nProbe™ Cento

light-speed 100 Gbps Probe for IPv4/v6

User Guide

Version 1.0

June 2016

Overview 4
Main Features 5
Installation 6
Executables 6
The cento Executable 6
The cento-ids Executable 6
The cento-bridge Executable 6
Licensing 7
Definitions 8
Use Cases 10
100Gbps Flow Exporter 12
Integration with ntopng 13
Integration with a NetFlow Collector 14
Flows Injection in Apache Kafka 14
Flows Dump to Plain Text Files 16
Flows Dump to Syslog 17
Full-Duplex TAP Aggregator + 100Gbps Probe 18
TAP-Aggregated Flows Export to a Netflow Collector 19
100Gbps Probe + Traffic Aggregator 20
Packet-to-Disk Recording 20
Policed Packet-To-Disk Recording 23
100Gbps Probe + Traffic Balancer for IDS / IPS 25
Integration with Suricata IDS/IPS 26
Integration with Snort IDS/IPS 26
Egress Queues 27
Policing Egress Queues Traffic 28
Policy Rules 28
The Egress Queues Configuration File 30
Shunting 30
Aggregated Egress Queue 31
Balanced Egress Queues 32
The Egress Queues runtime REST Configuration API 33
Identifying the base REST Endpoint 33
Configuring Queue-Level Rules 34
Configuring Subnet-Level Rules 34
Configuring Protocol-Level Rules 34
Network Bridge 36
Policing Bridged Traffic 36
Policy Rules 37
The Network Bridge Configuration File 38
Network Bridge Example 39
The Network Bridge Runtime REST Configuration API 40
Identifying the base REST Endpoint 40
Configuring Bridge-Level Rules 40
Configuring Subnet-Level Rules 40
Configuring Protocol-Level Rules 41
Command Line Options 42
Interfaces 42
Egress queues 43
Flows Generation 45
CPU Affinity 46
Flows Export Settings 47

Miscellaneous Settings 51
Zero Copy 53
REST 53

Overview

Measuring network traffic is a fundamental task in any modern packet-switched network. Accurate measurements offer an effective support in the timely diagnosis of network issues. Misbehaving hosts, faulty adapters, intruders, undesired traffic, are just a few examples of issues that are likely to occur in any real-world deployment. Other popular use cases that demand for accurate traffic monitoring include, but are not limited to, billing and reporting systems used by service providers and network operators.

The steady increase in network and adapter speeds is determining an alway increasing demand for novel measurement systems that are able to keep up with ultra-high-speed environments. Nowadays, 40 and 100 Gbps adapters are rapidly becoming popular and will slowly replace the mainstream 10 Gbps adapters. Being able to process traffic at these speeds in software requires to deeply reconsider the well-established design choices in order to leverage on modern multi-CPU, multi-core elaboration systems.

Another challenge that is currently being faced is the ability to actively measure network traffic. Passive measurements no longer suffice as they do provide answers but, at the same time, fail to provide timely solutions. Ex post analyses are still useful but can't prevent or alleviate the effects of an undergoing network issue. For these reasons, there is an increasing interest in systems that are able to proactively react and decide on the basis of monitored data such as, for example, systems that shape or even isolate an attacker in a network.

nProbe™ Cento is a high-speed NetFlow probe that has been designed to address the issues above. Cento is able to keep up with 1/10/40/100 Gbps adapters. However, it is not just a fast NetFlow probe, it has been designed as the first component of a modular monitoring system: besides capturing ingress packets and computing flow data, it can be used to classify the traffic via DPI (Deep Packet Inspection) and to optionally perform actions based on selected packets/flows when it is used as a traffic forwarder in combination with other applications such as IPS/IDS and traffic recorders.

Main Features

nProbe™ Cento features include:

- NetFlow v5/v9/IPFIX export support.
- Export also in JSON, Text, Kafka, Syslog.
- IPv4 and IPv6 support.
- Native PF_RING and PF_RING ZC support for high-speed packet processing.
- Support of FPGA-based network adapters.
- A scalable, multithreaded design: ingress traffic load can be balanced across multiple streams on multicore architectures.
- Full Layer-7 application visibility using nDPI (Deep Packet Inspection).
- Lightweight Layer-7 application visibility using *μ*-nDPI (a lightweight DPI library supporting the most important protocols such as HTTP/HTTPS/DNS).
- Flow-based Load Balancing to Intrusion Detection and Intrusion Prevention Systems IDS/IPS including Snort, Bro, and Suricata.
- Traffic filtering based on Layer-7 protocols, CIDR, and interfaces to reduce the load on the IDS/IPS.
- Feedback channel for traffic filtering/shunting from an IPS: "forward this flow", "drop this flow", "divert this flow through the IPS".
- Traffic filtering and slicing for saving storage space removing meaningless data when forwarding traffic to a packet-to-disk application such as n2disk.

Installation

nProbe™ Cento must be installed on a vantage point that can monitor all the traffic of interest. In packet-switched networks, it is common practice to either mirror the traffic, or use a network TAP.

Packages and installation instructions are available at http://packages.ntop.org/ for all supported distributions.

Once the installation is completed it is necessary to generate an nProbe™ Cento license otherwise the probe will operate in demo mode. A license is required for the probe to be fully operational. Any license is bound to a given system as it is created from a *"system id"* that, in turn, is generated by combining together the hardware details.

Executables

nProbe™ Cento comes in three separate executables, namely, `cento`, `cento-ids` and `cento-bridge`. Every executable is responsible for carrying out a particular task. The main features of every executable are briefly discussed in the remainder of this section. Detailed use cases, examples, and usage guides are reported in the rest of this manual.

The cento Executable

The executable `cento` features a no-frills 100Gbps NetFlow v5/v9 IPFIX Flow probe. This executable is recommended for users who are interested in ultra-high-speed packet capture and flow generation and who do not plan to use nProbe™ Cento inline as a bridge, or in combination with IDS/IPS or packet recording systems such as n2disk.

The cento-ids Executable

The executable `cento-ids` still features a 100Gbps NetFlow v5/v9 IPFIX Flow probe, but it also has special options that allow to balance and aggregate input traffic towards one or more output queues. This is particularly useful when nProbe™ Cento has to be used in combination with IDS/IPS and packet recorders. Use cases and examples will be discussed with great deals below.

The cento-bridge Executable

The executable `cento-bridge` has, in addition to the 100Gbps NetFlow v5/v9 IPFIX Flow probe features, peculiar traffic bridging capabilities. This executable should be chosen by users who need to policy traffic (e.g., filter out a certain application) in ultra-high-speed environments. Bridging examples and use cases are given in the remainder of this guide.

Licensing

nProbe™ Cento requires a per-system license that is released according to the EULA (End User License Agreement) as specified in the appendix. Each license is perpetual (i.e. it does not expire) and it allows to install updates for one year since the license issue. This means that a license generated on the 1/1/2016 will be valid for any update released until 12/31/2016. The purchase of a new license it is required to install updates released after 1 year from the initial license issue.

Licenses are available in various flavors, depending on the number and speed of network interfaces used to feed nProbe™ Cento. The following table outlines the licensees available

Max. Simultaneous ports	nProbe™ Cento S	nProbe™ Cento M	nProbe™ Cento L	nProbe™ Cento XL	nProbe™ Cento XXL
1Gbps	2	Unlimited	Unlimited	Unlimited	Unlimited
10Gbps	-	2	Unlimited	Unlimited	Unlimited
40Gbps	-	-	2	Unlimited	Unlimited
100Gbps	-	-	-	2	Unlimited

nProbe™ Cento licenses can be generated using the order id received and the email address provided when purchasing products at https://shop.ntop.org/. License generation page is https://shop.ntop.org/mklicense.

Definitions

Term	Description
Aggregated Egress Queue	A queue that is output by nProbe™ Cento which carries traffic that has been aggregated from multiple input interfaces.
Balanced Egress Queue	A queue that is output by nProbe™ Cento which carries a subset of traffic received from an input interface. The subset is build to make sure packets belonging to the same flow are always forwarded to the same balanced egress queue.
Collector	Shorthand for flow collector.
Egress Queue	A queue that is output by nProbe™ Cento and is consumed by some other software such and an IDS/IPS or a traffic recorder.
Exporter	Shorthand for flow exporter.
Flow	Network packets can be aggregated into logical pipes termed “flows”. A flow is uniquely identified by: source and destination IP addresses, source and destination ports, and layer 4 protocol.
Flow exporter	A piece of hardware/software that outputs flows to a medium (e.g., over the network, to file, to other other software).
Flow collector	A piece of hardware/software that collects flows from a medium (e.g., from the network, from file, from other software).
IDS	An Intrusion Detection System that detects known threats, policy violations and malicious behaviors.
IPFIX	The Internet Protocol Flow Information Export (IPFIX) is a protocol that defines how to transfer flow data from an exporter to a collector.
IPS	An Intrusion Prevention System that protects the network against possible known threats, policy violators and malicious hosts.
Kafka	A multi-producer, multi-consumer, publish-subscribe distributed messaging system.
n2disk	The ntop high-performance packet-to-disk software that records network packets to disk and indexes packet metadata in near realtime to enable fast searches.
NetFlow v5/v9	Standards that define and describe how to aggregate packets into flows, and how to transfer flow data from an exporter to a collector.
ntopng	The ntop network traffic visualization software.
Packet-to-Disk	The act of writing full network packets (i.e., headers and payloads at any level) to persistent storage. See also traffic recorder.
Probe	Shorthand for Flow exporter.
Shunting	The act of filtering network packets that limits the number of per-flow packets to a given fixed value k. Any flow packet that arrives after the k-th is dropped.
Snort	An open source network IDS for Unix and Windows.
Suricata	An IDS/IPS to match on known threats, policy violations and malicious behavior.
Slice-l3	The act of filtering network packets that truncates packets right after the IP headers.
Slice-l4	The act of filtering network packets that truncates packets right after the TCP/UDP headers.
Syslog	A standard for message logging.

Term	Description
Traffic Recorder	A piece of hardware/software that writes network packets to persistent storage (e.g., HDD, SSD, nVME) for archiving purposes or further processing.
TAP	A network TAP (Test Access Point) is a hardware device inserted at a specific point in the network to monitor full-duplex data.

Use Cases

In this section is listed a comprehensive, yet not exhaustive, set of nProbe™ Cento use cases. Use cases range from quick network flow export to more sophisticated integrations with traffic recorders and intrusion detection / intrusion prevention systems (IDS/IPS). The great flexibility of nProbe™ Cento will be able to effectively provide solutions to:

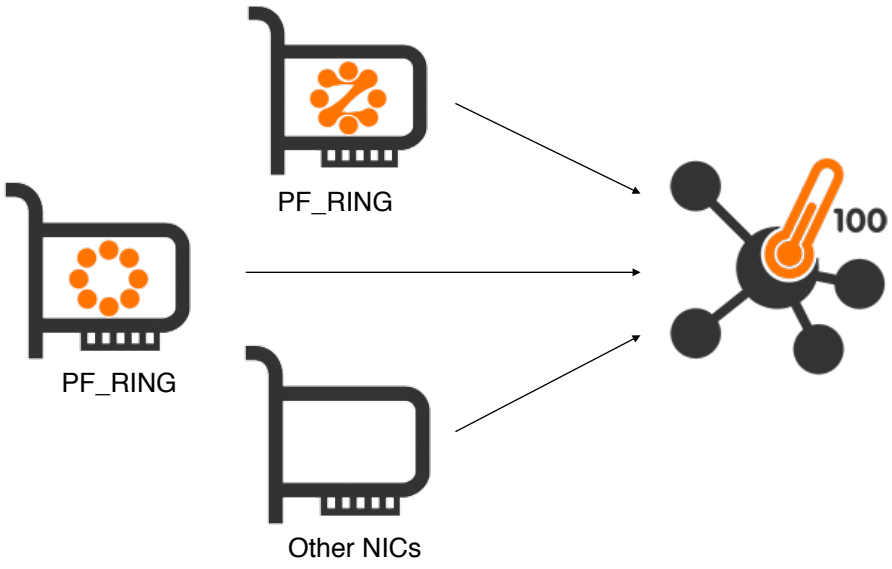
- Quickly convert network packets into flows that can be stored, analysed or exported to custom or third-party software. For example, flows can be sent to ntopng to carry on network-intelligence tasks such as historical investigations of congestions or intrusions. Alternatively, flows can be sent to Apache Kafka for further processing or storage in an Hadoop ecosystem.

- Forward network packets to a traffic recorder such as n2disk, with a substantial control on the forwarding policies. Forwarding policies allow, for example, to exclude encrypted traffic with the aim of alleviating the load on the recorder.

- Seamlessly integrate with intrusion prevention and intrusion detection systems (IDS/IPS) by delivering them network traffic on multiple output queues. Balancing the traffic on multiple queues has the benefit that several IDS/IPS processes/instances can work in parallel on meaningful subsets of traffic.

At the basis of every possible use case, there is the straightforward necessity to capture packets that traverse one or more network interfaces. nProbe™ Cento deeply parallel architectural design, allows to capture packets from multiple input interfaces in parallel.

nProbe™ Cento leverages on the accelerated PF_RING library[1] to capture packets from the network interfaces. Specialised (optional) PF_RING-aware drivers allow to further accelerate packet capture by efficiently copying packets from the driver to PF_RING without passing through the kernel. These drivers support the new PF_RING Zero Copy (ZC) library[2] and can be used either as standard kernel drivers or in zero-copy kernel-bypass mode by adding the prefix `zc:` to the interface name.



The PF_RING library is logically divided into modules to support network interface cards (NICs) manufactured by multiple vendors. To chose the right module, it suffices to pass proper interface names to nProbe™ Cento. Interfaces are passed using the `-i` option and are optionally preceded by a prefix. Examples include, but are not limited to:

- `-i eth1` to use the Vanilla Linux adapter;
- `-i zc:eth1` to use the ZC drivers;

For example, to capture traffic from interface eth0 nProbe™ Cento can be run as

```
./cento -i eth0 [...]
```

Alternatively, to capture traffic from four eth1 ZC queues the following command can be used

```
./cento -i zc:eth1@0 -i zc:eth1@1 -i zc:eth1@2 -i zc:eth1@3 [...]
```

As the number of queues as well as the number of interfaces can grow significantly, a compact notation that uses square brackets is allowed. The compact notation can be used to succinctly indicate interfaces and queues. For example the command above can be rewritten using the compact notation as

```
./cento -i zc:eth1@[0-3] [...]
```

When PF_RING is not available, nProbe™ Cento captures packets using the standard libraries for user-level packet capture.

Now that packet capture and network interfaces have been introduced, the discussion can move to the real-world nProbe™ Cento use cases.

100Gbps Flow Exporter

In this first -- perhaps even the most straightforward -- scenario, nProbe™ Cento operates as a flow exporter. A flow exporter, also typically referred to simply as a probe, captures network packets, aggregates them into flows, and propagates the aggregated flows to one or more collectors.

Flow export formats are manyfold, as manyfold are also the collectors. Common export format are NetFlow in its version 5 and 9, and IPFIX. Flows can also be exported to plain text files as well.

Having network packets reliably aggregated into flows paves the way for a huge number of network management and intelligence tasks, including:

- Export flows to one or more NetFlow v5/v9/IPFIX collectors to:
  - Create a modular, scalable network monitoring infrastructure;
  - Seamlessly integrate with an existing monitoring infrastructure by quickly adding new vantage points.

- Inject flows in an Hadoop[3] cluster through Apache Kafka, to enable further processing for example by mean of the realtime streaming computation systems such as Apache Spark Streaming[4], Storm[5] and Samza[6].

- Propagate flows to the opensource network visualization and analytics tool ntopng to effectively investigate network issues by means of modern interactive charts.

- Dump flows to file for event logging or future investigations.

In the remainder of this subsection we see real-world examples on how to effectively nProbe™ Cento as a flow exporter.

Integration with ntopng

ntopng is the next generation version of the original ntop, a network traffic probe and visualisation software. A wide number of operations can be done using ntopng, operations that range from simple network investigations jobs, to more complex network intelligence tasks. The great flexibility ntopng allows it to operate as an nProbe™ / nProbe™ Cento flow collector by means of a compressed, optionally encrypted, network exchange format built on top of Zero-M-Queue[7] (ZMQ).
This flexibility allows to completely decouple the heavy-duty packet processing tasks carried out by nProbe™ Cento from the visualisation and analytics done via ntopng.

Configuring nProbe™ Cento and ntopng to exchange flow data over ZMQ is basically a two-lines operation. Let's say an nProbe™ Cento has to monitor interface eth1 and has to export generated flows over ZMQ to an ntopng instance for visualisation and storage. The following command suffices to properly start nProbe™ Cento

```
./cento -i eth0 --zmq tcp://192.168.2.130:5556
```

The command above assumes there is an ntopng instance listening for ZMQ flow data on host 192.168.2.130 port 5556. The latter instance can be executed as

```
./ntopng --zmq-collector-mode -i tcp://*:5556
```

As it can be seen from the `-i` option, ntopng treats the remote streams of flows as if they were coming from a physical attached interface. Actually, there will be no differences in the ntopng user interface utilization. This great flexibility makes it possible to create complex and even geographical distributed network monitoring infrastructures. Visualization and storage of monitored flows data can take place on designated hardware, Virtual Machines (VMs), or even at physically different data centers that only need to share a network connection with nProbe™ Cento instances.

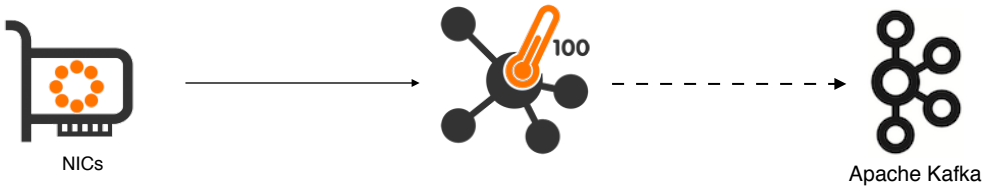
Integration with a NetFlow Collector

Let's say there is the need to add a vantage point to an existing network monitoring infrastructure. A new vantage point can be, for example, the switch at a new office that has just been opened. Let's also say the mirror port of the switch is connected to the nProbe™ Cento host interface eth1.

Assuming the existing infrastructure has a NetFlow v9 collector listening on host 192.168.1.200 port 2055, the following command can be executed to instruct nProbe™ Cento to act as a NetFlow v9 exporter

```
./cento -i zc:eth1 --v9 192.168.1.200:2055
```

Flows Injection in Apache Kafka



Apache Kafka can be used across an organization to collect data from multiple sources and make them available in standard format to multiple consumers, including Hadoop, Apache HBase, and Apache Solr. nProbe™ Cento compatibility with the Kafka messaging system makes it a good candidate source of network data.

nProbe™ Cento inject flow "messages" into a Kafka cluster by sending them to one or more Kafka brokers that are responsible for a given topic. Both the topic and the list of Kafka brokers are submitted using a command line option. Initially, nProbe™ Cento tries to contact one or more user-specified brokers to retrieve Kafka cluster metadata. Metadata include, among other things, the full list of brokers available in the cluster that are responsible for a given topic. nProbe™ Cento will use the retrieved full list of brokers to deliver flow "messages" in a round robin fashion.

nProbe™ Cento also features optional message compression and message acknowledgement policy specification. Acknowledgment policies make it possible to totally

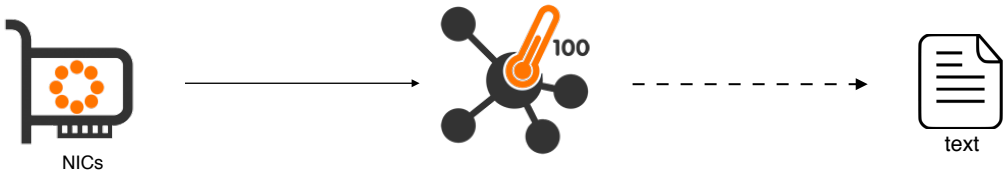
avoid waiting for acknowledgements, to wait only for the Kafka leader acknowledgement, or to wait for an acknowledgment from every replica.

For the sake of example, let's say an nProbe™ Cento has to monitor interface eth1 and has to export generated flows to Kafka topic "topicFlows". The following command can be run

```
./cento -i eth1 \
      --kafka "127.0.0.1:9092,127.0.0.1:9093,127.0.0.1:9094;topicFlows"
```

The command above assumes flows have to be exported on topic topicFlows and that there are three brokers listening on localhost, on ports 9092, 9093 and 9094 respectively. The aim of the command above is to give an example of how straightforward is to put in place an interaction between nProbe™ Cento and Apache Kafka. For a detailed description of the command used, we refer the interested reader to the "Command Line Options" section of the present manual.

Flows Dump to Plain Text Files



Sometimes, simply saving network flows to plain text files may suffice. Plain text files are very simple to manage, are easy to move around, and are very well-suited to be compressed and archived. Therefore, they are still one of the preferred choices, especially in fairly simple environments.

However, as the number of flows exported has the potential to grow unpredictably in time, a clear and straightforward way of dumping them to plain text files is required. nProbe™ Cento organises files in a hierarchical, time-based directory tree:

- The directory tree is time-based in the sense that the time is thought of as divided into fixed, configurable-size slots. Every time slot corresponds to a directory in the file system.

- The directory tree is hierarchical in the sense that flows seen in the same hour will be placed in one or more files contained in a directory which, in turn, is contained in another directory that contains all the hours in a day. The directory that contains all the hours in a day is contained in a directory that contains all the days in a month, and so on. A flow is saved on a given file based on the time it is exported from the probe.

Monitor flows from interface eth1 and dump monitored flows to files under directory `/tmp/flows/` directory is as simple as

```
./cento -i eth1 --dump-path /tmp/flows
```

nProbe™ Cento will create files and directories at runtime, e.g.,

```
/tmp/flows/eth0/2016/05/12/11/1463046368_0.txt
```

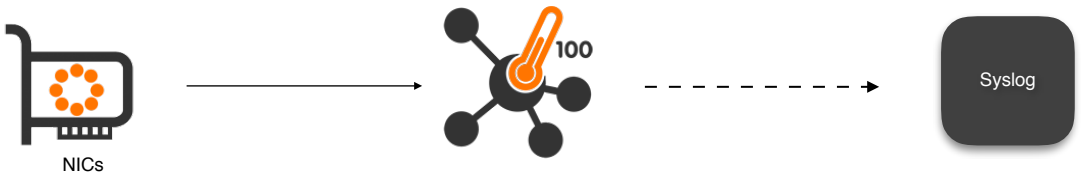
where eth0 is the interface name, 2016 is the year, 05 the month, 12 is the day of the month, 11 the hour of the day (24h format), and 1463046368 is the Unix Epoch that corresponds to the latest packet seen for flows dumped to the file. An optional _0 is required as more that one file can be output.

The text file contains flows, one per line, each one expressed as a pipe-separated list of its details. File contents, artificially limited to the first few entries, are

```
#  FIRST_SWITCHED|LAST_SWITCHED|VLAN_ID|IP_PROTOCOL_VERSION|PROTOCOL|IP_SRC_ADDR|L4_SRC_PORT|IP_DST_ADDR|L4_DST_PORT|
DIRECTION|INPUT_SNMP|OUTPUT_SNMP|SRC_PKTS|SRC_BYTES|DST_PKTS|DST_BYTES|TCP_FLAGS|SRC_TOS|DST_TOS
1463046303.171020|1463046368.234611|0|4|6|192.168.2.222|22|192.168.2.130|62804|R2R|1|2|7|912|12|804|24|0|16
1463046303.562318|1463046303.565914|0|4|6|192.168.2.130|63224|192.168.2.222|443|R2R|1|2|7|944|4|360|27|0|0
1463046303.563135|1463046303.565896|0|4|6|192.168.2.130|63225|192.168.2.222|443|R2R|1|2|7|944|4|360|27|0|0
1463046303.566085|1463046303.950107|0|4|6|192.168.2.130|63226|192.168.2.222|443|R2R|1|2|9|1578|7|1673|27|0|0
1463046303.566253|1463046303.697598|0|4|6|192.168.2.130|63227|192.168.2.222|443|R2R|1|2|9|1575|7|785|27|0|0
1463046304.545697|1463046304.548319|0|4|6|192.168.2.130|63228|192.168.2.222|443|R2R|1|2|7|944|4|360|27|0|0
[...]
```

Several other settings are available and user-configurable and include: the maximum number of flows per file, the maximum time span of every plain text file created, and the maximum time span of every directory created. Settings are discussed in greater detail in section "Comma Line Options".

Flows Dump to Syslog



Most Unix-based systems have a very flexible logging system, which allows to record the most disparate events and then manipulate the logs to mine valuable information on the past events. Unix-based systems typically provide a general-purpose logging facility called syslog. Individual applications that need to have information logged send the information to syslog. nProbe™ Cento is among these applications as it can send flow "events" tot syslog as if they were standard system events.

Monitoring flows from two eth1 ZC queues and dumping them to syslog is as simple as

```
./cento -i zc:eth1@0 -i zc:eth1@1 --syslog
```

An excerpt of the syslog that contains flow "events" is

```
May 12 12:04:28 devel cento[19497]: {"8":"192.168.2.222","12":"192.168.2.130","10":0,"14":0,"2":10,"1":1512,"24":
18,"23":1224,"22":1463047326,"21":1463047468,"7":22,"11":62804,"6":24,"4":6,"5":0}
May 12 12:04:28 devel cento[19497]: {"8":"192.168.2.130","12":"192.168.2.222","10":0,"14":0,"2":7,"1":944,"24":
4,"23":360,"22":1463047327,"21":1463047327,"7":65023,"11":443,"6":27,"4":6,"5":0}
May 12 12:04:28 devel cento[19497]: {"8":"192.168.2.130","12":"192.168.2.222","10":0,"14":0,"2":7,"1":944,"24":
4,"23":360,"22":1463047327,"21":1463047327,"7":65024,"11":443,"6":27,"4":6,"5":0}
May 12 12:04:28 devel cento[19497]: {"8":"192.168.2.130","12":"192.168.2.222","10":0,"14":0,"2":9,"1":1575,"24":
7,"23":785,"22":1463047327,"21":1463047327,"7":65025,"11":443,"6":27,"4":6,"5":0}
```

The format used for the syslog messages description is the actual flow together with its details, e.g.,

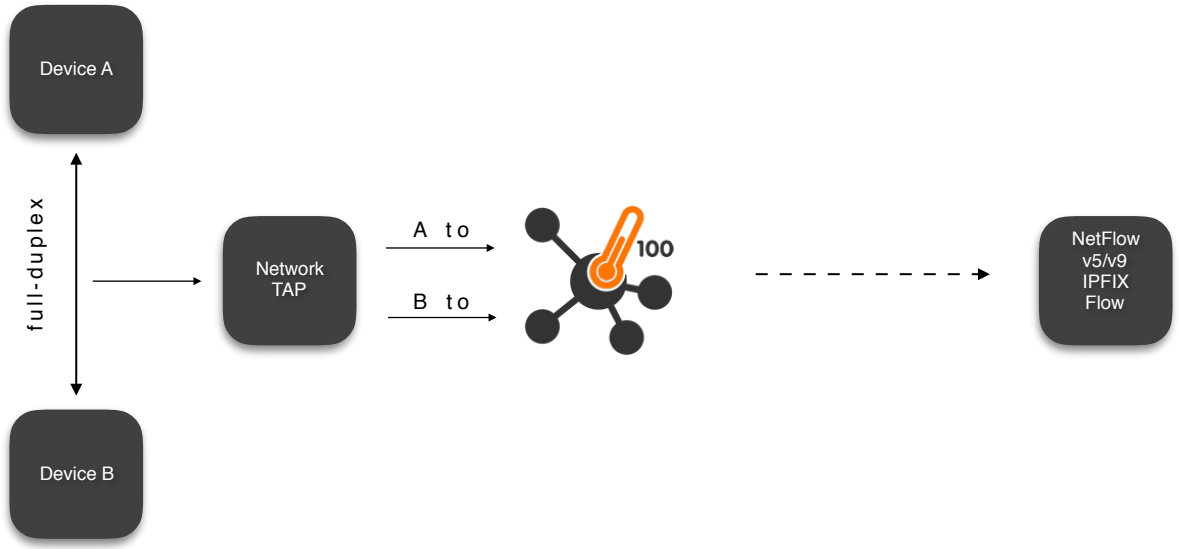
```
{"8":"192.168.2.222","12":"192.168.2.130","10":0,"14":0,"2":10,"1":1512,"24":18,"23":1224,"22":1463047326,"21":
1463047468,"7":22,"11":62804,"6":24,"4":6,"5":0}
```

In order to optimize performances and save space, A JSON format is used. Information is represented in the JSON as a list of pairs "key":value. Keys are integers that follow the NetFlow v9 field type definitions[8]. For example, IN_PKTS correspond to integer 2, whereas OUT_PKTS correspond to integer 24. The mapping between field ID and name, can be found in chapter 8 of RFC 3954 (https://www.ietf.org/rfc/rfc3954.txt).

Full-Duplex TAP Aggregator + 100Gbps Probe

Network administrators may decide to install physical TAP devices between two endpoints such as, for example, switches, hosts, servers or routers. The aim of a TAP is to capture traffic that the endpoints exchange each other. TAPs are alternative solutions to the integrated port mirroring functions available on many switches and routers. A TAP may be preferred over port mirroring as it does not consumes switch or router processing resources by keeping it busy with the generation of mirrored traffic.

Very often the traffic captured by the TAP flows over full-duplex technologies such as Ethernet. Full-duplex means that packets can travel simultaneously in both directions, e,.g., from device A to device B and from device B to device A. It follows that the aggregate throughput of a full-duplex technology is twice the nominal speed. For example, a 1Gbps network link actually allows 2Gb of aggregated throughput in every second. This raises a problem as a single-port monitoring device that receives traffic from a TAP may not be able to process all the packets received in the unit of time. For this reason, TAPs for full-duplex technologies usually have two exit ports, one for each half of the connection.



nProbe™ Cento has the ability to to merge the two directions of traffic it receives into one aggregate stream to see both directions of the traffic, namely A to B and B to A. The resulting aggregated traffic is complete and comprehensive by accounting for both the halves of the full-duplex link. Aggregating the two directions is necessary and fundamental, for example, to accurately generate flow data and to identify Layer-7 protocols.

TAP-Aggregated Flows Export to a Netflow Collector

Let's say nProbe™ Cento has to be used as a probe to export flow data obtained from traffic received from a full-duplex TAP. Let's also say the full-duplex TAP has its two exit ports connected to interfaces `eth1` and `eth2` respectively. Assuming there is NetFlow v9 collector listening on host 192.168.1.200 port 2055 that is in charge of collecting flows data gathered from the TAP, the following command can be executed

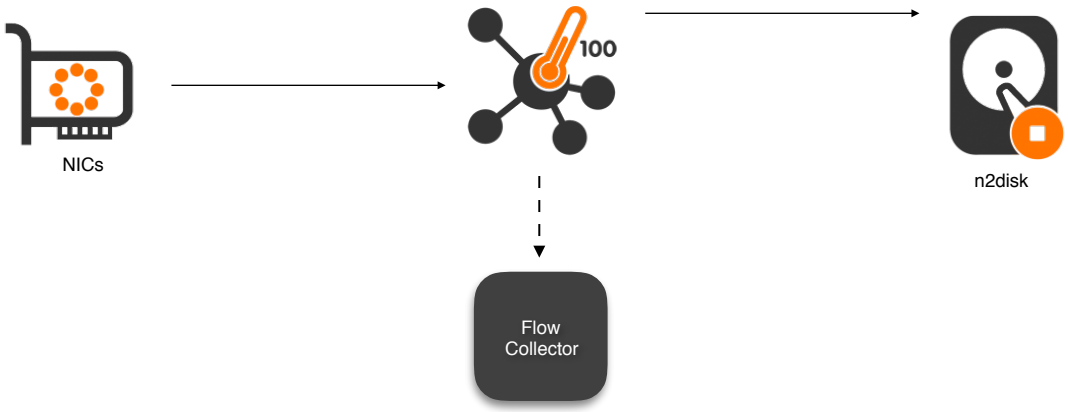
```
./cento -i eth1,eth2 --v9 192.168.1.200:2055
```

100Gbps Probe + Traffic Aggregator

In this scenario, nProbe™ Cento operates not only as a Flow Exporter, but also as a traffic aggregator. That is, it forwards, to a single egress queue, the input traffic that comes from one or more input interfaces. Therefore, as a traffic aggregator, nProbe™ Cento has the additional duty to reliably capture input packets and to properly forward such packets to a single output queue.

There are several real-world situations in which it is needed to aggregate multi-source traffic into a aggregate stream. Among those situations, it is worth mentioning the traffic recording, also known as packet-to-disk. The analysis of recorded network traffic is fundamental, for example, in certain types of audits and data forensics investigations. Indeed, any packet that crosses the network could potentially carry a malicious executable or could leak sensible corporate information towards the Internet. Being able to understand and reconstruct prior situations is of substantial help in preventing outages and infections or in circumstantiating leaks.

Packet-to-Disk Recording



Among all the commercial traffic recorders available, it is worth discussing n2disk[9] as nProbe™ Cento integrates straightforwardly with it. n2disk™ captures full-sized network packets at multi-Gigabit rate (above 10 Gigabit/s on proper hardware) from live network interfaces, and write them to files without any packet loss.

n2disk™ allows to specify a maximum number of distinct files that will be written during the execution, with the aim of enabling an a priori provisioning of disks. When n2disk™ reaches the maximum number of files, it will start recycling the files from the oldest one. In this way it is possible have a complete view of the traffic for a fixed temporal window and, at the same time, know in advance the amount of disk space needed.

Both nProbe™ Cento and n2disk™ have been designed to fully exploit the resources provided by multi-core systems, aggregating traffic that comes from multiple, independent input queues in the former, scattering packet processing on multiple cores in the latter.

Let's say nProbe™ Cento has to be used as a flow exporter for a v9 NetFlow collector listening on 192.168.2.221 port 2055 and, at the same time, has to aggregate monitored traffic received from four different eth1 ZC queues into a single aggregated egress queue. Then the following command can be issued

```
./cento-ids -i zc:eth1@[0-3] --aggregated-egress-queue \
--egress-conf egress.example --dpi-level 2 \
--v9 192.168.2.221:2055 \
--aggregated-egress-queue
```

Upon successful startup, nProbe™ Cento will output the ZC egress queue identifier that needs to be passed to n2disk™. As discussed in greater details later in this manual, egress queues identifiers have the format `zc:<cluster id>:<queue id>`. Assuming nProbe™ Cento assigned identifier `zc:10@0` to the `aggregated-egress-queue`, it is possible to record egress traffic via n2disk using the following command

```
./n2disk -i zc:10@0 --dump-directory /storage \
      --max-file-len 1024 --buffer-len 2048 --chunk-len 4096 \
      --reader-cpu-affinity 4 --writer-cpu-affinity 5 \
      --index --index-version 2 --timeline-dir /storage
```

The meaning of the options passed in the command above is:

- `-i zc:10@0` is the n2disk™ ingress queue , that is, the egress queue of nProbe™ Cento
- `--dump-directory /tmp/recorded` is the directory where dump files will be saved
- `--max-file-len 1024` is the maximum pcap file length in Megabytes
- `--buffer-len 2048` is the buffer length in Megabytes
- `--chunk-len 4096` is the size of the chunk written to disk in Kilobytes
- `--reader-cpu-affinity 4` binds the reader thread to core 4
- `--writer-cpu-affinity 5` binds the writer thread to core 5
- `--index` enables on-the-fly indexing
- `--index-version 2` enables flow-based indexing
- `--timeline` arranges recorded pcap files on a time basis

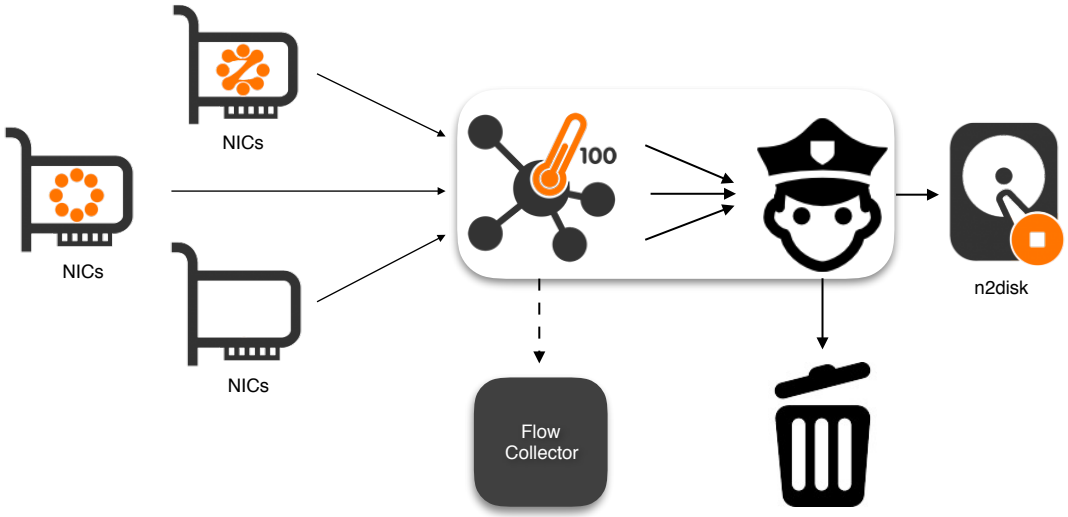
Reader and writer CPU affinities are not strictly necessary but their use is encouraged to avoid interferences. During the execution, new pcaps will start popping out under /tmp/recorded thanks to n2disk™. After a few seconds of execution this may be the result

```
ls -a /storage/*
/storage/1.pcap  /storage/2.pcap  /storage/3.pcap
```

n2disk™ has been designed to fully exploit multiple cores. The minimum number of internal threads is 2, one for packet capture (reader) and one for disk writing (writer), but it is also

possible to further parallelize packet capture and indexing using more threads. In order to achieve the best performance with n2disk™, it is also possible to parallelize packet capture using multiple reader threads by means of the new ZC-based n2disk that is bundled in the n2disk™ package. Please refer to the n2disk manual further n2disk configuration options.

Policed Packet-To-Disk Recording



Aggregated, multi-source egress traffic queues must be flexible, in the sense that the user should have a fine-grained control over the traffic that gets actually forwarded to them. Flexibility is fundamental for example to avoid forwarding encrypted traffic (e.g., SSL, HTTPS) to traffic recorders or, oppositely, to forward only the traffic that comes from a suspicious range of addresses.

Fine-grained control is enforced in nProbe™ Cento through a hierarchical set of rules. Rules are specified using a configuration file that follows the INI standard[10]. Rules can be applied at three different levels, namely, at the level of aggregated queue, at the level of subnet, and at the level of application protocol. Application protocol rules take precedence over subnet-level rules which, in turn, take precedence over the queue-level rules.

Rule types are five, namely: forward, discard, shunt, slice-l4 and slice-l3. Forward and discard are self-explanatory, shunt means that only the first K=10 packets of every flow are forwarded, and slice-l4 (slice-l3) mean that the packet is forwarded only up to the layer-4 (layer-3) headers included.

For example, let's say the input traffic that is coming from ZC interface eth1 has to be forwarded to a single aggregated output queue, with the extra conditions that:

- By default, at most K=10 packets are forwarded for every flow;
- Traffic that originates from or goes to subnets 10.0.1.0/24 and 10.10.10.0/24 has to be forwarded without any restriction on the maximum number of packets per flow.
- HTTP traffic has always to be forwarded regardless of the source/destination subnet
- SSL and SSH traffic must always be sliced to make sure encrypted layer-4 payloads are never forwarded to the output queue.

Having in mind the conditions above, it is possible to create the plain-text configuration file below

```
[shunt]
default = 10

[egress.aggregated]
default = shunt

[egress.aggregated.subnet]
10.0.1.0/24 = forward
10.10.10/24 = forward

[egress.aggregated.protocol]
HTTP = forward
SSL = slice-l4
SSH = slice-l4
```

For a thorough description of the configuration file format we refer the interested reader to the section "Egress Queues".

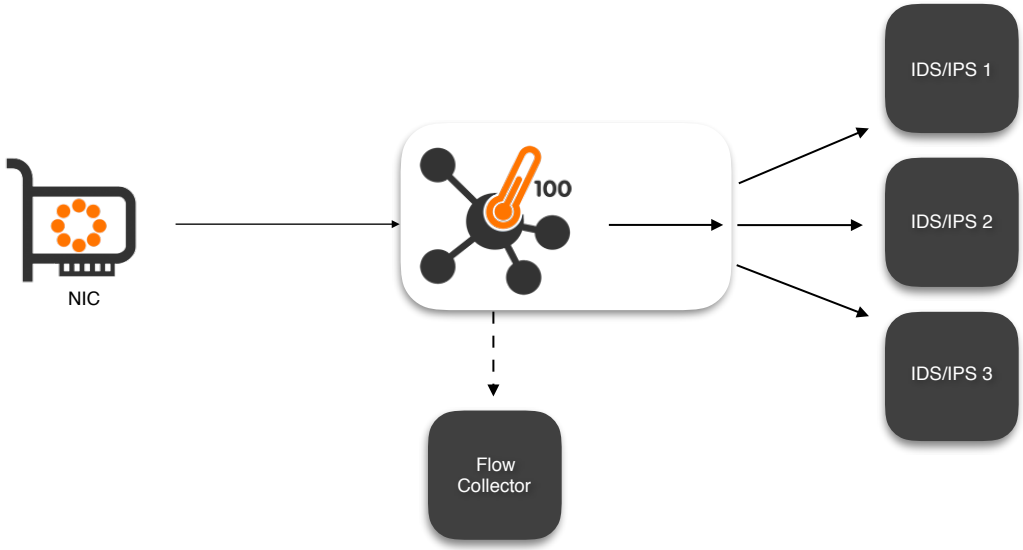
Assuming the configuration file above is named `egress.conf` and is located in the current working directory, then, in order to enforce the policies described above, nProbe™ Cento can be started as follow

```
./cento-ids -i zc:eth1 --aggregated-egress-queue \
      --egress-conf egress.conf --dpi-level 2
```

At this point it one can use n2disk™ on policier traffic in the same way as it has been described in the section above.

100Gbps Probe + Traffic Balancer for IDS / IPS

nProbe™ Cento has the ability to balance the input traffic that is coming from each input interface toward two or more "balanced" output queues. This feature is particularly useful when the traffic has to be forwarded to Intrusion Detection / Intrusion Prevention Systems (IDS/IPS). Leveraging on the nProbe™ Cento traffic balancer it is possible to run multiple IDS/IPS instances in parallel, each one on subset of the ingress traffic. This enables a better utilisation of multi-core elaboration systems.



nProbe™ Cento balancing is done on a per-flow basis using two-tuple hashing (IP src and IP dst) to guarantee coherency also for fragmented packets (not guaranteed by 5-tuple hashing). Therefore, it is guaranteed that the same flow will always be forwarded to the same balanced output queue. Fine-grained control over the balanced egress queues is available through a set of hierarchical rules. As already discussed in the previous section, rules can be applied at three different levels, namely: at the level of egress queue, at the level of subnet, and at the level of application protocol.

Rules are hierarchical in the sense that application-protocol rules take precedence over the subnet-level rules which, in turn, take precedence over the queue-level rules. The list of rules is specified via a text file in the INI standard.

Rule types are five also for the balanced egress queues, namely: forward, discard, shunt, slice-l4 and slice-l3. Forward and discard are self-explanatory, shunt means that only the first K=10 packets of every flow are forwarded, and slice-l4 (slice-l3) mean that the packet is forwarded only up to the layer-4 (layer-3) headers.

Integration with Suricata[11] IDS/IPS

Let's say one want to balance ZC eth1 queue (`@0` can be omitted) across two balanced output queues. Let's also feed Suricata IDS/IPS with the two balanced queues and, simultaneously, export flows to a NetFlow v9 collector listening on localhost, port 1234.

nProbe™ Cento can be run as

```
./cento-ids -i zc:eth1 --v9 127.0.0.1:1234 --balanced-egress-queues 2
```

Balanced queues ids are output right after the beginning of the execution. Assuming nProbe™ Cento assigned identifiers `zc:10@0` and `zc:10@1` to the queues, then it is possible to start Suricata IDS/IPS as

```
./suricata --pfring-int=zc:10@0 --pfring-int=zc:10@1 \
      -c /etc/suricata/suricata.yaml --runmode=workers
```

Integration with Snort[12] IDS/IPS

Let's say one wants to balance one ZC eth1 queue (`@0` can be omitted) across two balanced output queues and, at the same time, attach a Snort IDS/IPS instance to each queue. Assuming that nProbe™ Cento has also to export flows to a NetFlow v9 collector listening on localhost port 1234 , one can run the following command

```
./cento-ids -i zc:eth1 --v9 127.0.0.1:1234 --balanced-egress-queues 2 \
      --egress-conf egress.conf --dpi-level 2
```

Assuming nProbe™ Cento has given identifiers `zc:10@0` and `zc:10@1` to the queues, then it is possible to start two Snort IDS/IPS instances as

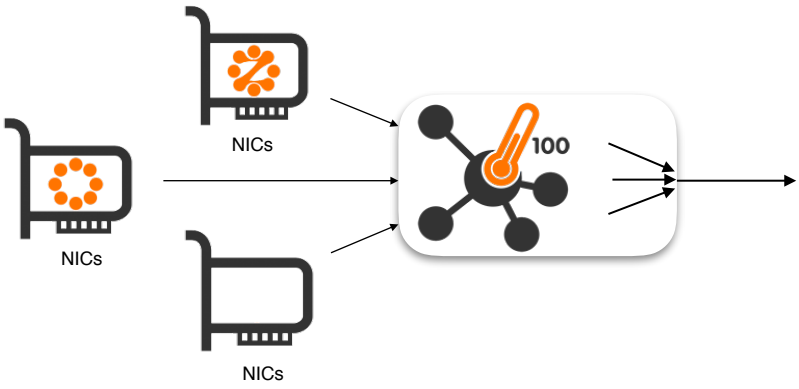
```
./snort --daq-dir=/usr/local/lib/daq --daq pfring_zc --daq-mode passive \
      -i zc:10@0 -v -e

./snort --daq-dir=/usr/local/lib/daq --daq pfring_zc --daq-mode passive \
      -i zc:10@1 -v -e
```

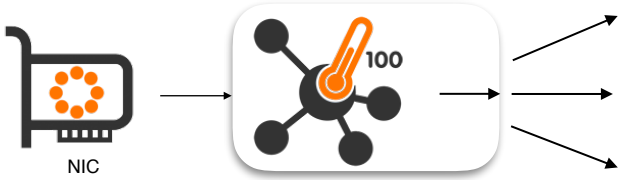
Egress Queues

As it has already been introduced in section “Use Cases”, nProbe™ Cento implements traffic forwarding functionalities that allow to:

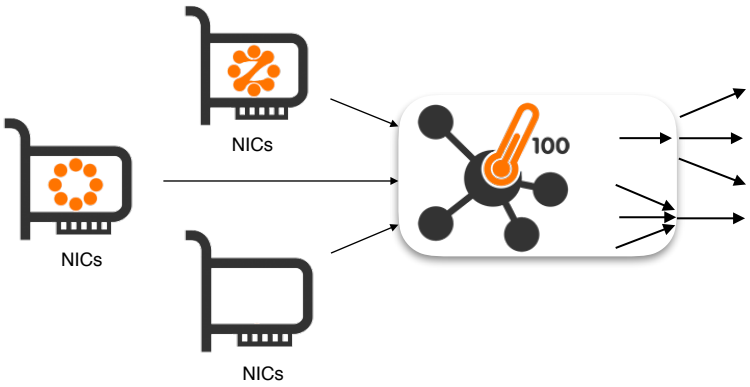
- Capture packets from multiple input interfaces and forward them towards a single, aggregated egress queue.



- Capture packets from an input interface and forward them towards multiple, balanced egress queues.



- Capture packets from one or more input interfaces and simultaneously forward them to multiple balanced queues and to a single aggregated queue.



An egress queue is said to be aggregated when it carries packets coming from multiple input interfaces. Conversely, it is said to be balanced when it carry a subset of packets originated at the input interface.

Often, a bare packet forwarder doesn't suffice as one may need some flexibility to policy the traffic that goes through the output queues. For example, if the egress queue used by a packet recorder, one may save valuable space by avoid recording encrypted traffic payloads. Similarly, when the balanced egress queues are sent to IDS/IPS, one may reduce the load on the processors by dropping encrypted packets.

Policing Egress Queues Traffic

Both aggregated and balanced egress queues allow a fine-grained traffic control through a set of hierarchical rules. Rules are submitted statically using a text file during nProbe™ Cento startup or dynamically via a REST API. Rules can be applied at three different levels, namely:

- At the level of queue;
- At the level of subnet;
- At the level of application protocol.

Application protocol rules take precedence over subnet-level rules which, in turn, take precedence over the queue-level rules. Queue-level rules can be thought of as the "default" queue rules.

Policy Rules

As already introduced above, policies can be enforced at three different levels by means of rules. Rule types are five, namely:

- *Forward*: forward the packet;
- *Discard*: do not forward the packet;
- *Shunt*: forward the first K packets in every flow, where K is a configurable constant;
- *Slice-l4*: cut the packet after the IP layer;
- *Slice-l3*: cut the packet after the TCP/UDP layer.

Forward means the traffic should be delivered to the output queue as-is, without being dropped or modified.
Discard means that the traffic doesn't get forwarded to the output queue.
Shunt means that only the first K packets of every flow are forwarded to the output queue. The value of K is configurable and can be set by the user.
Slice-l4 (slice-l3) mean that the packet is forwarded only up to the layer-4 (layer-3) headers. Any layer-4 (layer-3) payload is discarded.

Rules are expressed as <policed unit> = <policy>. For example, one that wants to shunt all the egress traffic will use the rule

```
default = shunt
```

Assuming an exception is needed to waive the default rule in order to record all the traffic that originates from (is destined to) subnet 10.0.0.0/8, one will use the additional rule

```
10.0.0.0/8 = forward
```

Supposing another exception is needed to waive both the default rule and the rule set on the subnet in order to always drop SSH traffic, one will add a third rule as

```
SSH = discard
```

Please note that the list of DPI protocols supported by cento can be printed with:

```
./cento --print-ndpi-protocols
```

The Egress Queues Configuration File

Rules are specified in a plain text file that follows the INI standard. INI is a very simple standard that specify the format of the configuration file. The configuration file is divided in sections. Every section starts with a "label" that is wrapped between square brackets, e.g., `[egress.aggregated]`. Every line contained after the label is related to the section. A section ends implicitly when a new section begins.

nProbe™ Cento configuration files contain, between square brackets, sections corresponding to the different hierarchical levels for both aggregated and balanced egress queues. Each section contains the rules that have to be applied. An additional section is present to configure shunting. An exhaustive example of configuration file is the following

```
[egress.aggregated]
default = shunt
banned-hosts = discard

[egress.aggregated.subnet]
10.0.0.1/32 = forward
10.10.10/24 = forward

[egress.aggregated.protocol]
HTTP = forward
SSL = slice-l4
SSH = slice-l4

[egress.balanced]
default = forward
banned-hosts = discard

[egress.balanced.subnet]
192.168.2/24 = discard

[egress.balanced.protocol]
SSL = discard
SSH = discard

[shunt]
default = 10
tcp = 12
udp = 2
HTTP = 40
SSL = 5
```

In the remainder of this section is given a description of configuration file sections.

Shunting

The section `[shunt]` of the configuration file allows to specify the number of the number of packets per flow that have to go through any egress queue. This section is global and applies both to aggregated and balanced queues. The default number of packets is set using the rule

```
default = 10
```

The rule above sets to 10 the maximum number of packets per flow that are forwarded to the egress queues. Caps can also be set for the maximum number of packets that have to go

though any tcp (e.g., `tcp = 12`) or udp (e.g., `udp = 2`) flow. Similarly, the maximum number of packets can be configured on a per-application-protocol basis. For example, to set to 40 the maximum number of packets allowed to go through ay HTTP flow, one can use the following line

```
HTTP = 40
```

The rule of last resort is `default`, that is, any flow that doesn't match any of the other rules is shunted using the `default` value. Application-protocol rules "override" layer-4 rules, that is, an HTTP flow transported over tcp is forwarded up to the 40th packet if one considers the examples above.

All reserved keywords (in addition to nDPI protocols) are:
- default
- tcp
- udp

Aggregated Egress Queue

Aggregated egress queues are configured via three sections, namely `[egress.aggregated]`, `[egress.aggregated.subnet]` and `[egress.aggregated.protocol]`. Rules are indicated in every section, one per line.

`[egress.aggregated.protocol]`
Section contains application-protocol forwarding rules that must be enforced. Any application protocol is expressed using a string. The full list of application protocols available can be queries simply by running nProbe™ Cento inline help

```
./cento --print-ndpi-protocols
```

Any line in this section has the format `<protocol name> = <policy>`, where policy can be any of: `forward`, `discard`, `shunt`, `slice-l4` and `slice-l3`.

An example is

```
[egress.aggregated.protocol]
HTTP = forward
SSL = slice-l4
SSH = slice-l4
```

`[egress.aggregated.subnet]`
Section contains forwarding rules that are enforced at the level of subnet. Every line in this section is formatted as `<CIDR subnet> = <policy>`. Subnets are expressed using the common CIDR `<address>/<netmask>` notation. Policy can be any of: `forward`, `discard`, `shunt`, `slice-l4` and `slice-l3`.

An example is

```
[egress.aggregated.subnet]
10.0.0.1/32 = forward
10.10.10/24 = forward
```

If a flow matches both a subnet- and a protocol-level rule, then the protocol-level rule is applied.

`[egress.aggregated]`
Section contains the "default" egress queue policies, that is, any flow that doesn't match any rule indicated in the subnet and protocol sections, then will be policed using the "default" rules. This section contains at most two lines, one for the default policy and the other for the policy that has to be applied to banned hosts. We refer the reader to the section "Command Line Options" for a detailed description of banned hosts.

An example is

```
[egress.aggregated]
default = shunt
banned-hosts = discard
```

In the example above, all the flows that are not policed via subnet- or protocol-level rules are `shunt`ed as specified in the `default` rule. If an host belongs to the list of `banned-hosts`, then all the traffic gets `discard`ed.

Balanced Egress Queues

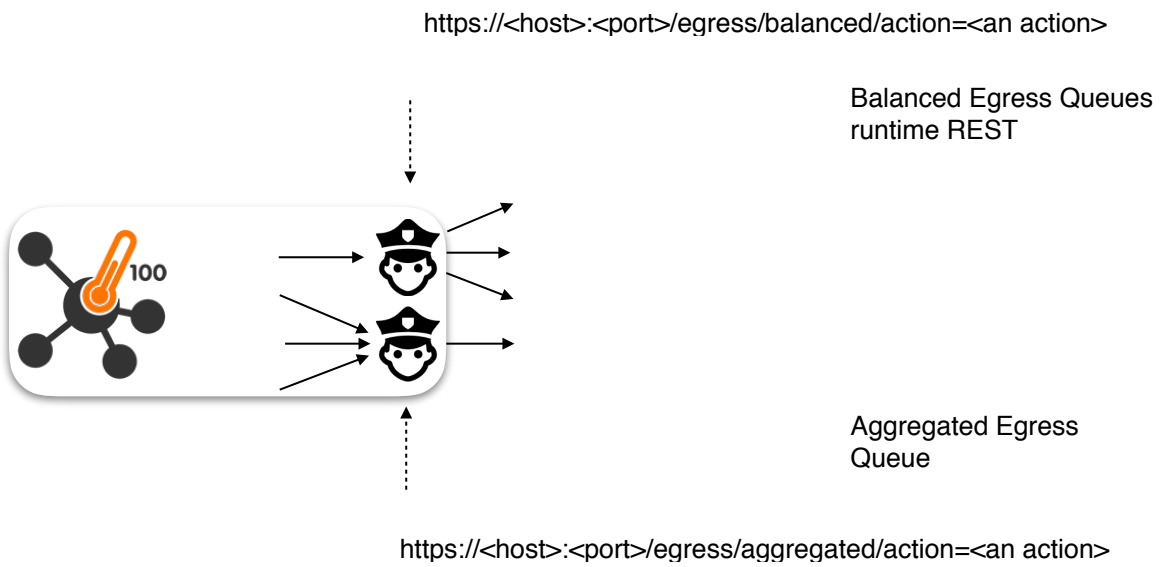
Balanced egress queues are configured exactly as it is described in the section above for aggregated egress queues. The only difference lays in the three section names that are:

- `[egress.balanced]`
- `[egress.balanced.subnet]`
- `[egress.balanced.protocol]`

Therefore, we refer the reader to the previous section for a detailed description of queues configuration.

The Egress Queues runtime REST Configuration API

nProbe™ Cento has been designed to allow egress queues to be dynamically configured by means of a REST API. One of the main reasons behind the choice of exposing queues configuration though an API is to provide a feedback channel that can be used, for example, by IDS/IPS such as Suricata od Snort. Indeed, it is fundamental to provide IDS/IPS with a quick way to policy the traffic on the basis of their runtime decisions. Thanks to this API, an IDS/IPS becomes able, for example, to block an host that has just been flagged as malicious. Similarly, it can use the API to instruct nProbe™ Cento to start sending the traffic of a suspicious subnet to a traffic recorder.



The REST API can be accessed via HTTP (on port 8880) or via HTTPS (on port 4443). This API that provides the same level of flexibility that can be achieved through the configuration file. Indeed, the API can be used to set policy rules at any of the three different levels thoroughly discussed above, namely, at: the queue-, the subnet- and the protocol-level. We refer the interested reader to the "Policy Rules" section above for a detailed description of these rules.

Identifying the base REST Endpoint

The base REST endpoint to contact when configuring egress queues has to be built according to the following format

```
http[s]://<host>:<port>/egress/{aggregated,balanced}/
```

One may switch between HTTP and HTTPS just by adding or removing the optional trailing `s` after `http`. The REST server will listen at address `<host>`, on a given `<port>`. Both parameters are user-configurable.

The user will specify `aggregated/` or `balanced/` after the `/egress/` part of the url depending on what is being configured.

Configuring Queue-Level Rules

Once the base endpoint has been built, one has to add the keyword `default` -- which is meant to indicate the default queue-level policies -- followed by a ?-separated `action` that can be any of: `forward`, `discard`, `shunt`, `slice-l4` and `slice-l3`.

For example, a default slice-l4 policy can be set to policy the queue-level of the aggregated egress queues as follow

```
http[s]://<host>:<port>/egress/aggregated/default?action=slice-l4
```

Similarly, a default shunt policy for balanced egress queues can be set as

```
http[s]://<host>:<port>/egress/balanced/default?action=shunt
```

Configuring Subnet-Level Rules

Subnet-level must specify the keyword `ip`, followed by a `?subnet=<address/mask>` needed to indicate, in CIDR, the subnet that is being policed. An extra `&action` is required to indicate the actual policy that has to be enforced. Action can be any of: `forward`, `discard`, `shunt`, `slice-l4` and `slice-l3`.

For example, the following request can be issued to enforce, on balanced egress queues, a discard policy for subnet 192.168.2.0/24

```
http[s]://<host>:<port>/egress/balanced/ip?subnet=192.168.2.0/24&action=discard
```

Alternatively, one may set a forward policy for subnet 192.168.3.0/24 on aggregated egress queues as

```
http[s]://<host>:<port>/egress/aggregated/ip?subnet=192.168.3.0/24&action=forward
```

Configuring Protocol-Level Rules

Protocol-level rules must specify the keyword `protocol`, followed by the actual protocol string that has to be policed. An additional `?action` is required to indicate the actual policy that has to be adopted. Action can be any of: `forward`, `discard`, `shunt`, `slice-l4` and `slice-l3`.

For example, to enforce, on balanced egress queues, a discard policy for protocol SSH, one may perform the following request

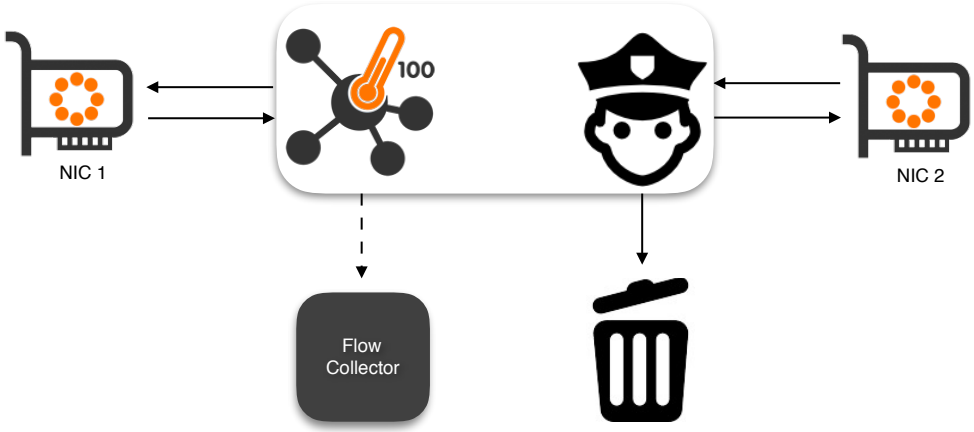
```
http[s]://<host>:<port>/egress/balanced/protocol/SSH?action=discard
```

Alternatively, one may set a forward policy for protocol HTTP on aggregated egress queue as

```
http[s]://<host>:<port>/egress/aggregated/protocol/HTTP?action=forward
```

Network Bridge

nProbe™ Cento features a bi-directional bridge mode. When functioning in bridge mode, nProbe™ Cento intercepts, policy, and (possibly) forward any traffic that goes through a pair of network interfaces. Flow export functionalities will continue to work smoothly and independently from the bridge.



nProbe™ Cento allow a fine-grained control of the bridged traffic through a set of hierarchical rules. Rules are submitted statically using a text file during nProbe™ Cento startup or dynamically via a REST API. The set of rules available to policy the bridged traffic is a subset of the one used to policy egress queues. Indeed, in bridge mode, slicing and shunting will be of no help.

A detailed description of network bridge configuration and policing is given in the remainder of this section. The careful reader will notice a certain degree of overlap with configuration and policing of egress queues. Indeed, in an effort to keep nProbe™ Cento as usable as possible, the developers have designed a clear workflow that can be almost seamlessly used both in the context of packet bridging and in the one of egress queues.

Policing Bridged Traffic

Bridged traffic is policed though a fine-grained set of hierarchical rules. Rules can be applied at three different levels, namely:

- At the level of bridge;
- At the level of subnet;
- At the level of application protocol.

Application protocol rules take precedence over subnet-level rules which, in turn, take precedence over the bridge-level rules. Bridge-level rules can be thought of as the "default" bridge rules.

Policy Rules

As already discussed above, policies can be enforced at three different levels by means of rules. Rule types are two, namely:

- *Forward*: forward the packet;
- *Discard*: do not forward the packet;

Forward means the traffic received from a bridged network interface card (NIC) will be forwarded to the other bridged NIC as-is, without being dropped, altered, or modified.
Discard means that the traffic received from a bridged NIC will not get forwarded to the other bridged NIC.

Rules are expressed as `<policed unit> = <policy>`. For example, one that wants to bridge all the NICs traffic will use the following expression

```
default = forward
```

Assuming an exception is needed to waive the default rule in order to drop all the traffic that originates from (is destined to) subnet 10.0.0.0/8, one will use the additional rule

```
10.0.0.0/8 = discard
```

Supposing another exception is needed to waive both the default rule and the rule set on the subnet in order to always drop Viber traffic, one will add a third rule as

```
Viber = discard
```

The full list of DPI protocols supported by can be printed with

```
./cento --print-ndpi-protocols
```

The Network Bridge Configuration File

Rules are specified in a plain text file that follows the INI standard. INI is a very simple standard that specify the format of the configuration file. For a more detailed introduction on the INI standard, the interested reader is referred to the section "The Egress Queues Configuration File" that follows the same standard.

nProbe™ Cento network bridge configuration files contain, between square brackets, sections corresponding to the different hierarchical levels of rules. Each section contains the rules, one per line, that have to be applied. An exhaustive example of configuration file is the following

```
[bridge]
default = forward
banned-hosts = discard

[bridge.subnet]
192.168.2/24 = discard

[bridge.protocol]
Skype = discard
```

In the remainder of this section is given a description of configuration file sections. As already anticipated with the configuration file example, network bridging is configured via three sections, namely `[bridge]`, `[bridge.subnet]` and `[bridge.protocol]`. Rules are indicated in every section, one per line.

`[bridge.protocol]`
Section contains application-protocol forwarding rules that must be enforced. Any application protocol is expressed using a mnemonic string. The full list of application protocols available can be queries simply by running nProbe™ Cento inline help

```
./cento --print-ndpi-protocols
```

Any line in this section has the format `<protocol name> = <policy>`, where policy can be any of: `forward` and `discard`.

An example is

```
[bridge.protocol]
Skype = discard
Viber = discard
```

`[bridge.subnet]`
Section contains forwarding rules that are enforced at the level of subnet. Every line in this section is formatted as `<CIDR subnet> = <policy>`. Subnets are expressed using the

common CIDR <address>/<netmask> notation. Policy can be any of: forward and discard.

An example is

```
[bridge.subnet]
10.0.0.1/32 = discard
10.10.10/24 = discard
```

If a flow matches both a subnet- and a protocol-level rule, then the protocol-level rule is applied.

[bridge]
Section contains the "default" bridge policies, that is, any flow that doesn't match any rule indicated in the subnet and protocol sections, then will be policed using the "default" rules. This section contains at most two lines, one for the default policy and the other for the policy that has to be applied to banned hosts. We refer the reader to the section "Command Line Options" for a detailed description of banned hosts.

An example is

```
[bridge]
default = forward
banned-hosts = discard
```

In the example above, all the flows that are not policed via subnet- or protocol-level rules are forwarded as specified in the default rule. If an host belongs to the list of banned-hosts, then all the traffic gets discarded.

Network Bridge Example

Assuming nProbe™ Cento has to be used to bridge ZC eth1 and eth2 interfaces using configuration file bridge.example and banned hosts list file banned.example, it is possible to use the following command

```
./cento-bridge -i zc:eth1,zc:eth2 --bridge-conf bridge.example --banned-hosts banned.example --dpi-level 2 -v 4
```

The Network Bridge Runtime REST Configuration API

nProbe™ Cento has been designed to allow the network bridge to be dynamically configured by means of a REST API. The REST API can be accessed via HTTP (on port 8880) or via HTTPS (on port 4443). This API that provides the same level of flexibility that can be achieved through the configuration file.

Identifying the base REST Endpoint

The base REST endpoint to contact when configuring the network bridge has to be built according to the following format

```
http[s]://<host>:<port>/egress/bridge/
```

One may switch between HTTP and HTTPS just by adding or removing the optional trailing `s` after `http`. The REST server will listen at address `<host>`, on a given `<port>`. Both parameters are user-configurable.

Configuring Bridge-Level Rules

Once the base endpoint has been built, one has to add the keyword `default` -- which is meant to indicate the default queue-level policies -- followed by a ?-separated `action` that can be either `forward` or `discard`.

For example, a default forward policy can be set to policy the bridge-level as follow

```
http[s]://<host>:<port>/egress/bridge/default?action=forward
```

Configuring Subnet-Level Rules

Subnet-level must specify the keyword `ip`, followed by a `?subnet=<address/mask>` needed to indicate, in CIDR, the subnet that is being policed. An extra `&action` is required to indicate the actual policy that has to be enforced. Action can be either `forward` or `discard`.

For example, the following request can be issued to enforce a discard policy for subnet 192.168.2.0/24 on the bridge

```
http[s]://<host>:<port>/egress/bridge/ip?subnet=192.168.2.0/24&action=discard
```

Alternatively, one may set a forward policy for subnet 192.168.3.0/24 as

```
http[s]://<host>:<port>/egress/bridge/ip?subnet=192.168.3.0/24&action=forward
```

Configuring Protocol-Level Rules

Protocol-level rules must specify the keyword `protocol`, followed by the actual protocol string that has to be policed. An additional `?action` is required to indicate the actual policy that has to be adopted. Action can be any of: `forward`, `discard`, `shunt`, `slice-l4` and `slice-l3`.

For example, to enforce a discard policy for protocol Skype, one may perform the following request

```
http[s]://<host>:<port>/egress/bridge/protocol/Skype?action=discard
```

Alternatively, one may set a forward policy for protocol Viber as

```
http[s]://<host>:<port>/egress/bridge/protocol/Viber?action=forward
```

Command Line Options

nProbe™ Cento command line options are briefly summarized below. The same summary shown below can obtained by running nProbe™ Cento with argument --help. Non-trivial options are then logically re-arranged and discussed in greater detail after the brief summary.

Interfaces

[--interface|-i] <ifname|pcap file>

nProbe™ Cento handles network adapters both with vanilla or PF_RING drivers. An interface can be specified as "`-i eth0`" (in-kernel processing with vanilla or PF_RING-aware drivers) or as "`-i zc:eth0`" (kernel bypass). The prefix `zc:` tells PF_RING to use the Zero Copy (ZC) module.

nProbe™ Cento can also be fed with multiple Receive Side Scaling (RSS) queues available from one or more interfaces. Queues are specified using the `@` sign between the interface name and the queue number. For example, the following command is to run nProbe™ Cento on 4 ZC queues of interface eth1

```
./cento -i zc:eth1@0 -i zc:eth1@1 -i zc:eth1@2 -i zc:eth1@3
```

The same 4 queues can be processed in-kernel by omitting the `zc:` prefix as follows

```
./cento -i eth1@0 -i eth1@1 -i eth1@2 -i eth1@3
```

nProbe™ Cento, for every distinct interface specified using `-i`, starts two parallel threads, namely a packet processor thread and a flow exporter thread. Additional threads may be spawned as discussed below.

The number of interfaces, as well as the number of queues, that have to be processed with nProbe™ Cento can grow significantly. For this reason a compact, square-brackets notation is allowed to briefly indicate ranges of interfaces (of queues). For example, using the compact notation, nProbe™ Cento can be started on the 4 ZC eth1 discussed above queues as

```
./cento -i zc:eth1@[0-3]
```

Similarly, one can start the software on four separate ZC interfaces as

```
./cento -i zc:eth[0-3]
```

Egress queues

[--balanced-egress-queues|-o] <num>

Using this option nProbe™ Cento creates, for each interface submitted with `-i`, a number `<num>` of egress queues. Egress traffic is balanced across the egress queues created using the 5-tuple. Therefore it is guaranteed that all the packets belonging to a given flow will be sent to the same egress queue. A detailed description of balanced egress queues is given in section "Egress Queues". Following is just a quick example.

Assuming nProbe™ Cento has to be configured to monitor two eth1 ZC queues such that the traffic coming from each one will be forwarded to two distinct egress queues, it is possible to use the following command

```
./cento -izc:eth1@0 -izc:eth1@1 --balanced-egress-queues 2
```

nProbe™ Cento output confirms the traffic will be properly balanced and forwarded to the egress queues:

```
[...]
Initialized global ZC cluster 10
Created interface zc:eth1@0 bound to 0
Reading packets from interface zc:eth1@0...
Created interface zc:eth1@1 bound to 2
Reading packets from interface zc:eth1@1...
Forwarding interface zc:eth1@0 balanced traffic to zc:10@0
Forwarding interface zc:eth1@0 balanced traffic to zc:10@1
[ZC] Egress queue zc:10@0 (0/2) bound to interface zc:eth1@0
[ZC] Egress queue zc:10@1 (1/2) bound to interface zc:eth1@0
Forwarding interface zc:eth1@1 balanced traffic to zc:10@2
Forwarding interface zc:eth1@1 balanced traffic to zc:10@3
[ZC] Egress queue zc:10@2 (0/2) bound to interface zc:eth1@1
[ZC] Egress queue zc:10@3 (1/2) bound to interface zc:eth1@1
[...]
```

Egress queues are identified with a string `zc:<cluster id>:<queue id>`, e.g., `zc:10@0`. Cluster and queue ids assigned are output by nProbe™ Cento. Egress queues can be fed to other applications such as Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS). A simple packet count can be performed using the PF_RING pfcount utility. Counting packets received on ZC queue 1 on cluster 0 is as simple as

```
pfcount -i zc:10@0
```

For a detailed description of the PF_RING architecture we refer the reader to the PF_RING User's Guide available at https://github.com/ntop/PF_RING/raw/dev/doc/UsersGuide.pdf

[--aggregated-egress-queue|-F [<dev>]]

nProbe™ Cento can aggregate the traffic coming from every input interface and forward it to a single global aggregated egress queue. In other words, nProbe™ Cento run with this option acts as an N-to-1 traffic aggregator. This is basically the "dual" outcome of what is achieved via `--balanced-egress-queues`. In the latter case, nProbe™ Cento acts as a 1-to-N traffic balancer. A detailed description of aggregated egress queues is given in section "Egress Queues". Following is just a brief example.

Assuming nProbe™ Cento has to be configured to aggregate together 4 eth1 ZC queues, such that the traffic coming from each one will be aggregated in a single egress queue, it is sufficient to execute the following command

```
./cento -izc:eth1@0 -izc:eth1@1 -izc:eth1@2 -izc:eth1@3 --aggregated-egress-queue
```

nProbe™ Cento output confirms the traffic will be properly aggregated into a single queue identified by zc:10@0.

```
[...]
Initialized global ZC cluster 10
Created interface zc:eth1@0 bound to 0
Reading packets from interface zc:eth1@0...
Created interface zc:eth1@1 bound to 2
Reading packets from interface zc:eth1@1...
Created interface zc:eth1@2 bound to 4
Reading packets from interface zc:eth1@2...
Created interface zc:eth1@3 bound to 6
Reading packets from interface zc:eth1@3...
Fwd.ing aggregated traffic to zc:10@0, you can now start your recorder/IDS
[...]
```

In general, as already highlighted above, any egress queue is identified with a string `zc:<cluster id>:<queue id>`, e.g., `zc:10@0`.

To count the aggregated traffic it is possible to use the PF_RING pfcount utility as

```
pfcount -i zc:10@0
```

The same queue can be input to traffic recorders such as n2disk as well as to IDS/IPS.

nProbe™ Cento can also aggregate and send incoming traffic to a physical network device. In this case the device name `<dev>` has to be specified right after the option `--aggregated-egress-queue`.

[--egress-conf|-e] <file>

Both aggregated and balanced egress queues allow a fine-grained traffic control through a set of hierarchical rules. Rules are specified in a plain text `<file>` that follows the INI standard. A thorough description of the file format is give in the section "Egress Queues" of the present manual.

Flows Generation

[--lifetime-timeout|-t] <sec>

nProbe™ Cento considers expired and emits any flow that has been active for more than <sec> seconds. If the flow is still active after <sec> seconds, further packets belonging to it will be accounted for on a new flow.

Default lifetime timeout is set to 300 seconds.

[--idle-timeout|-d] <sec>

nProbe™ Cento considers over and emits any flow that has not been transmitting packets for <sec> seconds. If the flow is still active and will transmit again after <sec> seconds, its further packets will be accounted for on a new flow.

This also means that, when applicable (e.g. SNMP walk) UDP flows will not be accounted on a 1 packet/1 flow basis, but on a single global flow that accounts all the traffic. This has the benefit in that it reduces the total number of generated flows and improves the overall collector performance.

Default idle timeout is set to 180 seconds.

CPU Affinity

[--processing-cores|-g <cores>]

nProbe™ Cento spawns a parallel processor thread for every input interface. Any processor thread is in duty to process all the packets associated to its interface and to build the corresponding flows. This design allows to leverage on modern multi-core multi-CPU architectures as threads can work independently and in parallel to process traffic flowing from multiple interfaces.

The affinity enables the user to bind any processor thread (and thus any interface) to a given core. Using this option, nProbe™ Cento will instruct the operating system scheduler to execute processor threads on the specified cores. Cores are identified using their core ids. Core ids can be found directly under /sys/devices.

For example, core ids for a 4-core hyper-threaded CPU Intel Xeon E3-1230 v3 can be found with

```
~$ cat /sys/devices/system/cpu/cpu*/topology/thread_siblings_list
0,4
1,5
2,6
3,7
0,4
1,5
2,6
3,7
```

Every line has two numbers: the first is the core id and the second is the associated hyper-threaded core. Lines are repeated two times as the same information is contained both for the core and for its associated hyper-threaded core. Using affinity and core ids it is possible to greatly optimize cores usage and make sure multiple applications don't interfere each other in high-performance setups.

To start nProbe™ Cento on two ZC interfaces and to bind processor threads to cores with id 0 and 1 respectively it is possible to run the following command

```
./cento -i zc:eth0 -i zc:eth1 -g 0,1
```

[--exporting-cores|-G <cores>]

nProbe™ Cento spawns an exporter thread for every input interface. Any exporter thread deals with the export of flows according to the user submitted options. Examples of exporter threads are Netflow V5 / V5 exporters. The same description given for the `--processing-cores` option applies here to set the affinity of exporter cores.

Flows Export Settings

These settings instruct the interface exporter threads on the output format.

[--dump-path|-P] [<d1>:<d2>:<ml>:]<dir>

nProbe™ Cento exports flows in a plain, pipe-separated text format when `--dump-path` is specified. Every interface is exported in a `<dir>` sub-directory that has the same name of the interface. Every subdirectory contains a tree of directories arranged in a nested, "timeline" fashion as: year, month, day, hour of the day, minute of the hour. The subdirectories that are the "leaves" of this tree contain the actual text files.

The subdirectory tree is populated according to the parameters `<d1>`, `<d2>` and `<ml>`. Specifically, nProbe™ Cento creates a new subdirectory every `<d1>` seconds. A text file contains at most `<ml>` flows, one per line. If more that `<ml>` flows are going to be written in less that `<d2>` seconds, then one or more files are created. If less that `<ml>` flows have been written in `<d2>` seconds, the flows file is closed and a new one is created.

For example, to capture from two eth1 ZC queues and write gathered flows to /tmp it is possible to use the following command

```
./cento -izc:eth1@0 -izc:eth1@1 -P 300:50:20000:/tmp/
```

The command instructs nProbe™ Cento to dump flows into /tmp with the extra conditions that:

- Every flows file shouldn't contain more that 20000 flows and shouldn't contain flows worth more than 50 seconds of traffic.
- Every subdirectory shouldn't contain more that 300 seconds of traffic.

nProbe™ Cento output confirms it is going to use two subdirectories in /tmp one for each interface

```
[...]
Dumping flows onto /tmp/zc_eth1@0
Dumping flows onto /tmp/zc_eth1@1
[...]
```

An example of the resulting directory structure is as follow

```
~/ find  /tmp/zc_eth1\@0/*
/tmp/zc_eth1@0/2016
/tmp/zc_eth1@0/2016/05
/tmp/zc_eth1@0/2016/05/04
/tmp/zc_eth1@0/2016/05/04/18
/tmp/zc_eth1@0/2016/05/04/18/1462379942_0.txt
/tmp/zc_eth1@0/2016/05/04/18/1462380145_0.txt
/tmp/zc_eth1@0/2016/05/04/18/1462379214_0.txt
/tmp/zc_eth1@0/2016/05/04/18/1462379603_0.txt
/tmp/zc_eth1@0/2016/05/04/18/1462379881_0.txt
```

The default value for `<d1>` is 300 seconds.
The default value for `<d2>` is 60 seconds.
The default value for `<ml>` is 10000 seconds.

[--kafka <brokers>;<topic>[;<ack>;<compr>]

nProbe™ Cento can export flows to one or more Kafka <brokers> that are responsible for a given <topic> in a cluster. While the topic is a plain text string, Kafka brokers must be specified as a comma-separated list according to the following format

```
<host1>[:<port1>],<host2>[:<port2>]
```

Initially, nProbe™ Cento tries to contact one or more user-specified brokers to retrieve Kafka cluster metadata. Metadata include, among other things, the list of brokers available in the cluster that are responsible for a given user-specified topic. Once the metadata-retrieved list of brokers is available, nProbe™ Cento starts exporting flows to them.

The user can also decide to compress data indicating in <compr> one of none, gizp, and snappy. Additionally, the <ack> policy can be specified as follows:

- <ack> = 0: Don't wait for ACKs
- <ack> = 1: Receive an ACK only from the Kafka leader
- <ack> = -1: Receive an ACK from every replica

Example

Assuming there is a Zookeeper on localhost port 2181 that manages the Kafka cluster, it is possible to ask for the creation of a new Kafka topic. Let's say the topic is called "topicFlows" and it has to be split across three partitions with replication factor two. The command that has to be issued is the following

```
bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic topicFlows \
--partitions 3 --replication-factor 2
```

Now that the Kafka topic has been created, it is possible to execute nProbe™ Cento and tell the instance to export to Kafka topic "topicFlows". We also tell the instance a list of three brokers all running on localhost (on ports 9092, 9093 and 9094 respectively) that will be queried at the beginning to obtain kafka cluster/topic metadata.

```
./cento --kafka "127.0.0.1:9092,127.0.0.1:9093,127.0.0.1:9094;topicFlows"
```

At this point the nProbe™ Cento instance is actively exporting flows to the Kafka cluster. To verify that everything is running properly, it is possible to take messages out of the Kafka topic with the following command

```
bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic topicFlows\
--from-beginning
```

[--v5|-5 <host:port>]

Exports NetFlow to the destination <host:port> using version 5.

[--v9|-9 <host:port>]

Exports NetFlow to the destination <host:port> using version 9.

[--ipfix|-I <host:port>]

Exports IPFIX to the destination <host:port>.

[--monitor|-A]

Export flows to a monitoring virtual interface for traffic visibility

Miscellaneous Settings

[--dpi-level|-D] <level>

This option enables Deep Packet Inspection (DPI) for flows. DPI attempts to detect Layer-7 Application Protocols by looking at packet contents rather than just using Layer-4 protocols and ports. As DPI is a costly operation in high-speed environments, nProbe™ Cento can optionally operate in μ-DPI mode that increases performances by focusing only on the most common Layer-7 protocols.

The `<level>` is used to specify the DPI version that has to be activated:

- `0`: DPI disabled
- `1`: enable uDPI (micro DPI engine)
- `2`: enable nDPI (full fledged nDPI engine)

[--active-poll|-a]

Enables active packet polling. Using this option nProbe™ Cento processes will actively spin to check for new packets rather than using polling mechanisms or sleeps.

[--pid-file|-p <path>]

Write the process Id in the specified file

[--hash-size|-w] <size>

Sets the per-interface flow cache hash size. The size is exposed using a value optionally followed by the suffix "MB". If the suffix is not specified, then the value is interpreted as the number of buckets. If the suffix is specified, then the value is interpreted as the in-memory size of the flow cache.

Default: 512000 [estimated hash memory used 144.5 MB]

[--redis|-R] <host[:port][@db-id]>

Redis DB host[:port][@database id]

[--version]

Print the version and system identifier

[--check-license]

Checks if the license is present and valid

[--check-maintenance]

Checks the maintenance status for the specified license

[--local-nets|-L] <local nets>
Local nets list (default: no address defined). Example: -L "192.168.0.0/24,172.16.0.0/16". Simone: TODO: ask Luca the use of local networks here.

[--banned-hosts|-B] <path>

nProbe™ Cento features a host ban functionality that can be enabled using this option. The list of banned hosts must be specified, one per line, in a text file located at `<path>`. *μ*-nDPI engine is required.

For example, to ban facebook.com and google.com from interface eth0, one should create a text file, let's say `banned.txt`, with the following two lines

```
google.com
facebook.com
```

An then start nProbe™ Cento as

```
./cento -ieth0 -D 1 -B /tmp/banned.txt
```

Zero Copy

[--zc|-Z]

Forces the use of zero copy on linux. Simone: TODO: for what? I checked that input interfaces still require the prefix zc.

[--cluster-id|-C] <id>

Forces nProbe™ Cento to use the ZC cluster id <id>. If no cluster id is specified, and arbitrary id will be assigned by nProbe™ Cento.

REST

Luca: Curl examples ought to be added.

[--rest-address|-Y <address>]

Enable a REST server binding it to the specified address

[--rest-http-port|-T <port>]

REST server HTTP port

[--rest-https-port|-S] <port>

REST server HTTPs port

[--rest-docs-root|-J] <dir>

REST server docs root directory

1 Official PF_RING repository: https://github.com/ntop/PF_RING

2 Official PF_RING ZC page: http://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/

3 Apache Hadoop page: http://hadoop.apache.org/

4 Apache Spark Streaming page: http://spark.apache.org/streaming/

5 Apache Storm page: http://storm.apache.org/

6 Apache Samza page: http://samza.apache.org/

7 Zero-MQ page: http://zeromq.org/

8 NetFlow v9 field type definitions: http://www.ibm.com/support/knowledgecenter/SSCVHB_1.1.0/collector/cnpi_collector_v9_fiels_types.dita

9 n2disk page: http://www.ntop.org/products/traffic-recording-replay/n2disk/

10 INI standard: https://en.wikipedia.org/wiki/INI_file

11 Suricata page: https://suricata-ids.org/

12 Snort page: https://www.snort.org/