwidth that can be inspected is defined by the packet size. Small packets (64 bytes) can be inspected at a maximum rate of 220 Mb/s. The maximum Mb/s for large packets (1518 bytes) could not be determined because the NIC limited the inspection speed. However, we can conclude that the maximum Mb/s that can be inspected is equal or higher than 975 Mb/s.

There is no technical limitation to make BRO work as a NIDS. Currently there is no NIDS version of BRO known. The proposed architecture to combine a NIDS with a NIPS could not be tested with multiple inspection nodes. With a single inspection node a static route could be inserted to redirect traffic to a NIPS.

## 4.1 Future Work

Multiple load balancing algorithms have been researched. Further research has to be done in order to create a load balancing algorithm via OpenFlow. The scalabilty of Bro for multiple terabits of traffic could not be tested. Further research is needed to determine the overhead with multiple proxies. Because of the 1000Base-T NICs the maximum bandwidth that can be inspected with large (1518 bytes) packets could not be determined. Tests with faster NICs are needed to determine this.

# References


[1] Child Pornography - First report of the Dutch National Rapporteur. Bureau of the Dutch National Rapporteur, The Hague. 2011.

[2] Online threats in the last moth [Online] Available: http://www.securelist.com/en/statistics#/en/top20/wav/month

[3] Black Tulip - Report of the investigation into the DigiNotar Certificate Authority breach. Fox IT, Delft. 2012.

[4] Bandwidth, Packets Per Second, and Other Network Performance Metrics [Online]. Available: http://www.cisco.com/web/about/security/intelligence/network_performance_metrics.html

[5] NCSC moet netwerken overheid 24/7 gaan monitoren [Online]. Available: http://tweakers.net/nieuws/86792/ncsc-moet-netwerken-overheid-24-7-gaan-monitoren.html

[6] Foong, Annie P., et al. "TCP performance re-visited." Performance Analysis of Systems and Software, 2003. ISPASS. 2003 IEEE International Symposium on. IEEE, 2003.

[7] List of Internet exchange points by size [Online]. Available: http://en.wikipedia.org/wiki/List_of_Internet_exchange_points_by_size

[8] Cisco ASA 5500 Series Next Generation Firewalls [Online]. Available: http://www.cisco.com/en/US/products/ps6120/index.html

[9] Network (AS number) statistics [Online]. Available: https://www.ams-ix.net/connected_parties

[10] Statistics [Online]. Available: https://www.ams-ix.net/technical/statistics

[11] Peering around the globe [Online]. Available: https://www.ams-ix.net/connect-to-ams-ix/peering-around-the-globe

[12] Frame size distribution [Online]. Available: https://www.ams-ix.net/technical/statistics/sflow-stats/frame-size-distribution

[13] What is OpenFlow? [Online] Available: http://www.openflow.org/wp/learnmore/

[14] sFlow Stats [Online]. Available: https://www.ams-ix.net

[15] Vasiliadis et al, "Regular Expression Matching on Graphics Hardware for Intrusion Detection," Saint-Malo, France. 2009.

[16] Clark, Christopher, and David Schimmel. "Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns." Field Programmable Logic and Application (2003).

[17] Song, Haoyu, et al. "Snort offloader: A reconfigurable hardware NIDS filter." Field Programmable Logic and Applications, 2005. International Conference on. IEEE, 2005.

[18] Vallentin, Matthias, et al. "The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware." Recent Advances in Intrusion Detection. Springer Berlin/Heidelberg, 2007.

[19] Kumar, Sailesh. "Survey of current network intrusion detection techniques." (2007): 1-18.

[20] Lakhina, Anukool, Mark Crovella, and Christophe Diot. "Mining anomalies using traffic feature distributions." ACM SIGCOMM Computer Communication Review. Vol. 35. No. 4. ACM, 2005.

[21] The Bro Network Security Monitor. [Online] Available: `http://www.bro-ids.org/`

[22] What is Snort? [Online] Available: `http://www.snort.org/`

[23] Weaver, Nicholas, and Robin Sommer. "Stress testing cluster Bro." DETER workshop. 2007.

# A Packet distribution

# B NIDS product list

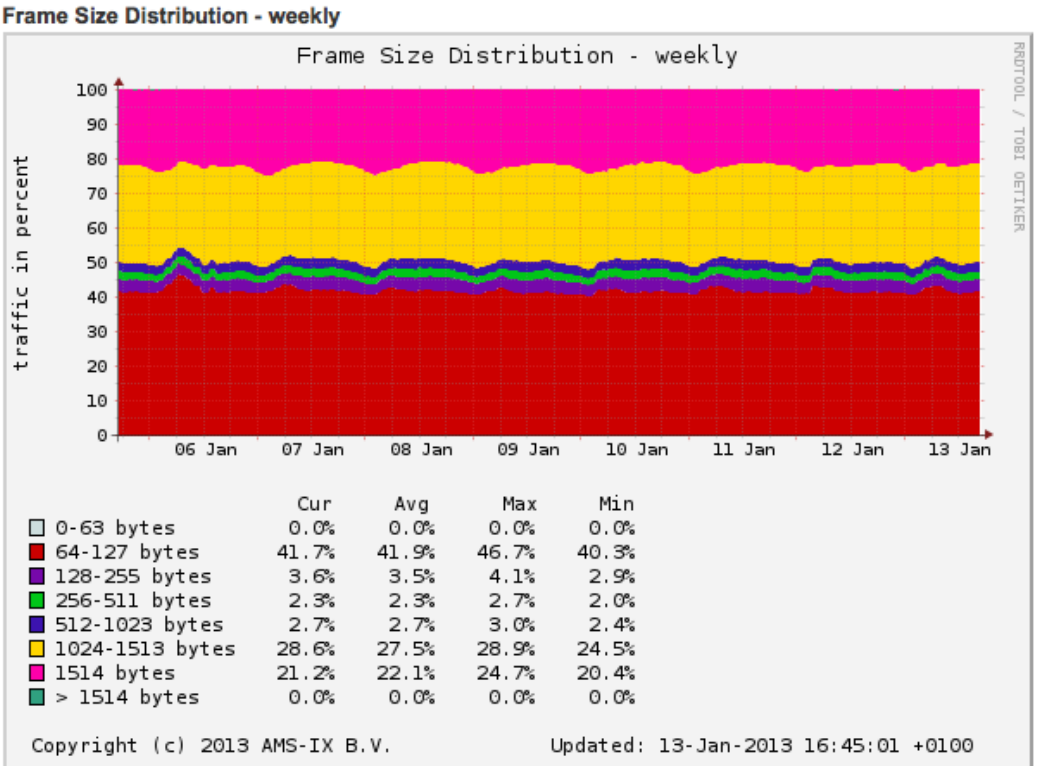


Figure 7: Frame size distribution in AMS-IX

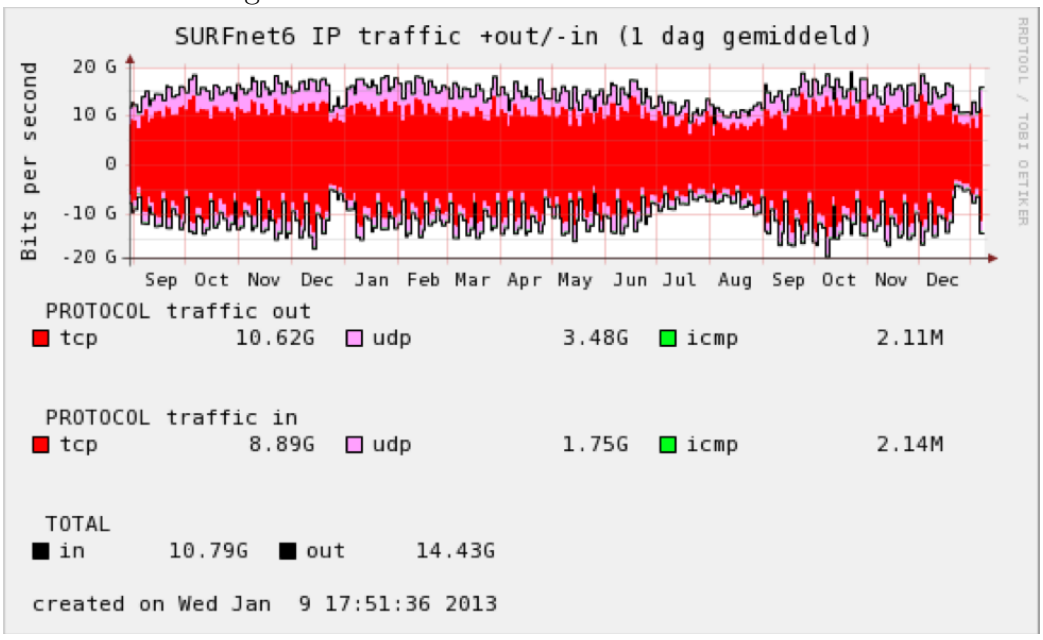


Figure 8: Network protocol distribution in Surfnet


## Intrusion Prevention Comparison Guide

| COMPANY | CONTACT | PRODUCT | DESCRIPTION |
|---|---|---|---|
| **TippingPoint** (A 3Com company) | 512-681-8000<br>888-878-3477<br>www.tippingpoint.com | TippingPoint IPS | Produces a widely-deployed device that is based on the company's own Threat Suppression Engine, custom ASICs and single vulnerability security filters |
| **McAfee** | 800-338-8754<br>www.mcafee.com/us/ | IntruShield Network IPS + Host Intrusion Prevention | Host solution uses a single software agent for both desktop and server protection in concert with Microsoft Windows stateful firewalls |
| **Internet Security Systems** (An IBM company) | 404-236-2700<br>www.iss.net | Proventia | One of the oldest security companies with a product that allows in-line simulation to determine what the best blocking behavior is before activating the blocking |
| **Sourcefire** | 800-917-4143<br>www.sourcefire.com | Sourcefire Intrusion Sensor | Sourcefire devices use the popular open-source Snort engine, which was created by the company's founder |
| **Juniper Networks** | 408-745-2000<br>www.juniper.net | Juniper Networks IDP and ISG series | An IP network systems provider, its IPS systems also provide views of network and application-level activity to help determine the cause of attacks |
| **Top Layer Networks** | 508-870-1300<br>www.toplayer.com | IPS 5500 series | Offers systems that content based IPS with stateful firewalls and mitigation algorithms to defend against content, access and rate-based attacks at the same time |
| **Cisco Systems** | 408-526-4000<br>www.cisco.com | Cisco Intrusion Prevention Solution | The top network infrastructure provider, the company's IPS is an integral part of Cisco's network security system |
| **Check Point Software** | 800-429-4391<br>www.checkpoint.com | IPS-1 and InterSpect | One of the leading firewall vendors, Check Point provides IPS either as a dedicated solution or combined with network access control |
| **Force 10** | 408-571-3500<br>www.force10networks.com | P-Series | A provider of network systems, Force 10 several years ago acquired the first IPS solution capable of 10 Gbps line-rate performance |
| **Enterasys** | 978-684-1000<br>www.enterasys.com | Dragon | Founded in 1983, the company now develops secure network technology with IPS delivered either as host or network-based defense |

Copyright © 2007, Tippit, Inc., All Rights Reserved.

# C Average troughput IX

| Short name | Name | Throughput | Throughput (Gbit/s) a | Values updated |
|---|---|---|---|---|
| BNIX | Belgium Nati | 51[101] | N/A | 41171 |
| DIX | Danish Interr | 29[129] | N/A | 41171 |
| DTEL-IX | Digital Telecc | 30 | N/A | 40774 |
| INXS | Internet Excl | 21[145] | N/A | 41171 |
| RheintalIX | Rheintal Inte | N/A | N/A | 41310 |
| SFINX | Service for Fr | 38[113] | N/A | 41171 |
| CATNIX | Catalunya Ne | 1[214] | 1[214] | 40919 |
| CIXP | CERN Interne | 2[206] | 1[206] | 40518 |
| JINX | Johannesbur | 2[217] | 1[217] | 40782 |
| Moebius | PTT Moebius | 2[203] | 1[203] | 40581 |
| Pacific Wave | Seattle Netw | 8[180] | 1 | |
| VSIX | North East N | 1[211] | 1[211] | 40449 |
| GN-IX | Groningen In | 7[184] | 2[185] | 40528 |
| IIX | Israeli Intern | 3[200] | 2[200] | 40549 |
| TIX | Telehouse In | 4[194] | 2 | |
| TWIX | Taiwan Inter | 3[197] | 2[197] | 40549 |
| AtlantaIX | Atlanta Inter | 9[171] | 3 | |
| Kh-IX | Kharkov Inte | 5[191][27] | 3[191] | 40488 |
| LAIIX | Los Angeles I | 9[174] | 4[174] | 40549 |
| PIRIX | PIRIX Interne | 7[188] | 4[188] | 40687 |
| ALP-IX | ALPs Interne | 13[162] | 5[162] | 40606 |
| EKT-IX | Ekaterinburg | 9[177] | 5 | 40687 |
| NFX | Neutral czFre | 16[151] | 7[151] | 41171 |
| RoNIX | Romanian Ne | 11[168] | 7[168] | 40549 |
| BCIX | Berlin Comm | 35[116] | 8 | 41117 |
| GigaPix | Gigabit Portu | 13[165][27] | 8 | 41310 |
| GR-IX | Greek Intern | 21[142] | 8[142] | 41171 |
| LONAP | London Netw | 25[136] | 8 | 40969 |
| NaMeX | Nautilus Mec | 21[148] | 8[148] | 41171 |
| NIXI | National Inte | 14[157] | 9[157] | 40650 |
| TOP-IX | Torino Piemc | 33[122] | 10[122] | 40549 |
| InterLAN | Romanian In | 25[133][27] | 11 | 40549 |
| PARIX | Paris Interne | 15[154] | 11 | 38989 |
| INEX | Internet Ne | 24[139] | 13[139] | 40708 |
| Pipe IX | Pipe IX[158] | 13[159] | 13 | |
| SIX | Slovenian Int | 35[119] | 16[119] | 40979 |
| FICIX | Finnish Comr | 31[126][125] | 18[125] | 41171 |
| NIX | Norwegian Ir | 39[107] | 20 | 40657 |
| SPB-IX | Saint-Peterst | 39[110] | 20 | 40687 |
| SwissIX | Swiss Interne | 43[104] | 22.5[104] | 41017 |
| SIX | Slovak Intern | 59[95] | 26 | 41171 |

| | | | | |
|---|---|---|---|---|
| SVAO-IX | SVAO-IX[87] | 61[89] | 28[89] | 41171 |
| BalkanIX | Balkan Interr | 55[98] | 30[98] | 41171 |
| Home-IX | Home-IX[90] | 60[92] | 31[92] | 41171 |
| BIX.BG | Bulgarian Int | 79[83] | 40[83] | 41171 |
| MIX | Milan Interne | 94[79][80] | 40[80] | 40865 |
| VIX | Vienna Interr | 73[86] | 42 | 41171 |
| PTT Metro | PTT Metro[7 | 100[76] | 60[76] | 41015 |
| ECIX | European Co | 107[73] | 64 | 41043 |
| TorIX | The Toronto | 113[70] | 70 | 41300 |
| HKIX | Hong Kong Ir | 182[55] | 71 | 40865 |
| BIX | Budapest Int | 160[61] | 75[61] | 40865 |
| SIX | Seattle Interr | 155[67] | 78[67] | |
| France-IX | France-IX[62 | 158[64] | 86[64] | 41277 |
| JPIX | Japan Interne | 167[58] | 88 | 40864 |
| NYIIX | New York Int | 282[48] | 93[48] | 40864 |
| Any2 | Any2 Exchan | 250[45] | 100 | 40865 |
| NL-ix | Netherlands | 188[51] | 113[51] | 41095 |
| NIX.CZ | Neutral Inter | 250[42] | 121 | 41232 |
| PLIX | Polish Intern | 220[39] | 132[39] | 40958 |
| ESPANIX | Spain Interne | 259[34] | 159 | 40966 |
| UA-IX | Ukrainian Int | 369[26][27] | 197[26] | 40865 |
| JPNAP | Japan Netwo | 363[31][30] | 203[30] | 41314 |
| NL-ix | Netherlands | 226[36] | 220[36] | 41314 |
| Netnod | Netnod Inter | 515[23][22] | 290[22] | 41314 |
| MSK-IX | Moscow Inte | 903[19] | 417[19] | 40986 |
| Data IX | Data IX[14] | 1100[16] | 756[16] | 41289 |
| Equinix | Equinix Exch | 1409[13] | 990[13] | 40954 |
| LINX | London Inter | 1574[10] | 1040[10] | 41218 |
| DE-CIX | Deutscher Cc | 2232[3] | 1369[3] | 41218 |
| AMS-IX | Amsterdam I | 2091[7] | 1379[7] | 41218 |
| | | | Average: 135.35 | |

# D Realtime flow management using Floodlight

```
#!/usr/bin/python
from random import randrange
import time
import os

class Load_Balancer:
       worker_ports = []
       worker_order = []
       in_port1 = 50
       in_port2 = 50
       rand = 0
       time_span = 1000000
       pronto_mac = "00:00:01:00:42:04:30:02"
       index_flow = 0
       def __init__(self):
              self.time_span = raw_input("Please␣enter␣the␣
                 time␣span(in␣seconds)␣you␣want␣to␣change␣the␣
                 flows:\n")
              self.rand = raw_input("Do␣you␣like␣random␣load␣
                 balancing?␣1/0\n")
              worker_port = raw_input("worker␣port␣(If␣you␣
                 have␣added␣all␣of␣them,␣enter␣-1):␣\n")
              while worker_port != "-1" :
                     self.worker_ports.append(worker_port.
                        rstrip('\r\n'))
                     worker_port = raw_input()
              # initializing the worker_order array
              self.worker_order = [None] * len(self.
                 worker_ports)
              if self.rand == "1":
                     self.random_order()
              else:
                     self.round_robin_order()
              self.flows_in_timespan()
# This function changes the order of worker ports in a random
    way
       def random_order(self):
```

```
                ind = 0 # indexes start from 0 in python
                while (ind < len(self.worker_ports)):
                        self.worker_order[ind] = randrange(0, len
                            (self.worker_ports) )
                        ind = ind + 1
                ind = 0
                while (ind < len(self.worker_ports)):
                        wp = self.worker_ports[self.worker_order[
                            ind]]
                        self.worker_order[ind] = wp
                        ind = ind + 1

# This function puts the worker ports in the order user
    entered them
        def round_robin_order(self):
                # I keep the order of worker ports as it is
                    entered by user
                ind = 0
                while (ind < len(self.worker_ports)):
                        self.worker_order[ind] = self.
                            worker_ports[ind]
                        ind = ind + 1

# This function will clear the switch from flows, create new
    one for the specific worker
        def flow_management(self):
                worker = self.worker_order[self.index_flow]
                name = str(self.in_port1) + "_" + worker
                create_flows = "curl␣-d␣'{\"switch\":␣\"" + self
                    .pronto_mac + "\",␣\"name\":\"" + name + "
                    \",\"cookie\":\"0\",␣\"ingress-port\":\"" +
                    str(self.in_port1) + "\",\"active\":\"true
                    \",\"actions\":\"output=" + worker + "\"}'␣
                    http://127.0.0.1:8080/wm/
                    staticflowentrypusher/json"
                os.system("curl␣http://127.0.0.1:8080/wm/
                    staticflowentrypusher/clear/"+ str(self.
                    pronto_mac) +"/json")
                os.system(create_flows)
                if self.rand == "1" and self.index_flow == len(
                    self.worker_ports):
```

```
                self.random_order()
            if self.index_flow == len(self.worker_ports) -1
                :
                self.index_flow = 0
            else:
                self.index_flow = self.index_flow + 1

# This function will do the job in an infinite loop
        def flows_in_timespan(self):
            while True:
                self.flow_management()
                time.sleep(float(self.time_span))

##########################
x = Load_Balancer()
```