Seventh FRAMEWORK PROGRAMME

FP7-ICT-2007-2 - ICT-2007-1.6

New Paradigms and Experimental Facilities

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D3.2

"Implementation of adaptive traffic sampling and management, path performance monitoring and cooperative intrusion and attack/anomaly detection techniques"

Project description
Project acronym: **ECODE** Project full title: **Experimental Cognitive Distributed Engine** Grant Agreement no.: **223936**
Document Properties
Number: FP7-ICT-2007-2-1.6-223936-D3.2 **Title**: Implementation of adaptive traffic sampling and management, path performance monitoring, and cooperative intrusion and attack/anomaly detection techniques **Responsible**: Philippe Owezarski (CNRS) **Editor(s)**: Philippe Owezarski (CNRS) **Contributor(s)**: Chadi Barakat (INRIA), Amir Krifa (INRIA), Yann Labit (CNRS), Imed Lassoued (INRIA), Johan Mazel (CNRS), Philippe Owezarski (CNRS), Kavé Salamatian (ULANC) **Dissemination level**: Public (PU) **Date of preparation:** October 30, 2009 **Version:** 1.0

D3.2 - ***Implementation of adaptive traffic sampling and management, path performance monitoring and cooperative intrusion and attack/anomaly detection techniques***

Executive Summary

The aim of the ECODE experimental research project is to introduce a new Internet architectural component realized by means of a cognitive system and that preserves the original Internet design principles, including the end-to-end principle and its transparency, to sustain growth of an Internet that remains inline with what it supposed to deliver to the end-user and that performs in accordance to what it is expected to deliver to the end-user. As the purpose of this new architectural component is to sustain growth of an Internet that performs in accordance to what it is supposed to deliver to the end-user and performs according to these expectations in order to satisfy the end-users, large-scale testing and validation is explicitly in the scope of this research project. Combined experimentation will allow determining whether composing the Internet high-level goals - societal, economical, etc. - can be translated into lower-level objectives (in terms of functionality and performance) and constraints (both technical and non-technical) and enforced via the newly introduced cognitive component as part of the Internet routing system.

This deliverable is part of a series of 3 deliverables – one per technical objectives (TO) [ECODE 2007] – which aims at describing the design and the implementation of the different use case functions. More specifically, this deliverable addresses the implementation, development and first experiments performed in the framework of TO1 related to:

- Adaptive sampling
- Path performance monitoring
- Cooperative intrusion and attack/anomaly detection

This deliverable describes the technical problems that need to be solved. A particular attention is given to the machine learning techniques designed and implemented for the different use cases; machine learning techniques are the heart of this deliverable (as well as of the whole project) as they are providing the necessary advantage for a better router design, and better network performances and manageability. After having described how the monitoring techniques are implemented for solving the different use cases, first performance evaluations are detailed. At this stage, evaluation of the concepts and their implementation is done by means of simulation and/or small scale experiments (emulation most of the time, but can be also experiment in real environments). Nevertheless, the first comparisons with other solutions can be made, and are exhibiting the gain provided by the ECODE router architecture based on the integration of strong cognition capabilities for enforcing and managing routing functions.

Last, first premises on how the different components designed in TO1 will be integrated all together in the future ECODE cognitive router, are studied and detailed in Section 6.

List of authors

Affiliation	Author
CNRS	Philippe Owezarski
CNRS	Johan Mazel
CNRS	Yann Labit
INRIA	Chadi Barakat
INRIA	Amir Krifa
INRIA	Imed Lassoued
ULANC	Kavé Salamatian

Table of Contents

Executive Summary **2**
List of authors **3**
Table of Contents **4**
1. Introduction **6**
2. Formalization of the different technical problems **7**
2.1 Case a1: Adaptive traffic sampling and management 7
2.2 Case a2: Global monitoring 7
2.3 Case a3: Cooperative distributed anomaly detection 7
2.3.1 Attribute based local detection and classification of anomalies 8
2.3.2 Distributed detection of anomalies 10
3. Machine Learning techniques/algorithmic **11**
3.1 Case a1: Adaptive traffic sampling and management 11
3.1.1 Monitoring Engine (ME) 12
3.1.2 Machine Learning Engine 12
3.2 Case a2: Global monitoring 13
3.3 Case a3: Cooperative distributed anomaly detection 13
3.3.1 Attribute based local detection of anomalies 13
3.3.2 Primary selection of machine learning techniques for anomaly detection 13
3.3.3 Application of machine learning to anomaly detection 13
3.3.4 Application of machine learning to anomaly classification 14
3.3.5 Conclusion 18
3.3.6 Distributed detection of anomalies 19
4. Implementation **20**
4.1 Case a1: Adaptive traffic sampling and management 20
4.1.1 The Traffic Emulation Service 20
4.1.2 The Traffic Monitoring and Sampling Service 22
4.1.3 The Data Collection and Analysis Service 23
4.2 Case a2: Global monitoring 23
4.2.1 Specification 23
4.2.2 Global architecture 24
4.2.3 Passive measurements 25
4.2.4 Traffic processing 28
4.3 Case a3: Cooperative distributed anomaly detection 29
4.3.1 Attribute based local detection of anomalies 29
4.3.2 Distributed detection of anomalies 33
5. Experimentation **35**
5.1 Case a1: Adaptive traffic sampling and management 35
5.1.1 Performance objectives, and evaluation criteria 35
5.1.2 Methodology: scenarios and tools 37
5.1.3 Future work 39
5.2 Case a2: Global monitoring 39
5.2.1 Performance objectives, and evaluation criteria 39
5.2.2 Methodology: scenarios and tools 39
5.2.3 Experimental results 40
5.2.4 Future work 40
5.3 Case a3: Cooperative distributed anomaly detection 40
5.3.1 Attribute based local detection of anomalies 40
5.3.2 Distributed detection of anomalies 42
6. Recommendation for integration into common ECODE architecture (see D2.1) **45**
6.1 Case a1: Adaptive traffic sampling and management 45
6.1.1 Communication between the MLE and the ME 45
6.1.2 Communication between two different MLE(s) 46
6.2 Case a2: Global monitoring 46
6.3 Case a3: Cooperative distributed anomaly detection 46
6.3.1 Attribute based local detection and classification of anomalies 46
6.3.2 Distributed detection of anomalies 47
6.3.3 Communication between the MLE and the ME 47
6.3.4 Communication between MLEs 47

6.3.5 Communication between the MLE and the FE and the RE 47
7. Conclusion 48
8. References 49
Annex 1: Supervised and semi-supervised learning 50
Annex 2: Linear Estimation 52
Annex 3: Source Compression 53

1. Introduction

This deliverable aims at describing the design and the implementation of the different use case functions of TO1. More specifically, it addresses the implementation, development and first experiments related to:

- Adaptive sampling
- Path performance monitoring
- Cooperative intrusion and attack/anomaly detection

This deliverable describes the technical problems that need to be solved for each of these use cases. A particular attention is given to the machine learning techniques designed and implemented for the different use cases; machine learning techniques are the corner stone of this deliverable (as well as of the whole project) as they are providing the necessary advantage for a better router design, and better network performances and manageability.

After having described how the monitoring techniques are implemented for solving the different use cases, first results of performance evaluations of the proposed solutions are detailed. At this stage, evaluation of the concepts proposed and their implementation is performed by means of simulation and/or small scale experiments (emulation most of the time, but can be also experiment in real environments). Nevertheless, the first comparisons with other solutions can be made, and are exhibiting the gain provided by the ECODE router architecture based on the integration of strong cognition capabilities for enforcing and controlling routing functions.

Last, it will be explained how the different components designed in TO1 will be integrated all together in the ECODE global architecture

2. Formalization of the different technical problems

2.1 Case a1: Adaptive traffic sampling and management

The main technical problems we faced were:

- Getting NetFlow widely collected traces of real traffic.
- Having control of the different measurement points towards running our adaptive traffic sampling and management solution online.

These difficulties triggered the need for fully controlled and realistic platform. As part of our contribution to the ECODE project, we present the validation platform we used to evaluate our adaptive traffic sampling and management solution. Note that the platform we are proposing could also be used by other partners who are encountering the same challenges and targeting the development of adaptive architectures.

2.2 Case a2: Global monitoring

The global monitoring system consists of two subsystems:

- The passive monitoring and measurement system
- The active monitoring and measurement system

Note that the active probing system will not be described in this deliverable. As the active monitoring probing fits TO2 requirements, it is presented in deliverable D3.4 for readability and understanding purposes.

On the other hand, the passive monitoring system is essentially used in use cases A1 and A3 of the first technical objective. Therefore, its design and implementation are described in this deliverable. The passive monitoring system to be developed in the ECODE project is devoted to serve for all machine learning methods which will require packet traces. It is supposed to provide two main facilities:

- capture packets,
- and providing a framework for easily integrating any analysis module for the captured packet.

Concerning the capture, it can be launched on any interface on-the-fly. The analysis framework needs to offer two facilities:

- launch any analysis on the fly
- export the result of these analysis through logging, reporting or trace generation.

The analysis framework also needs to be able to launch, stop and manage several analysis sessions independently.

2.3 Case a3: Cooperative distributed anomaly detection

The Internet has greatly grown in complexity, changing from a single best effort service to a multi-services network that is ever more demanding of guaranteed quality of service (QoS), for instance for Voice over IP (VoIP) or IP TV which require a strict delivery process of audio and/or video data units. Network traffic anomalies can seriously impact or disrupt the normal operation of networks. It is then vital that their detection and mitigation be quickly identified by network administrators. A specific type, volume anomalies, is responsible for unusual modifications on network traffic volume characteristics (normally identified on the #packets, #bytes and/or #new flows). These anomalies can be caused by various events: from physical or technical network problems (e.g. outages, routers mis-configuration), to intentionally malicious behaviour (e.g. denial-of-service attacks, worms related traffic), to abrupt changes caused by legitimate traffic (e.g. flash crowds, alpha flows). This diversity coupled with the great (natural) variability of normal Internet traffic volume [Owezarski 2005], makes the identification and mitigation of these anomalies a very challenging task.

Despite these difficulties, constant progress has been realized in network traffic anomaly detection. Methods have been created to detect anomalies in single-links and network-wide data, and techniques have been used to cope with the high dimensionality of network traffic data (e.g. sketches [Li 2006] [Dewaele 2007] and

principal components [Lakhina 2004] [Lakhina 2005]. Algorithms for network traffic anomaly detection have evolved from only being able to timely signal an anomaly (e.g. [Barford 2002] [Scherrer 2007]) to providing information about the actual flows that cause the anomaly [Li 2006] [Dewaele 2007]. This information is necessary for network administrators that need to manually verify and mitigate potential anomalies, but is still not sufficient. Because of the characteristics of network traffic and the frequency of anomalies, real-time analysis of all anomalies detected by state-of-the-art detection algorithms is not possible. Network operators need more information than just the anomalous flows to efficiently prioritize between detected anomalies.

2.3.1 Attribute based local detection and classification of anomalies

Although some efforts have been dedicated to characterize network traffic anomalies, automated classification has not received much attention (a notable exception is [Lakhina 2005]). Automated classification intends to add meaningful information to the alert of a detected anomaly. Ideally, the computed information can then be used to define the type of the anomaly or to at least help characterizing autonomously its underlying cause. In this use case, we propose a new algorithm for automated classification of network traffic anomalies. We show how the information obtained by further analyzing the identified anomalous flows, can be used in a signature-based classification module to reliably characterize different types of anomalies, e.g., DDoS, network scans, attack responses. One of the objectives of this approach is to provide the flexibility needed by network operators to understand and manipulate the anomaly classification process.

For this purpose, two issues have to be addresses leading to a two-step algorithm:

(i) Detect anomalies and identify all (or most) related packets or flow;

(ii) Use these packets or flow records to derive several distinct metrics directly related to the anomaly and classify the anomaly using these metrics with a signature-based approach.

These steps are based on the observation that much information is needed to reliably classify different types of anomalies and even to distinguish between subtypes, like the many types of DoS attacks. Since current detection algorithms are based on few parameters (i.e. traffic volume metrics or traffic features like IP address and ports), steps are necessary to obtain more information about the anomaly. Naturally, the best source of information is traces of packets or records of flows that actually cause the anomaly. From now on, we will refer only to packets traces, but similar results can be obtained using flow records.

For the first step, we use a variation of the simple traffic volume anomaly detection algorithm presented on [Farraposo 2007]. The detection algorithm can be explained as follows. Given a trace of duration T and a time-scale granularity of Δ (i.e. 30s throughout this report), divide the trace in N slots where $N \in [1, T/\Delta]$. For each slot i obtain the data time series X of each traffic volume metric $\in$ {#packets, #bytes, #syn}. Obtain the absolute deltoids [Cormode 2005], i.e., absolute difference P of X and calculate their standard deviation $\sigma(p)$. For any p_i over the threshold $K*\sigma(p)$, mark its slot as anomalous. Using the deltoids of the data time series is important to consider the variation over the amplitude of the curve instead of the variation of network traffic, as the latter is insignificant due to its natural high variability. Our choice of metrics is based on [Lakhina 2004] (with #syn instead of #new flows), but the algorithm permits the use of any other data time series.

$$\Delta = time\_granularity$$
$$X = \{x_1, x_2, ..., x_n\}, x_i = \{\#\ packets\ |\#\ bytes\ |\#\ flows\}/\Delta$$
$$P = \{p_1, p_2, ..., p_{n-1}\}, p_i = x_{i+1} - x_i$$
$$\begin{cases} p_i \geq k\sigma_p, anomalous \\ p_i < k\sigma_p, normal \end{cases}$$

Detection of low intensity anomalies is important especially for DDoS anomalies [Owezarski 2005] and for anomalies concealed in highly aggregated traffic. To detect low intensity anomalies, we apply the detection algorithm to different aggregation levels at the same time. Aggregation is performed based on destination IP address and a bit mask modifier for each packet. Up to now, we use the following prefix sizes as aggregation levels /0 (i.e. whole traffic), /8, /16 and /24. As with any other detection algorithm, this increase in sensitivity generates a higher rate of false positives (i.e. normal traffic variations are considered anomalous). With the

multi-level feature, the algorithm presented above is particularly sensitive to infrequent communications where only a few packets are seen for a given network/mask aggregation. Although this would generally make the algorithm unusable, we built the classification process as a filter that greatly reduces the number of false positives. The simplicity of the detection algorithm makes the next step (i.e. identification of corresponding packets and derivation of metrics) a straightforward task and permitted us to concentrate on the characterization of anomalies.

2.3.1.1 Obtaining information

From the characterization of network traffic anomalies as proposed in [Barford 2002] [Lakhina 2004] [Lakhina 2005], different types of anomalies can affect volume metrics and traffic features, such as IP addresses and ports, in the same manner. This clearly shows that we cannot perform reliable classification based only on these metrics, and further information needs to be identified. For this purpose, we introduce the notion of anomaly attributes. An anomaly attribute is a feature that helps to characterize a specific anomaly (see Table 1). The classification module uses signatures based on attributes derived directly from the packets that compose the anomaly.

The detection algorithm retrieves these packets straightforwardly. A detected anomaly is identified by its slot, network address and mask. We also know exactly why it was considered anomalous (i.e. the deltoid for one or more of the volume metrics was above the threshold). Using this information, we then read all packets of the corresponding slot that are destined to that network, so that we can find the responsible destination hosts (i.e. IP address/32). Our idea of responsible destinations is similar to the notion of dominant IP address range and/or port of [Lakhina 2004]. In our algorithm, the set of responsible destinations comprises all the destination hosts that appear in any of the possible combinations of minimum sets that would bring the anomaly's corresponding deltoid below a fraction of the original threshold. After identifying these hosts, we follow an equivalent approach to determine the responsible sources, ports and protocols. This notion could also be applied to any other traffic feature. Potentially, finding the packets (or flows) that compose an anomaly can be performed with any detection algorithm that identifies the starting time and anomalous flows of the anomalies (e.g. [Li 2006] [Dewaele 2007]).

Attribute	Type	Description
found{p,b,s}	integer	If the corresponding metric was anomalous, value of P, zero otherwise
impactlevel{p,b,s}	integer	The impactlevel of the anomaly (see Section 3.1)
duration{p,b,s}	integer	For how many slots the metric stayed above the threshold
decrease{p,b,s}	float	The biggest negative deltoid during the anomaly as a fraction of the threshold
#respdest	integer	Number of responsible destinations
#rsrc/#rdst	integer	Ratio of responsible sources to responsible destinations
avg#rdstports	integer	Average number of responsible destination ports
avg#rsrcports	integer	Average number of responsible source ports
#rpkt/#rdstport	integer	Ratio of number of packets to responsible destination ports
#rpkt/#rsrc	integer	Average number of packets of responsible sources
bpprop	integer	Average packet size (only packets of the anomaly)
spprop	float	Ratio of number of syn to number of packets of the anomaly
samesrcpred	boolean	If a specific responsible source appears for the majority of destinations
samesrcportpred	boolean	If the majority of responsible sources use the same source ports
oneportpred	boolean	If only one destination port dominated
invalidpred	boolean	If the anomaly was mainly consisted of invalid packets (e.g. malformed headers)
invprotopred	boolean	If the anomaly was dominated by packets using invalid protocol numbers or types
landpred	boolean	If most packets had the same source and destination IPs
echopred	boolean	If most packets were of type ICMP Echo Request/Reply
icmppred	boolean	If most packets were ICMP of any other type
rstpred	boolean	If most packets were TCP with RST flag set

Table 1. Attributes derived from a given anomaly. p, b and s are for packets, bytes and syn respectively

During the anomaly detection and responsible flows identification phases, we compute the attributes shown in Table 1. Attributes *found* and *impactlevel* are specific to the detection algorithm we use in this work, but similar attributes should be available for other detection algorithms. The remaining attributes are derived

while identifying the responsible flows. This list is by no means exhaustive and can be further extended. These attributes are those we identified as useful up to now.

2.3.1.2 Classification

The main objective of our algorithm is to automatically label network traffic anomalies during their actual detection. The vast number of different types of anomalies [Lakhina 2004] and the variations of individual types make it necessary to create very specialized signatures to achieve low misclassification rates. We finally created 5 rules which are able to classify any known anomaly types. The 5 rules are indicated in Table 2.

Id	Anomaly Type	Signature
1	ICMP Echo DDoS	#respdest == 1 and echopred and (#rpkt/#rdstport > 30*gr or #rsrc/#rdest > 15)
2	TCP SYN DDoS	#respdest == 1 and founds and spprop > 0.9 and oneportpred and #rpkt/#rdstport > 10*gr
3	Network Scan	#respdest > 200 and samesrcpred
4	SYN Port Scan	#respdest == 1 and #rsrc/#rdest == 1 and spprop > 0.8 and avg#rdstports > 5
5	Attack Response	#respdest == 1 and (rstpred or icmppred) and foundp > 20*gr and (not (impactlevelp == 3)) and (#rsrc/#rdest == 1 or samesrcportspred)

Table 2. Examples of strong signatures used in this work. (gr stands for the time series granularity and sspp is an abbreviation for the attribute samesrcportspred)

For more information about this anomaly detection / classification algorithm, we refer to [Fernandes 2009].

In this context, the following objectives we started to work on deal with:

- determine the thresholds for each element of each rules for the classification system
- find new rules for detecting new anomalies.

2.3.2 Distributed detection of anomalies

The classical setting in networking generally assumes that the information that is exchanged is statistically independent. However very frequently this is not strictly the case and nodes do exchange information that is correlated. Distributed anomaly detection has to deal precisely with such a setting: each monitoring node in the network maintains a vector of numerical state information. As the different monitoring nodes observe interacting traffics the state maintained by them is correlated.

Distributed monitoring and anomaly detection involves exchanging information between nodes such that each node obtains an approximate view of the states of all other nodes. Through the information exchanged the local observation of a node is extended to the global state of the network. One important challenge here is to deal with nodes selfishness, *i.e.* a node wants to achieve the best approximation of other nodes states consuming itself the lowest amount of resources. This means that we need an incentive/punishment cooperative mechanism to motivate node to exchange information. One should also assume that the quality of approximation about other states is not defined *a priori*; it needs to be defined online during the operation of the distributed system. The distributed anomaly detection, a communication scheme initially designed by Lancaster University enables the sharing of correlated node states; a punishment scheme has been proposed that solves the node selfishness issue; a signalling method has been proposed that enables a node to announce to other nodes its state variable of interest as well as their importance. The proposed scheme merges approximation obtained at different nodes into a single consistent approximation with better quality. We will show that the proposed scheme implements a negotiation between neighbours that trade-off an increase of the node transmission rate to its neighbour with a mechanism achieving a better approximation of the state of other nodes.

3. Machine Learning techniques/algorithmic

3.1 Case a1: Adaptive traffic sampling and management

The main goal is to build a network-wide system that, given a measurement task (in the form of a filter to apply on the traffic) and an overhead threshold, is able to:

- Collect and obtain data from different monitors deployed in network routers. All monitors integrate sampled NetFlow-like traffic capturing tools as well as reporting capabilities for exchanging information with the collector.
- Introduce a cognitive component that processes the collected measurements from the different routers, correlates them and calculates the targeted metric together with its estimated error. This component also measures the amount of overhead caused by the actual configuration of monitors (volume of collected measurements, CPU, memory, disk space, etc).
- Drive its own deployment by automatically and periodically reconfiguring the different monitors in a way that improves the overall accuracy (according to monitoring application requirements) and reduce the resulting overhead (respecting some resource consumption constraints).
- Support a general class of monitoring applications. In the sequel, algorithms and results will be illustrated by means of a traffic accounting application.

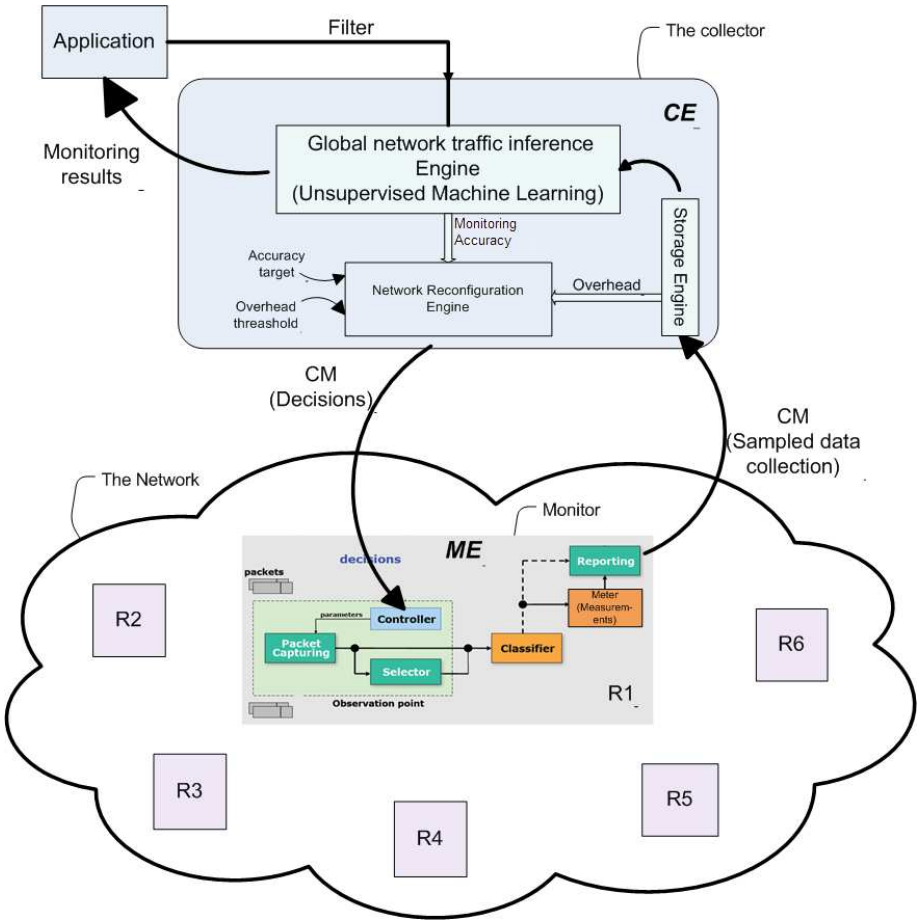

Figure 1: General principle of adaptive sampling

As illustrated in Figure 1, the proposed monitoring system relies on local NetFlow-like measurement tools (Monitoring Engine (ME)) deployed in network routers as well as reporting capabilities for exchanging information and decisions with the central unit (Cognitive Engine (CE) referred to as Machine Learning Engine since using Machine Learning techniques). The targeted application provides input to the system in the form of

a filter to apply on the data, and the system adjusts the sampling rates in routers to answer the application needs with the best accuracy and the lowest overhead. Next, we give a detailed description of the architecture components.

3.1.1 Monitoring Engine (ME)

The monitoring engine runs in each router and aims at sampling and capturing packets at the interfaces of the router. It then exports them in the form of NetFlow records to the central collector. As depicted in Figure 1, one can observe four main modules:

- **Packet capturing**: this module listens to the network interface and sample data at a given sampling rate. This sampling rate is configured each time by the Cognitive Engine (CE) next to the optimization it carries out after correlating measurements from all routers.
- **Classifier**: Once a packet is sampled by the Packet capturing module, the classifier identifies flows by a key (in our case this key corresponds to the 5-tuple consisting of source and destination addresses, source and destination port numbers, and protocol number). The Classifier then determines if a flow is active or if it is a new flow. If the flow is active, it updates real-time statistics on that flow such as the number of packets and bytes. If it is a newly observed flow, it inserts a new flow record for this new packet's key. The ME maintains the keys of flows forwarded by the router together with the collected statistics on those flows. A flow is declared terminated by the Classifier in one of three cases: *(i)* when observing a FIN or a RST packet (TCP control), *(ii)* when a timeout expires after the record for that flow was created, and finally *(iii)* when the number of records exceeds a given threshold in order to release memory.
- **Reporting**: Once collected, flow records are exported using UDP messages to the central Unit (Cognitive Engine) through the CM (Cognitive-Monitoring) interface.
- **Controller**: Based on the collected data and machine learning methods, the cognitive component takes a decision on how to tune the sampling rates and sends the decision back to the ME. The controller in each router receives the decision and updates the sampling rate in the ME accordingly.

3.1.2 Machine Learning Engine

This component is motivated by the need to extend the local existing monitoring engines (see Section 3.1.1) with a network-wide machine learning engine (MLE) implementing machine learning methods and able to:

- Investigate the local measurements collected from the different MEs (local view) to construct a global view of the traffic and the network state.
- Automate and enhance network-wide monitoring control capabilities while decreasing their resulting cost. The automation of the control of sampling rates is achieved by learning experiences from the accuracy of the collected data and the overhead of measurements.

The (processing part of the) MLE runs two processes:

3.1.2.1 Global Network Traffic Inference Process

Given a measurement task to realize, the Inference process investigates the local measurements made by the different routers to have a global more reliable view. The engine takes as inputs the sampling rate vector of network routers as well as the local estimations calculated from the reports sent to the collector by the different MEs and stored in the Observation Information Base (OIB). The process tries then to combine the local estimators and derive a better estimation. This combination should be motivated by the need to minimize the amplitude of estimation errors given the monitoring objective one has in mind. To this end, we start by constructing a global estimator for each network flow as a weighted sum of the different local estimators. The weights are chosen to be inversely proportional to the estimation errors of the local estimators. This way, local estimates with smaller errors have a larger impact on the global estimator than those with high errors. Such weighted summation of local independent estimators is known to be the best linear combination one can envisage in terms of mean square error.

3.1.2.2 Network Reconfiguration Process

The Network Reconfiguration process is fed with the global estimation and its accuracy from the Inference process as well as the resulting overhead from the OIB. It aims to coordinate responsibilities across the different monitors (routers) in order to increase global accuracy while avoiding unnecessary measurements. To achieve this goal, we proceed by an adaptive centralized control of sampling rates in routers based on measurements of the estimation error and the overhead. Periodically, we collect measurements and calculate the estimation accuracy and the overhead. Then, we propose one of two actions: ***(i)*** either to decrease the sampling rate of the least significant monitor when the resulting overhead is higher than some overhead threshold, **(*ii*)** or to increase the sampling rate of the best monitor when the estimation accuracy is below some chosen threshold and the overhead is acceptable. The least significant and best monitors are respectively those providing the minimum loss and maximum gain in global accuracy when playing with the sampling rate. This method, which can be seen as an implementation of the Gradient Projection Method (GPM), is able to reach the optimum in a relatively short time that depends on the steps with which we adapt the sampling rates. The system keeps oscillating around this optimal configuration until the network conditions change, in this case it moves smoothly towards the new optimal configuration.

3.2 Case a2: Global monitoring

By design, monitoring is considered as a basic functionality aiming at quickly providing the traffic information required by the different other modules developed in the other use cases. The monitoring engine is thus not the most appropriate component for deciding what kind of traffic information is best suited for other modules that require this information. Thus, the monitoring engine does not implement machine learning techniques. The machine learning functionality is integrated at the upper level, i.e., by the different modules (sampling, anomaly detection, etc.).

3.3 Case a3: Cooperative distributed anomaly detection

3.3.1 Attribute based local detection of anomalies

The use of machine learning in the context of anomaly detection aims at:

- determining the thresholds for each element of each rules for the classification system
- autonomously finding new rules for detecting new anomalies.

3.3.2 Primary selection of machine learning techniques for anomaly detection

See Annex 1.

3.3.3 Application of machine learning to anomaly detection

3.3.3.1 Objectives

In this step of our anomaly detection system, the goal is to detect every single anomalous event. In order to do this, we currently use a model based on deltoids. However, we plan to test several other techniques like models based on Gamma-FARIMA eventually including sketch-based techniques. Based on traffic models, there is a need to use machine learning to determine the thresholds to be used for the detection of anomalies events.

3.3.3.2 Supervised/semi-supervised learning and detection

These two techniques require data for the training phase. Training data consists of several packet traces including some anomalies. Inside the packet traces, each packet will be marked in order to know whether it is part of an anomaly or not and if yes, which one.

In practical terms, we will split a trace in several time slots. We will then apply our model (deltoid, Gamma-FARIMA, etc.) on each time slotted data. At the same time, we plan to mark every slot containing an anomaly by using our own knowledge of the data contained in the traces, e.g., the knowledge potentially resulting of previous analysis of the traces, or because we could have synthesized ourselves these training traces.

As exposed in the previous chapter, the training data used by a supervised algorithm is composed of a pair of objects. The first element of the pair is the value to feed the learning algorithm with, here, the data processed by the model. The second element is the result, here, the presence or not of an anomaly. In the same way that the supervised algorithm can determine the threshold for the temperature and humidity that make the player go to the green to play golf (cf. Annex 1), we can determine the thresholds for the values of the models that predict that we have an anomaly.

3.3.3.3 Unsupervised learning and detection

Due to its simplicity and ease-of-use, we plan to first try to implement supervised learning. However, we still keep into consideration the capabilities of unsupervised learning. In fact, depending on the results of supervised learning, we might be lead to implement unsupervised learning technique.

3.3.4 Application of machine learning to anomaly classification

3.3.4.1 Objectives

In this step of our anomaly detection system, the goal is to classify the anomalies detected by the detection system. For this purpose, we use the rules presented in the Table 1.2.

We have two objectives as part of the anomaly classification step:

- Apply a machine learning algorithm that will allow us to determine the threshold for these signature based rules.
- Discover new anomalies i.e. anomalies never seen before (0day anomalies). This is definitely one of the most difficult goals to accomplish in anomaly detection. In our system, any 0day anomaly needs to be associated with a new rule in order to be detected. Our goal is to automate this process and to be able to automatically find a new detection rule suited for the new 0day anomaly.

3.3.4.2 Supervised/semi-supervised learning and classification

1. Application

For reaching the first objective (finding automatically existing rules thresholds), training data is needed. Such data consists of several packet traces including some anomalies inside. After the anomaly detection stage, each packet will be marked to indicate whether it is part or not of an anomaly and if yes, which one. Supervised and semi-supervised learning algorithms are perfectly suited for re-determining the rules and determining the thresholds in a reliable and simple way.

2. Restrictions

However, these algorithms present several limitations. First of all, their use implies that we have labelled data at our disposal. In our case, these data are traces where the anomalous packets have been previously labelled. The process used to obtain these traces is long and tedious. Moreover, these types of algorithms are based on labelled anomaly. This implies that the labelled anomalies are known and identified. This excludes one of the goals of the system, i.e.: the discovery of anomalies never seen before (as 0day attacks).

3.3.4.3 Unsupervised learning and classification

In this section, the second of this work is addressed, i.e. automatically generating new rules for 0day anomalies.

1. Representation of anomalies in unsupervised learning

First of all, when we address the representation of traffic in our multi-dimensional data space, we consider each point as a packet. Each point will then be either associated to a model representing a traffic class (normal or anomalous) or, if it is considered too far from the model(s), let alone and be considered as an outlier. It is the link between being part of a model or being an outlier and belonging to the normal or anomalous traffic that we will address in this subsection.

In all previous work that we are aware of, two possible representations of network traffic through unsupervised learning have been considered. In the first representation, the network traffic is represented by several models and each model is associated with a part of the network traffic. In Figure 7, one can observe presence of several models (two actually) that can represent either a class of normal traffic or a class of anomalous traffic. If Figure 4, there is one (or more) cluster(s) for the normal network traffic (here the green cluster) and one (or more) cluster(s) for each type of anomaly present (the red cluster and the blue cluster).

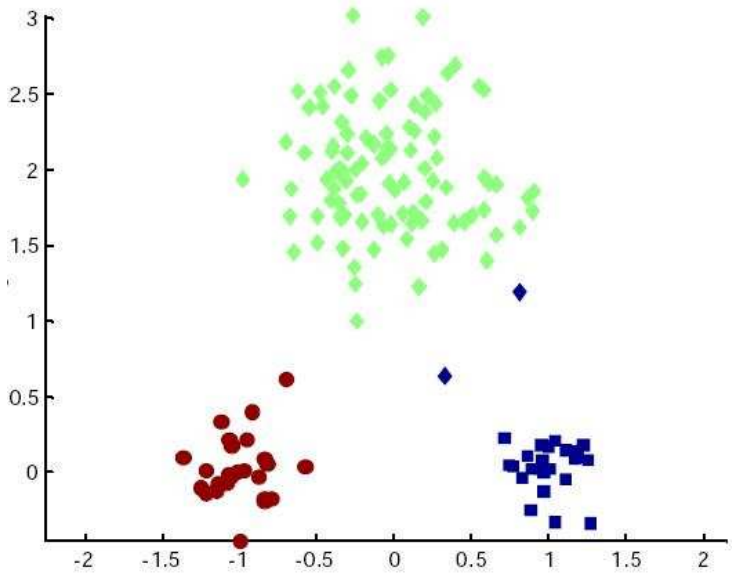

Figure 4: Model with clusters

In the other representation, there is one or more models for the normal traffic and any points that do not fit the model(s) are considered outliers and thus, part of the anomalous traffic. In Figure 6, there is only one model (the red Gaussian curve) that represents a single class of normal traffic. Any point located too far from the model is anomalous. In Figure 5, there is(are) several cluster(s) (in our example below, only one) for the normal traffic (here the green cluster) and each outlier represents an anomaly (the black point and the brown point).

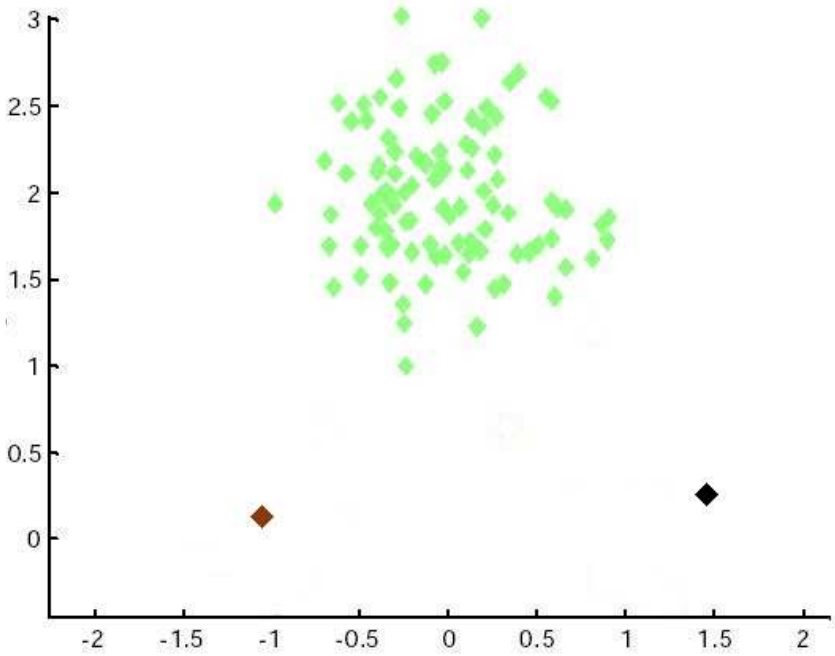

Figure 5: Model with one cluster and several outliers

At this point of our experimentation, we have no indication about actual representation of the anomalies. Nevertheless, we think that, since anomalies are usually composed of many packets, the second representation (as illustrated by Figure 5) should not be used and we should consider only the first representation (illustrated by Figure 4).

2. Choice between the different forms of unsupervised learning

In the next paragraphs, we address the problem of the choice of the unsupervised techniques. The objective is to build a technique that can identify several classes of traffic and keep some understandable attributes. We will only talk about dimensional reduction, density estimation and clustering since they are the three most represented techniques in the literature and the one presented during the several talks about machine learning inside ECODE consortium.

- ***Dimensional reduction***: the principle of dimensional reduction is to project the data from a set of great number of dimensions to a set of smaller dimension. In our case, it means that we would end up with a set of variables that would have no physical/concrete meaning. One of our goal being to keep some understandable attributes in order to have easy to understand and meaningful rules (i.e. anomaly characteristics), the dimensional reduction is in clear contradiction with our requirements.

- ***Density estimation***: efficient technique to detect outlier if the normal traffic is a single class (and not composed of several classes) and that anomalies are only outliers (and not one or more class(es)). As depicted in Figure 6, the single class considered could be the one of the normal traffic. Anomalies would be represented as outliers (points that are too far from the model). Interpreting Figure 7 we could have for example, two classes for the normal traffic (green and red curves) and the anomalies would be the outliers. We could also have one class for the normal traffic (green curve) and one class for the anomalous traffic (red curve). In both of these cases, density estimation would be ineffective since it is able to consider several classes. Therefore, this form of unsupervised learning seems to not be adapted to our case because we cannot guarantee that we will be in the case of Figure 6.

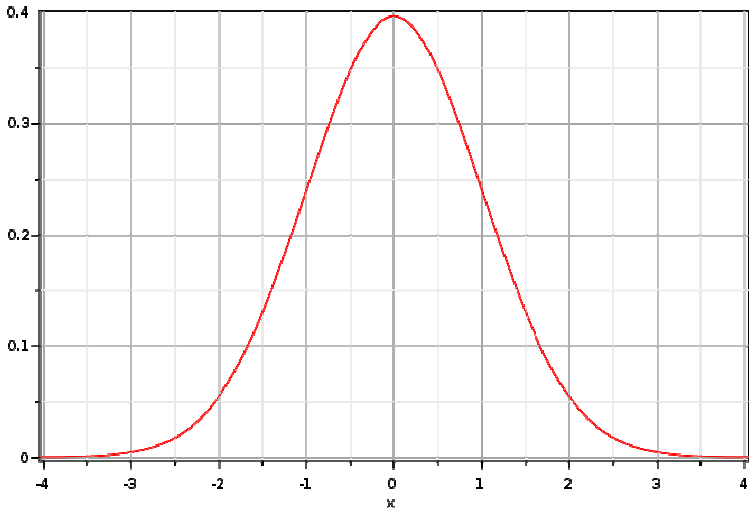

Figure 6: One class system

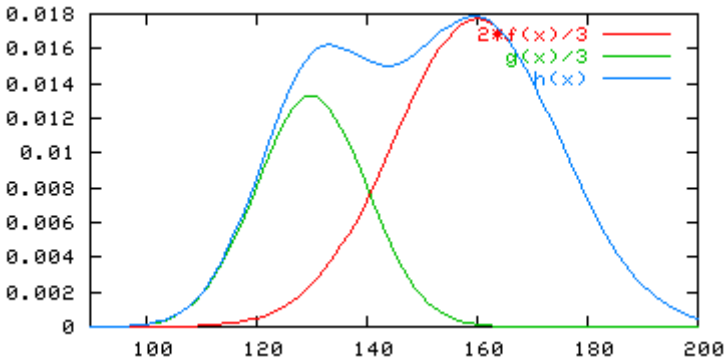

Figure 7: Two classes system

- ***Clustering***: cluster analysis or clustering is the assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar under some chosen criteria. Clustering does not have any of the limitations listed above: it keeps all the attributes in a clear and intelligible form and it can consider and analyze them without any limitation on the number of classes (in our case, the number of classes of traffic including both normal and anomalous ones).

Based on this initial analysis of potential techniques, we selected the clustering technique as it appears to be the most adapted and promising form of unsupervised learning to address our problem.

3. Anomaly identification

However, many problems linked to the use of unsupervised learning occur. The first of these problems is the identification of the clusters. In fact, the system is unable to determine whether a cluster represents the "normal" behaviour of the network or an anomaly.

In order to cope with this problem, three solutions are available. The first solution is to manually identify whether each cluster corresponds to normal or anomalous behaviour. The second solution would be to perform a two-step processing: a first one to identify the cluster(s) which represent(s) the "normal" behaviour by comparing it to reference network traffic without anomaly (we may use supervised learning in this case) and the second step would consist in manually identifying the cluster(s) which correspond(s) to anomaly(ies). The last solution would be possible under the hypothesis that anomalies are less important than normal behaviour in term of surface or volume. As we haven't run any tests since so far, we cannot make any assumption on which would be more suited to our needs.

4. Discovering new anomalies with machine learning

One aspect that differentiates unsupervised learning from supervised and semi-supervised learning is that it theoretically allows to discover new anomalies through the detection of new clusters (or new outliers, depending on the anomaly representation selected). However, discovery of anomalies seems to be correlated to the choice of the used attributes or fields of the packets headers. Indeed, by inspecting table 1.2, one can notice that every rule is using different attributes. This implies that if we try to find these new rules from scratch, we will need to search for new previously unused attributes/fields. Therefore, the discovery of new types of anomaly seems to be heavily linked to the discovery of new pertinent attributes/fields. This search seems to be the most important point in the discovery of new anomaly through unsupervised learning.

The basic method that we intend to use to find new rules is to create new attributes, and then, systematically try to find clusters inside the captured data to assess the presence of a new rule built upon the newly created attributes. The problem of creating new pertinent rules can then be split into two tasks: first, create new attributes, second, create pertinent rules.

5. Create new attributes

- Attribute generation: Table 1 is actually composed of attributes that are built from packet header fields. We plan to imitate this technique automatically and create attributes through a simple calculation (count, division, mean, distributions, etc...) of the packets fields (IP address, TCP/UDP ports source/destination, etc.). However it is obvious that such a variety of possible calculations applied to a large number of packet header fields will generate a huge amount of possible combinations. The next issue will be to eliminate the attributes that seem to be of less interest.

- Attribute interest assessment: We want to assess whether a generated attribute contains enough information. In fact, if we want to extract clusters from the data spaces, we will need attributes that have a quantity of information as large as possible. The evaluation of the quantity of information may be computed through entropy or another index. We have already started to investigate several indices and we plan to compare them.

6. Create new rules

- Rule generation: The rule generation will simply consist of choosing several dimensions (or attributes). We intend to sweep through all the attributes previously generated and generate every possible rule. Once the new rule is created, we generate the cluster(s) in the chosen dimensions. The next step is then to assess the interest of the new rule.

- Rule interest evaluation: As explained in the previous paragraph, a new rule to evaluate is actually a new set of dimensions where we'll try to find pertinent clusters. The key component to evaluate the quality of a newly built rule is the ability to measure the quality of the cluster(s) found in the dimension(s) of the new rule. In fact, clustering methods allow us to find some clusters but if we lack a metric to assess the reliability or the pertinence of the clustering, the process would be useless.

  Note: Kave Salamatian talked in his presentation called An Introduction to non-Supervised Learning Techniques [Salamatian 2008] about a criterion called Normalized Mutual Information that might be useful to evaluate the clustering quality. We are currently looking for this kind of method.

3.3.5 Conclusion

In a nutshell, creating new rules will consist in creating new spaces through the association of new attributes built from previously unused fields from the packets headers.
The most critical part in this proposal is that the number of attributes/rules/clusters to process will be huge. As part of the first step, reducing this number implies to eliminate useless attributes through the use of several indices to evaluate their interest. We plan to try to find other way to reduce the amount of calculation to do.

We think that by having an efficient and fast search of attributes/rules, our system will be really efficient and that we will be able to find new anomalies.

3.3.6 Distributed detection of anomalies

The class of parametric statistical anomaly detectors needs to implement three phases:

1. A modelling phase that consists of capturing essential correlation structure of the state vector to monitor. This phase has a strong machine learning flavour and generally uses statistical model calibration techniques as Expectation Minimization to implement Maximum Likelihood fitting or Principal Component Analysis to derive approximate low dimensional models.
2. A filtering phase that consists of using the correlation structure model obtained in the first phase in order to reduce the entropy of the observation by removing the information's irrelevant to anomaly detection. The approach assumes that everything that goes along with the correlation structure obtained in the first phase is irrelevant to anomaly detection.
3. A decision phase that takes the signal filtered in the second phase and applies to it a statistical test that will decide if there was an anomaly somewhere in the network or not.

These three phases are now well investigated in the centralized case where all observations are available at a single point. However, moving from a centralized setting to a distributed setting -our ambition in this project- necessitates implementing the two above defined phases in a distributed way. Our machine learning approach consisted of developing distributed machine learning techniques to deal with distributed modelling. The approach followed consisted of developing a network wide state-sharing scheme that enables each node to obtain an approximation with a controllable precision of other nodes states. This approximation is obtained through a distributed optimization that could be seen as a distributed Principal Component Analysis; a technique widely used in centralized machine learning. This technique was chosen because of the nice and rich theoretical framework that exists around it.

4. Implementation

4.1 Case a1: Adaptive traffic sampling and management

Our platform consists of three main components: the traffic emulation service, the traffic monitoring and the sampling service, and the flow data collection and analysis service. Figure 8 depicts the interactions between the three main components.

The traffic emulation service is an advanced emulation environment we implement to evaluate the adaptive monitoring and sampling solution we are proposing. The tool is general enough that other partners could twin it if needed and use it for the evaluation of their own solutions.

The traffic monitoring and sampling service as well as the data collection and analysis service are part of the global ECODE architecture. We develop these services in such a way they could either be connected to the emulation service or directly to a real network interface. Hence, the proposed modules can be easily exported and tested in a real environment.

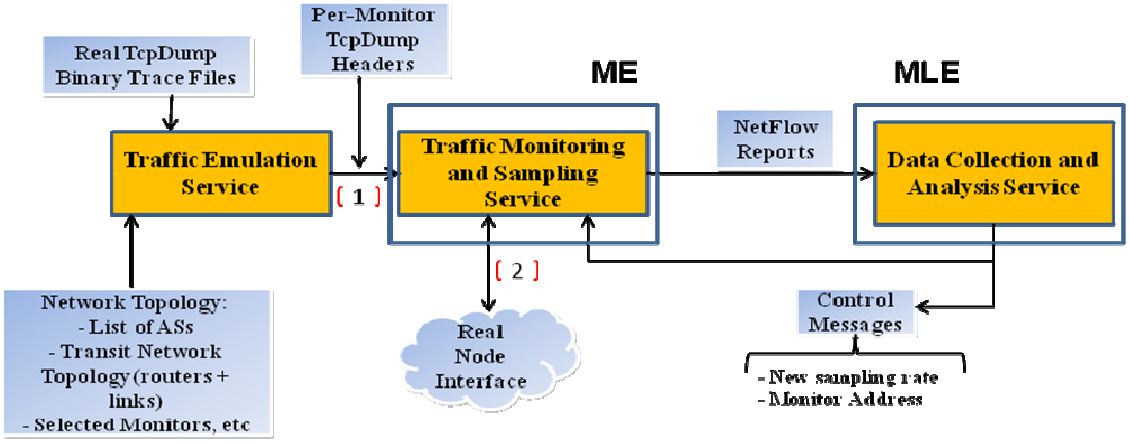

Figure 8: Global Architecture

A common usage scenario of our platform would include the following steps: ***(i)*** The user starts by supplying the traffic emulation service with two XML files describing respectively the list of traces to play (the path) and the network topology (AS(s), transit networks, routers, monitors, links,). ***(ii)*** Once done, the user can run the three services either in the same machine or in three different machines towards achieving better performance and scalability. Running the traffic emulation service will result in emulating the described topology within the same machine, dispatching the set of packet-level traces and playing them over the topology. Running the traffic monitoring and sampling service will result on monitoring routers selected as monitors by the user, constructing the 5-tuples flows starting from the sampled packets, and forwarding the flows reports to the data collection and analysis service. ***(iii)*** Within the last service, the user can run advanced traffic analysis algorithms. In our case, we use the adaptive traffic sampling and management scheme described in Paragraph **3.1**. We are able to run adaptive algorithms from within the Data Collection and Analysis service thanks to the API we provide for remote online controlling the monitors.

In the following paragraphs, we detail each of these services.

4.1.1 The Traffic Emulation Service

Many proposed monitoring solutions by the network research community deals mainly with traffic monitoring at the flow level. The best approach to evaluate such solutions would be the presence of real network wide traffic where all packet headers are available in addition to time stamps and location information that indicate when and where each packet was seen. Unfortunately, getting these network-wide traces is rather difficult if

not impossible (since packet traces are confidential). Even when available, these traces prevent testing different topologies and different IP prefix allocations. However, one can easily find large TcpDump traces captured on some high speed link in a backbone transit network. The traces of the MAWI working group are a good example. One of the main contributions of our platform is the traffic emulation service that splits these TcpDump traces and plays them over some well defined network topology to reproduce the network wide packet-level traces users are missing.

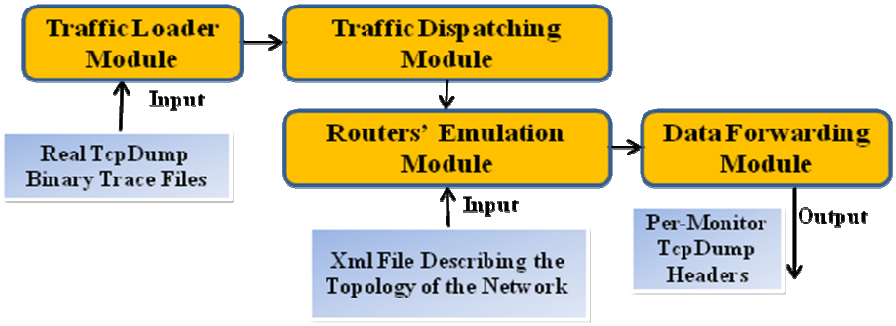

Figure 9: Architecture of the Traffic Emulation Service

The prefixes dispatching process requires to have a-priori idea about the importance of the traffic resulting from each autonomous system connected to the backbone network with respect to the whole network traffic. To this end, we propose for our platform users a dispatching module that works at the IP prefix level. As the traffic loader module (described in Figure 9) reads packets from the TcpDump trace using the Pcap library, the dispatching module (described in Figure 9) associates them online to the right autonomous system based in one hand on the list of weights the user attributes to the different ASes and in another hand on the prefix length specified by the user. The weights should be described in the XML configuration file by the user as well as the network topology. For this, we propose a highly flexible configuration methodology via XML files through which users can describe their emulated network topology. For clarity, simplicity and future extensibility, we use XML structuring capabilities and we represent the whole network components in terms of hierarchical XML elements and attributes. Figure 10 depicts an example of a simple topology that users can describe via XML configurations files. Users are able to describe the list of autonomous systems and the topology of the transit network that interconnects them. Within the transit network, it is possible to describe the list of routers (Interfaces, IPs, etc), the list of monitors as well as the set of links interconnecting them including the characteristics of these links. In general, we provide as much details as it is needed to re-produce the closest possible topology to the real one. A detailed XML schema of our configuration files is available at http://planete.inria.fr/GEANT.

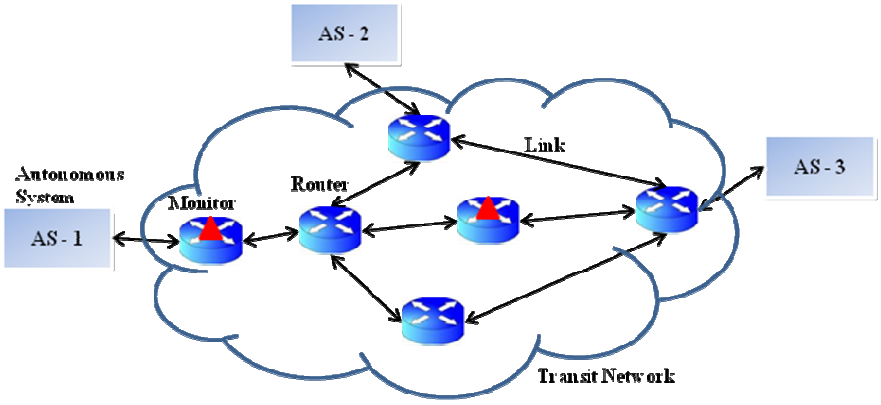

Figure 10: Example of an Emulated Topology

The default dispatching method we are implementing is the **weighted random**. Suppose we have 23 ASes to which we associate different weights such that these weights divide the interval [0, 100] into 23 consecutive bins, where each bin represents the weight of an AS. Suppose that the user chooses the prefix length to dispatch as being the /16. Then, each time a new prefix of the same length appears while reading the raw trace, a random number is selected within the interval [0, 100] using a uniform distribution. This selected number points then to the AS to which the new prefix is to be associated. All subsequent packets having the same prefix as source or destination are associated to the same AS2.

We also put emphasis on the extensibility of the traffic emulation service. Indeed, adding new traffic control and monitoring methods should be possible without major changes to the traffic emulation service design. For this purpose, we adopt an object oriented design and development methodology that ensures a modular architecture. Users are able to easily develop and plug different packet filtering policies or even distributed sampling protocols. In addition to the traffic emulation features proposed via this service, users interested in studying the performance of different routing algorithms can plug their own routing module providing that they maintain the same programming interface as the default one. For the moment, our routing module only support shortest path routing protocol based on the Dijkstra algorithm. Routes are static and can be set up via the XML configuration file. Users can also use our platform to deploy and study traffic engineering algorithms.

4.1.2 The Traffic Monitoring and Sampling Service

As shown in the Figure 8, the traffic monitoring and sampling service belongs to the Monitoring Engine (**ME**) and more specifically to the set of passive monitoring points (**passive MP**).

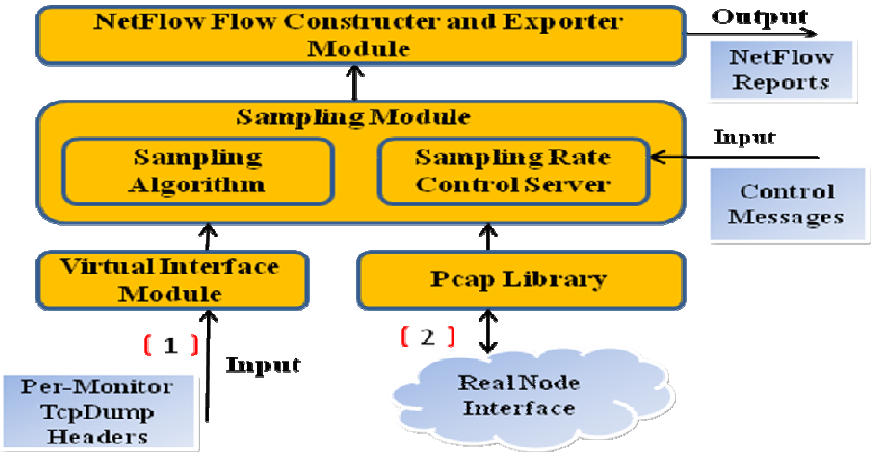

Figure 11: Architecture of the Traffic Monitoring and Sampling Service

We design and develop the traffic monitoring and sampling component as an extension of Softflowd (http://www.mindrot.org/projects/). Indeed, Softflowd is a flow based network traffic analyzer capable of Cisco NetFlow data export. It fully tracks 5-tuples traffic flows by listening on a network interface. These flows are then reported via NetFlow reports to a collecting host. Note that Softflowd does not include support for packet sampling neither fixed nor adaptive. Our extensions to Softflowd add this sampling functionality, which is essential for monitoring scalability. They also allow the integration of this tool over emulated nodes within our traffic emulation service, while being able to run over real routers by sniffing packets directly on their interfaces. Indeed, as shown in Figure 11, the traffic monitoring and sampling service could either work with TcpDump packets headers received from an emulated monitor within the traffic emulation service via the Virtual Interface Module described in Figure 11 of work directly on a real node interface. In the latter case, it captures traffic promiscuously using the Pcap library as described in Figure 11. Note that it is likely that the sampling module will place additional load on hosts or gateways on which it runs. Our implementation has been designed to minimize this load as much as possible. Indeed, in order to decide either to retain the packet being read or to reject it, the sampling module makes a decision each time the Pcap library returns a handle to a new packet and before the packet is being loaded to the memory. If the result of the sampling algorithm is to

capture the packet, the entire packet is then loaded and the maintained flow list is updated via the NetFlow flow constructer module, otherwise the packet is simply discarded.

As described in Figure 11, the sampling module encloses a sampling algorithm as a core and a sampling rate control server. The default sampling algorithm we are providing is the following: if a user chooses a sampling rate of **A/B** (**A** packets among **B** packets, **A <= B**, **B > 0**, **A >= 0**), then every B packets, the sampling module generates randomly a set **S** of **A** numbers within the interval **[1, B]**. Packets with numbers outside the set **S** are rejected and only the remaining packets are considered for 5-tuple flow construction. Generated flows are then encapsulated within NetFlow reports and are exported via the NetFlow flows constructer and exporter module. The sampling rate control server enables users to change remotely the sampling rate of a given monitor whether it is in a real network or downstream the traffic emulation service. The remote monitor controller, which we will describe later, proceeds to change the local sampling rate to each monitor. This remote control functionality allows users to control and change online the sampling rate of one or multiple monitors, or simply offline from one experiment to the other.

4.1.3 The Data Collection and Analysis Service

As shown in Figure 8, the Data Collection and analysis service belongs to the **Machine Learning Engine (MLE)** and more specifically to the **Processing** bloc.

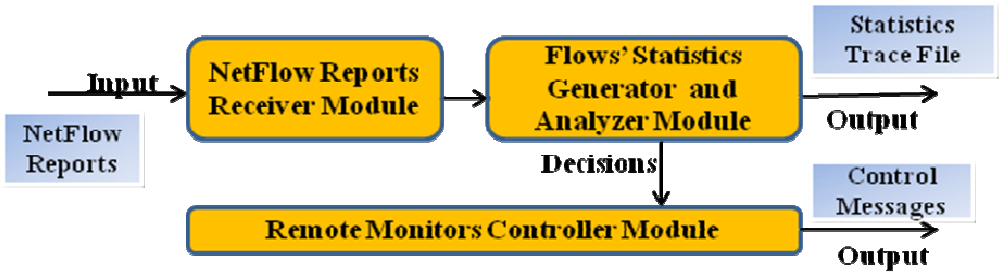

Figure 12: Architecture of the Data Collection and Analysis Service

We design and develop the data collection and analysis service starting from the functionalities proposed by Flowd (http://www.mindrot.org/projects/). Flowd is a secure NetFlow collector. It works with any standard NetFlow exporter, including hardware devices or software tracking agents. As described in Figure 12, we enclose the Flowd capabilities within the NetFlow reports receiver module around which we develop other modules namely the flow statistics generator and analyzer and the remote monitor controller modules.

For the moment, we provide the adaptive sampling and management schema described in Section **3.1** within the Flows' Statistics Generator and Analyzer Module. However, depending on user needs, one can easily adapt this module. For example, one can implement anomaly detection based on the collected NetFlow reports or introduce quality of service algorithms that can improve traffic routing as a function of the monitored network status. Independently of the data analysis process, the user can decide to change the sampling rate of one or more monitors to improve the accuracy of the algorithms implemented at the analyzer, either offline or online. For this purpose, the remote monitor controller shall be used. This controller is a shared module that we propose to plug and use within the data collection and analysis service but it can also be used separately either as a tool or with another software. The remote monitor controller provides an API to send control messages to a monitor specific control server, which modifies the sampling rate of the given monitor.

4.2 Case a2: Global monitoring

4.2.1 Specification

4.2.1.1 Functions

Our passive monitoring system is supposed to have two main capabilities: the capture of packets and the framework to analyze of the captured packet. Concerning the capture, we want to be able to launch a capture

on any interface on-the-fly. The analysis framework need to offer two facilities: launch any analysis on the fly and export the result of these analyses through logging, reporting or trace generation. The analysis framework also needs to be able to launch, stop and manage several analysis sessions independently.

4.2.1.2 Technical solutions

In order to improve the flexibility of our tool, we choose to use a multi-thread architecture (see Figure 13). The passive measurements are done though the libpcap library. We did this choice to be able to benefit from the flexibility and portability of the libpcap library. In fact, the use of libpcap is wide and its cost is null. We dedicate a thread for each interface for the capture. All the captured data from the different thread is then merged inside an interface which run inside a thread (cf. Figure 13). In order to be able to manage several analysis sessions in parallel, the processing is performed through several pipelines. A thread is used for each pipeline. The thread-interface then manages the forwarding of the captured data to the different process pipelines.

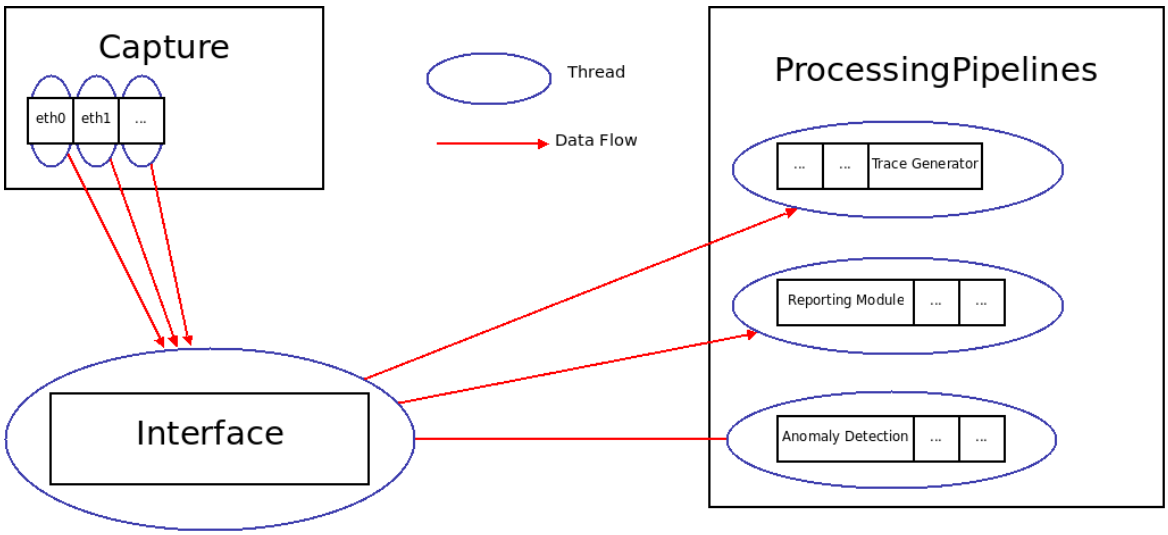

Figure 13: Specification of the general architecture of the monitoring system

Each process pipeline is designed to be composed of several analysis module and one output module. Each analysis module takes some data in input, and gives some in output. Output modules are supposed to export the result from the last analysis module in the chain through trace generation, reporting, logging, etc.

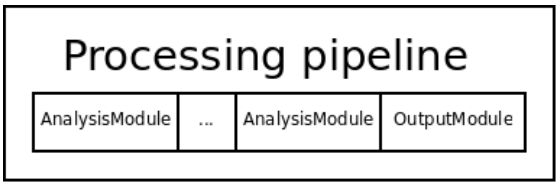

Figure 14: Specification of the processing pipeline

4.2.2 Global architecture

4.2.2.1 Class diagram

MonitorDaemon is the base class of our system. It manages Interface and CaptureManager. Interface then manages PipelineManager. This structure respects the principles explained in the paragraph Technical Solutions and the scheme present in it. MonitorDaemon manages the captures through the class CaptureManager, and the centralization of data and the process pipelines through the class Interface.

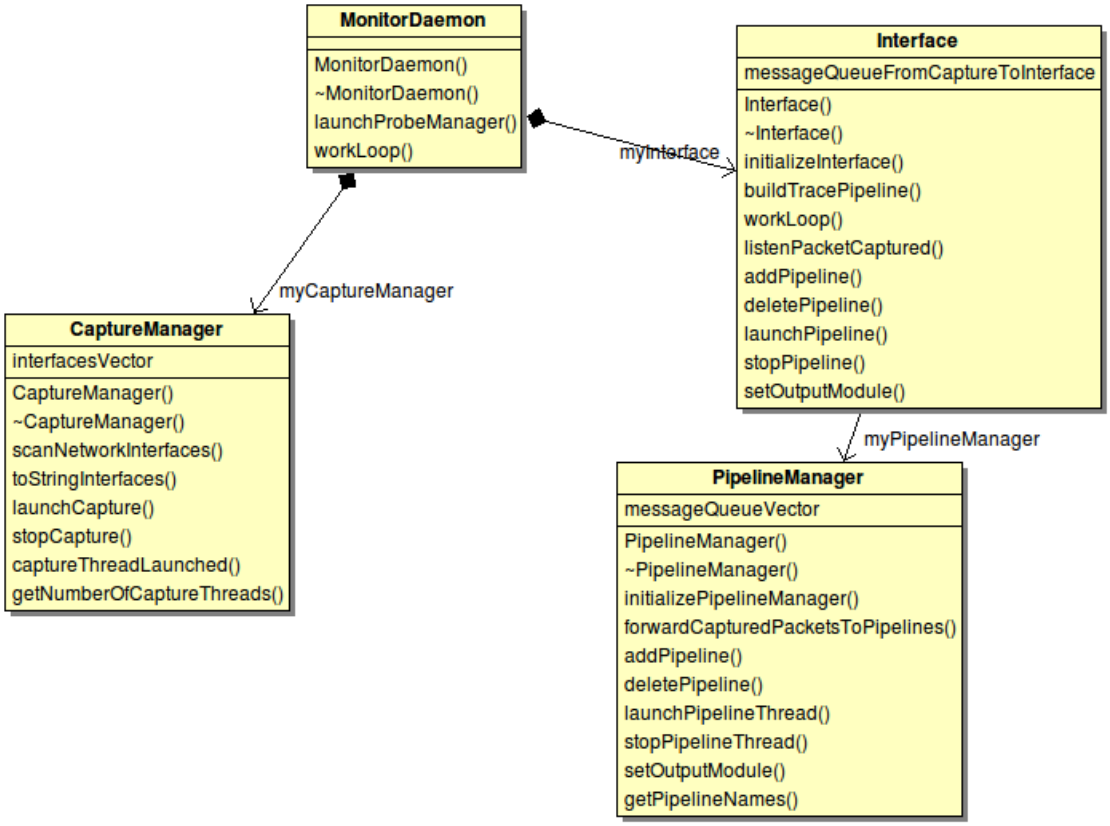

Figure 15: Monitoring system class diagram

4.2.3 Passive measurements

4.2.3.1 Class diagrams

BaseThread implements the thread library from the Boost library. We choose to use this library for there is no default implementation of thread in C++ and because this library is free and quite well documented. Since CaptureController inherits from BaseThread, each instance of CaptureController can be a thread.

CaptureManager then manages all the instances of CaptureController (i.e. the state of the threads and the parameters).

CaptureManager

This class aims at managing all captures including the capture parameters (e.g., size of headers to capture, etc...) and the thread management.
The data stored inside the class is the following: a vector which contains all the name of the available interfaces (interfacesVector) and a vector which contains all the launched capture (captureControllerVector).
The methods of this class include:

- scanNetworkInterfaces(): scan network interfaces on the system and store the result in interfacesVector.
- launchCapture(): launch a capture and store it in captureControllerVector
- stopCapture(): stop a capture in captureControllerVector and remove it from captureControllerVector.

CaptureController

The function of this class aims at managing the capture of packet on a single interface (e.g. eth0, wlan0, etc.).

The methods of this class are:

- initializeCaptureController(): initialize the capture with the pcap primitives and prepare the forwarding through a message queue.
- workLoop(): threaded method executed.
- parsePacket(): pre-analysis of the packet and forwarding if the packet fulfills the criteria (IPv4).
- sendHeaders(): send data to the centralization point.

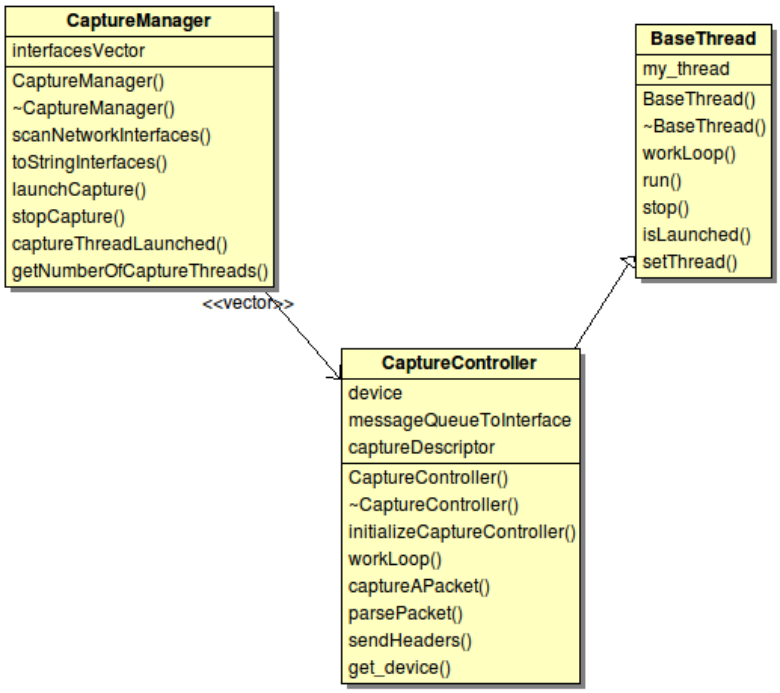

Figure 16: Passive measurement system class diagram

4.2.3.2 Sequence diagrams

The timeline in UML is from the top to the bottom. Each arrow is method call. Each vertical line represents a class and its own life line. These diagrams illustrate the different method call available in the class MonitorDaemon and what this produces inside the other classes.

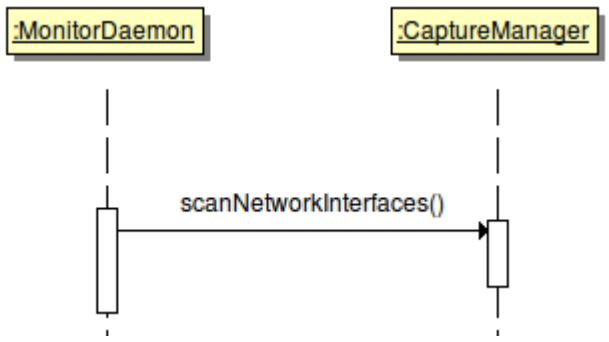

Figure 17: Sequence Diagram 1 for the MonitorDaemon class

LaunchCapture creates the CaptureController, launches the thread and then adds it to the data stored inside the class.

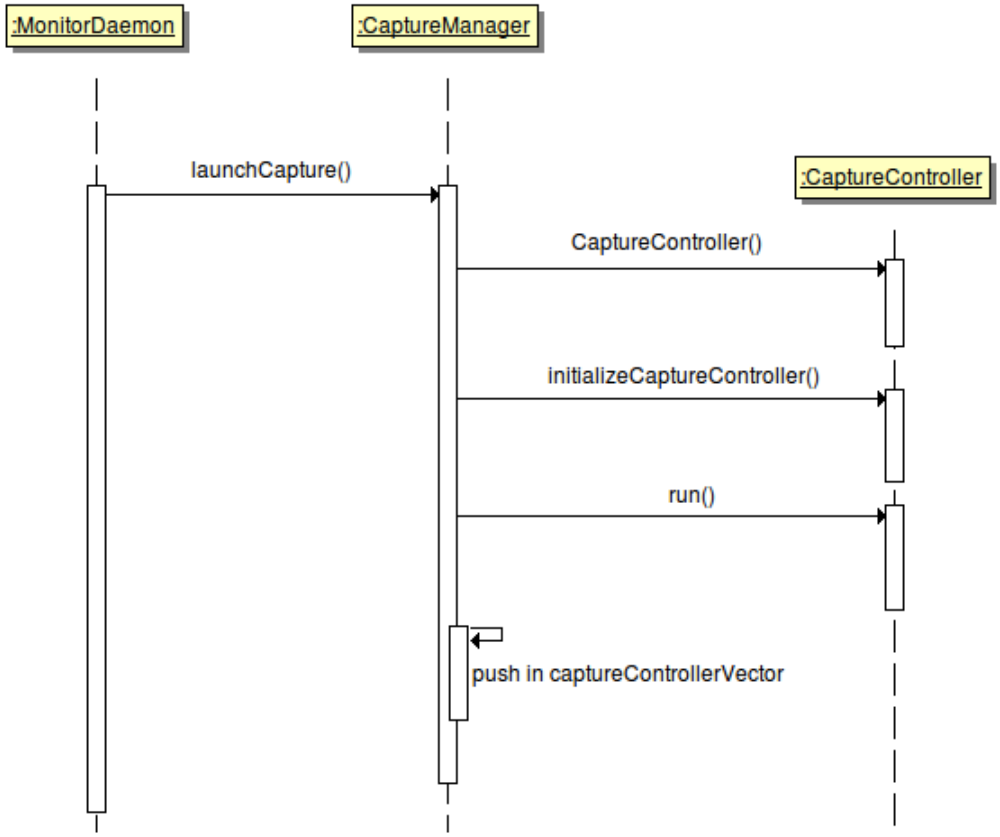

Figure 18: Sequence Diagram 2 for the MonitorDaemon class

StopCapture stops the thread, destroys the instance of CaptureController and then removes it from the data stored inside the class.

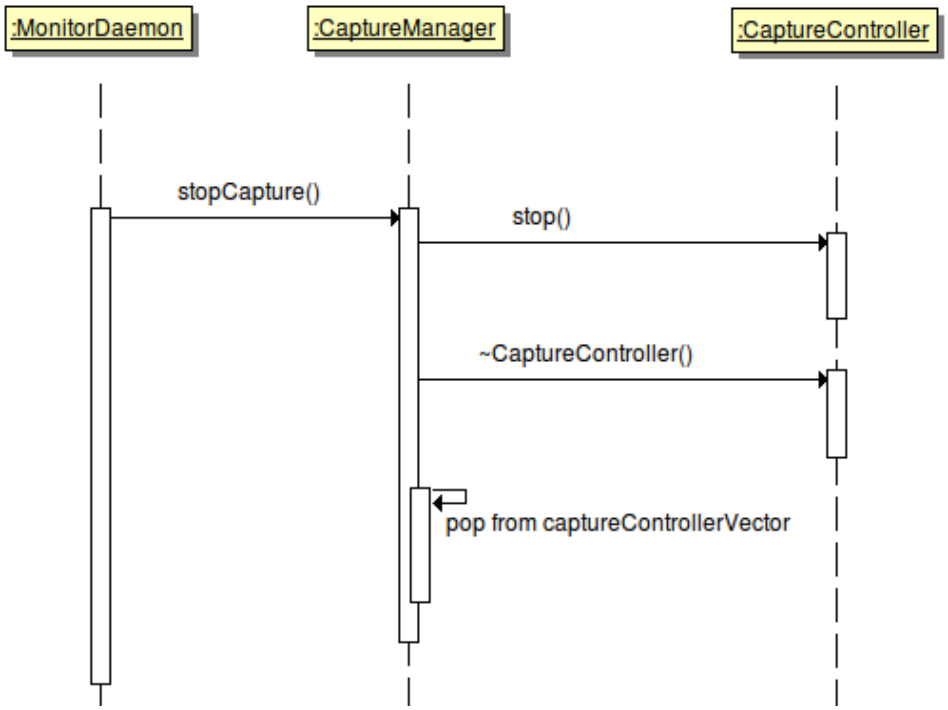

Figure 19: Sequence Diagram 2 for the MonitorDaemon class

4.2.4 Traffic processing

4.2.4.1 Class Diagrams

Since ProcessPipeline inherits from BaseThread, each instance of ProcessPipeline can be a thread.

PipelineManager then manages all the instances of ProcessPipeline (i.e. the state of the threads and the related parameters).

ProcessPipeline also stores all the analysis modules and the output module.

The classes that will actually implement the traffic analysis and the export of results will have to respectively inherit from the classes AnalysisModule and OutputModule. In fact, we choose to implement the control functions on the data flow through the pipeline inside these two classes and ProcessPipeline. This leaves the programmer with less work to do when he tries to implements an analysis algorithm. E.g.: The class TraceBuilder which creates a trace in the pcap format inherits from OutputModule.

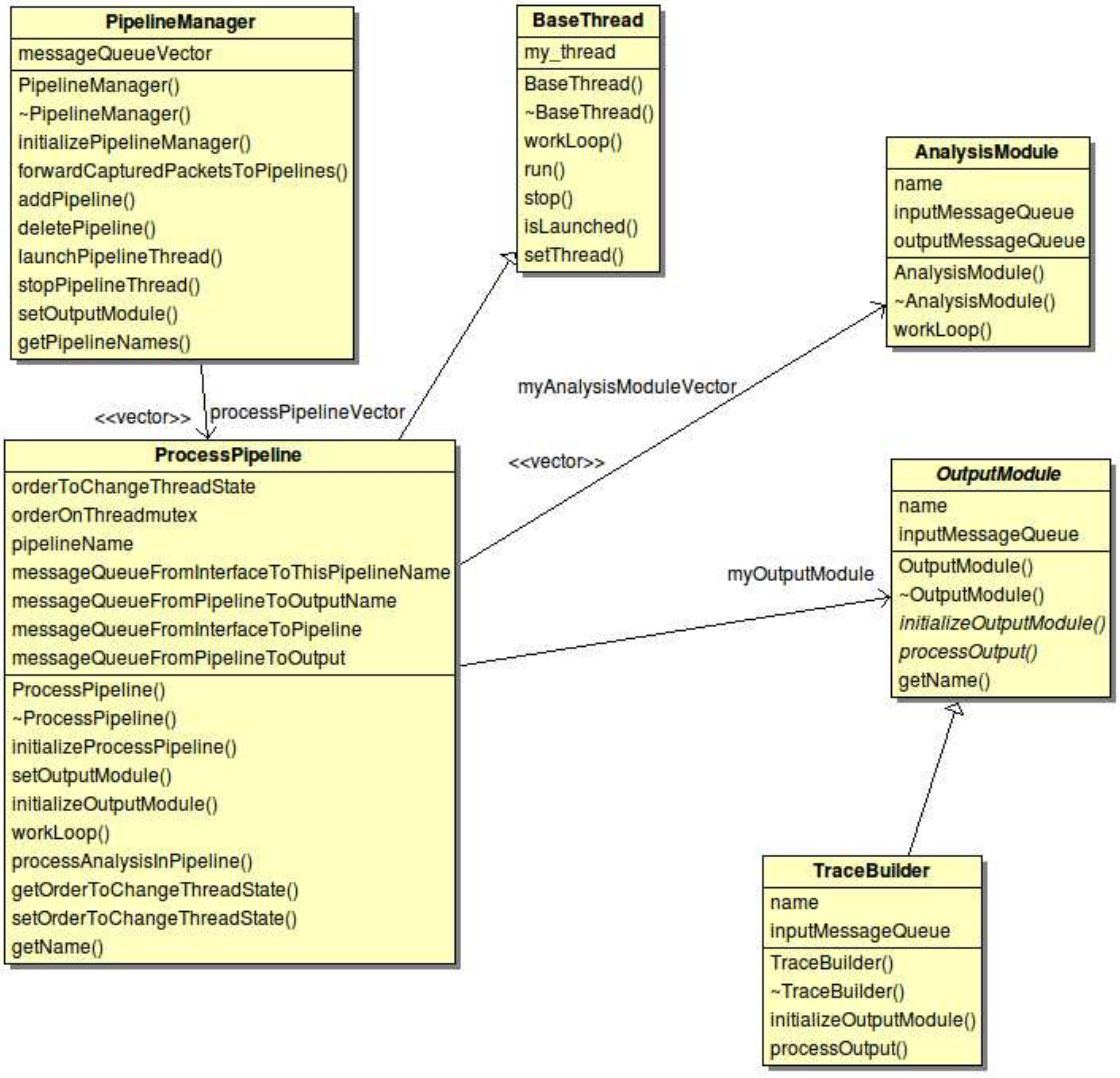

Figure 20. Organization of classes for the Pipeline process

PipelineManager

This class aims at managing all the processing pipelines.

The methods of this class include:

- forwardCapturedPacketsToPipelines(): forward all the centralized packets to all the processing pipelines.
- addPipeline() / voiddeletePipeline(): add/delete a pipeline.
- launchPipelineThread() / stopPipelineThread(): start/stop the associated thread.
- setOutputModule(): set the output module of the process pipeline.

ProcessPipeline

This class aims at reading the data to be processed and at managing the processing of the data by the pipeline. To manage the processing of the data trough the pipeline, ProcessPipeline put the data through each AnalysisModule (if present) and then send it to the OutputModule (if present).

The data stored inside the class includes a vector which contains all the analysis modules to execute (myAnalysisModuleVector) and an output module (myOutputModule).

The methods of this class are:

- initializeProcessPipeline() : initialize the processing through building the dedicated message queue.
- workLoop() : threaded method executed.
- processAnalysisInPipeline() : process the input data through the AnalysisModule(s) and the OutputModule.

TraceBuilder

This class aims at building a pcap trace from the captured packets.

The only data stored in this class is a pointer to the trace.

The method processOutput implements the inherited method from OutputModule which writes packets to the tracefile through the pcap primitives.

4.3 Case a3: Cooperative distributed anomaly detection

4.3.1 Attribute based local detection of anomalies

4.3.1.1 Specifications

Our system will follow the principle introduced in Section 2. For this purpose, we have implemented a two-stage Anomaly Detection System (ADS). The first step aims at detecting any suspicious traffic in order to have no false negative (at the expense of the number of false positives). The second step aims at using a classification mechanism in order to eliminate the false positives and just keep the true positives.

4.3.1.2 Global architecture

The system will be implemented as an analysis module inside our traffic analysis framework presented in the implementation section of the Use Case a2 (see Section 4.2). For the final online version, we plan to have two threads running.

- The first one is the one from the process pipeline. It is launched and stopped through the traffic analysis framework.
- The second one performs the anomaly detection. It is launched when the analysis module (inside the traffic analysis framework) is created and stopped when is it is destroyed.

The code running in the process pipeline (cf. Use Case a2) simply stores the packet captured according to the specified granularity inside what we call a set of packets. Every time a time slot is finished, the process pipeline forwards the set of packets to the ADS inside the second thread. A system of buffer containing sets of packets

is implemented inside the ADS in order to cope with the fluctuation of the amount of data to process. The ADS processes the sets of packets in a FIFO order.

4.3.1.3 Algorithm structure

1. Pseudo-algorithm

This algorithm is executed on each set forwarded by the process pipeline. The algorithm follows the two step architecture presented in the specifications (see section 4.3.2.1.). First, we analyze the network and detect the anomalous slots through the functions Process_Network_Data and Find_Anomalous_Slots. Then, we only keep the slots at the network level /24 and we classify them through the function Classify.

```
function CLASSIFY (address)
GENERATE_MATRICES (address)
search matrices
classify and identify

function NADA_ANALYSIS ()
PROCESS_NETWORK_DATA ()
FIND_ANOMALOUS_SLOTS ()
FOR each anomalous slot found in address at level /24
CLASSIFY (ADDRESS)
ENDFOR

main ()
Algorithm:
NADA_ANALYSIS()
```

Table 3. Pseudo-Code for the new NADA in offline mode

2. Network data process

Once the set of packets is received, all the destination networks that appear in that set of packets are extracted in three prefix lengths: 8, 16 and 24 by the function Process_Network_Data. For now, this restriction in terms of prefix lengths limits the analysis of our system to three network levels. Extending this to any level is easy, but performance and memory consumption will suffer greatly.

The result of this extraction is a three levels linked-list, where an element of this list represents an address and also contains a pointer to the next level (this is actually represented as next byte). Like this, the destination address 140.93.4.4 would result in the node 140, with next byte pointing to the /16 address prefix 140.93, which in turn would have a next byte pointing to /24 address ,prefix 140.93.4. Having the whole list of addresses in an easy to find manner makes it possible to calculate all the time series for all the addresses in a single run through the forwarded packets.

The function Process_Network_Data also calculates the mean, the variance and the standard deviation of all the metrics for all the addresses extracted at the current level. It also saves the different metrics used for the detection (#packets, #bytes, #SYN) for the current time slot (or set of packets) for each of the networks (so when applying the formulas we have all the necessary information).

3. Anomaly detection

The algorithm then searches for anomalies through the function Find_Anomalies. It sweeps through all the found addresses at level /24. When a network is found anomalous for any of the parameters, the network is added to a list of anomalous networks and the algorithm tries to classify the anomaly. To do so, the algorithm launches the function Process_Host_Data passing the extracted network as parameter.

4. Anomaly classification

The following paragraph deals with the classification inside the function Process_Host_Data. We use hash tables and several linked list to store the data needed for the processing. There are 7 hash tables (for the mappings between key, i.e. source IP or source IP/source port, and id, and the opposite).

The function Apply_Formula processes all data inside the network list to generate the matrices.

The function Classify is then called to apply the previously established rules of classification to the detected anomalies in order to eliminate the false positives among all the anomalous alerts. These rules are built through the use of several attributes with some conditions over their values. The attributes used in each rule and the related thresholds have been chosen through expert knowledge.

At the end, we obtain a list of anomalies with all their characteristics: source (port/IP), destination (port/IP), type of attack, etc.

4.3.1.4 Data structure

In order to store the data extracted from the set of packets, we use different data structures. We will here expose the main structures. We will first expose the different structures generated by Process_Network_Data to store the addresses from the sets of packets. The second part of this section will address the data generated by Find_Anomalies to describe the anomalies found.

1. List of addresses

The struct address_properties is used to store the statistics for a network/mask for the three metrics (#packets, #bytes and #SYN) used for the detection step of our ADS. There are three instances of this structure in the structure address_list.

```
struct address_properties {
        int mean;
        uint64_t variance;
        uint32_t curr;
        uint32_t ant;
};
typedef struct address_properties address_properties_t;
```

Here is a detail of all these variables:

int mean:	mean for this particular variable over all slots
uint64_t variance:	variance for this particular variable over all slots
uint32_t curr:	temp variable for calculating P (this would be Xi+1)
uint32_t ant:	temp variable for calculating P (this would be Xi)

This structure is used inside address_list to store the address and the mask for the network.

```
typedef struct{
        struct in_addr ip;
        uint32_t mask;
} address_t;
```

This structure is used to store the IP address from the sets of captured packets.

```
struct address_list {
        address_t address;
        int *deviation;
        unsigned int *pktcount;      // per slot
        struct address_properties *properties;
        struct anomaly_list *anomalies;
        struct address_list *next_octect;
        struct address_list *parent;
        struct address_list *next;
};
typedef struct address_list address_list_t;
```

Here is a detail of all these variables:

address_t address: address of the current address (including mask cf. address_t).
int *deviation: holds the original threshold (sigma*K) for each variable (PKT, SYN, BYTES).
unsigned int *pktcount:
struct address_properties *properties: statistics of the current address
struct anomaly_list *anomalies: list of anomalies in this address
struct address_list *next_octect: pointer to the next level in IP network (e.g. we are at 140.93.0.0 and this points on 140.93.4.0).
struct address_list *parent: pointer to the previous level in IP network (e.g. we are at 140.93.0.0 and this points on 140.0.0.0).
struct address_list *next: pointer to the next IP network inside the same IP level (e.g. we are at 140.93.0.0 and this points on 140.94.0.0).

2. List of anomalies

This structure is used to store the anomalies for a specific network. All the tables (find, grows, impact, impactlevel, duration, decrease, children) store the different value for the metrics of the anomaly. The value NVAR counts the number of metrics used during the detection, i.e. NVAR = 3 (#packets, #bytes and #SYN).

```
struct anomaly_list {
        slot_t *slot;
        unsigned pktcount[2];
        int find[NVAR];
        int grows[NVAR];
        int strongestvar;
        float impact[NVAR];
        int impactlevel[NVAR];
        int duration[NVAR];
        float decrease[NVAR];
        int children[NVAR];
        int destcount;
        struct responsible_list *dest;
        struct anomaly_list *next;
};
typedef struct anomaly_list anomaly_list_t;
```

Here is a detail of all these variables:

slot_t *slot: pointer to the first time slot where the anomaly occurred
unsigned pktcount[2]: packet count separate for each slot
int find[NVAR]: anomaly present on each of the 3 metrics (packets, bytes, SYN).
int grows[NVAR]: increasing anomaly on the next slot on each of the 3 metrics.
int strongestvar: which metric among the three have the strongest variation.
float impact[NVAR]: percentage of "variation" that this anomaly contributes to the last level that it is responsible for (imagine if an anomaly at 1.1.1/24 has a P of 500, and an anomaly at 1.1/16 has a P of 450, and there's no anomaly at 1/8, then the impact for that metric will be 500/450)
int impactlevel[NVAR]: the last level that the anomaly "impacted" for each metrics (i.e. in our example above, the impactlevel would be 1)
int duration[NVAR]: the duration of the anomaly in number of time slots
float decrease[NVAR]: biggest decrease on each of the 3 metrics
int children[NVAR]: number of anomalies inside all the networks which are in the next level in IP network(next_octet) of the network where this anomaly list is located
int destcount: number of IP addresses target of the anomaly
struct responsible_list *dest: pointer to the data structure with the details about the destinations of the attacks
struct anomaly_list *next:

4.3.2 Distributed detection of anomalies

We will first present an abstracted and generic description of our proposed state exchange scheme. Let us assume that we have a network with M nodes in $\boldsymbol{N}$ that are connected through a given graph ($\boldsymbol{N}$, $\boldsymbol{E}$) with edges in $\boldsymbol{E}$. Each node $\boldsymbol{i}$ observes at time $\boldsymbol{k}$ a state vector $\mathbf{X}^i[k]=\left(X_1^i[k],...,X_L^i[k]\right)^T$. The state vector can be any set of numerical metrics that the node has observed, *e.g.* netflow-like information, or measured load, or the sensed metric from the environment in a sensor network. This definition of state vector covers virtually all scenarios where nodes want or need to exchange numerical information. We will use bold characters to represent vectors of variables and light characters for single variables. All vectors are assumed to be column vectors. We will assume that $\mathbf{X}^i$ is a centred Gaussian stochastic process. We will later discuss the effect of deviation from the Gaussian hypothesis. The estimated value of the state vector of node *i* at node $\boldsymbol{j}$ is written as $\hat{\mathbf{X}}^{i,j}[k]$. For readability, we will frequently drop the time index for vectors. We assume that each vector, unless otherwise stated, is a multidimensional stochastic process with independent temporal samples.

4.3.2.1 Linear Estimation

A crash course in linear estimation, see Annex 2.

4.3.2.1 Source Compression

A crash course in source compression, see Annex 3.

4.3.2.3 State Sharing and Compression

To explain how the state-sharing scheme works, we will describe a simple one hop scenario. In this scenario, we assume that m nodes are all connected to a node c. All these m nodes send some information to the node c such that an approximation $\hat{\mathbf{X}}[k],\quad k=1,...,n$ is derived at this node. Node c may thereafter re-forward the estimated states to the other nodes. Obviously, one solution consists in sending the exact values of the state vector to other nodes. However this solution suffers from several drawbacks. First, this will results in wasting the bandwidth, as the state elements at different as well as at a single node, might be correlated. By making use of these correlations one should be able to reduce largely the number of bits to send. Secondly, we assumed that the nodes are selfish, *i.e.* they want to send as much as needed and not more. Sending the entire state vector is therefore the last resort. When a node has disclosed all its states it has nothing else to trade and he is not even insured that the other node acting as game player will give him back any information about its own states. We will describe here how to address these two issues.

1. Compression by sampling

Let's first deal with the efficiency of the communication between the nodes. A bandwidth cautious node will at first attempt compress the data to be sent to the other node. Let's assume that the node is ready to consume up to R bits per time step to send its state to the node c. A first and simple solution to reduce the bandwidth and to make it fitted in the R bits per time steps budget is to sample the states and to send the samples in place of all values. As the state values are correlated, applying 1: n sampling reduces the consumed bandwidth by a factor of n and .One can take advantage of the correlation to estimate the missing values. The sampling can be done temporally (choosing 1:n of temporal indices) or spatially (choosing 1: n of nodes), or spatio-temporal (mixing temporal and spatial sampling). Indeed the performance of such a sampling depends on temporal (correlation between successive samples of a single state variable) and spatial (correlation between variables of the state vector of a single node or between state variables of different nodes) correlation. We can formalize the sampling solution by observing that if we group sampled values over a time period of length T in a vector $\mathbf{Y}$, this vector can be described as $\mathbf{Y}=C\mathbf{Z}$, where $\mathbf{Z}=\left(X_1^i[k],....,X_1^i[k-T],...,X_L^i[k],....,X_L^i[k]\right)$ of dimension LT is constructed by concatenating T state vectors $\mathbf{X}^i[k],\quad t=k,....k-T$; the projection matrix C is a 0-1 matrix where $C_{ij}=1$ if the *j*'th value to be sampled is the *i*'th value in the vector $\mathbf{Z}$. This is the vector $\mathbf{Y}$ that is sent to c. At node c, the unsampled values are reconstructed by applying linear estimation formula with

projection matrix C and $\mathbf{V}$=0. However, this derivation needs the knowledge of the covariance matrix of $\mathbf{Z}$. As we assume stationary state vectors, this covariance matrix can be sent once at the beginning of the communication and can be used thereafter. As this transmission will be done only once its effect on the transmitted rate tends to become negligible.

2. Local compression

A second solution consists of compressing the state vector of each node independently from other nodes. In this setting one takes advantage of the temporal and the spatial correlation in a node state vector $\mathbf{X}^i[k]$ to reduce the amount of information to be sent. Let's assume that the temporal correlation on variable $X_j^i[k]$ becoming negligible after T steps. We construct at the node the vector $\mathbf{Z}=\left(X_1^i[k],\ldots,X_1^i[k-T],\ldots,X_L^i[k],\ldots,X_L^i[k]\right)$ of dimension LT described above. This vector behaves as an *iid* vector (as it encloses all temporarily correlated valued up to the T steps horizon), so we are falling into what described previously in the crash course on compression. As the elements of $\mathbf{Z}$ are local, its covariance matrix is easy to derive locally and one can easily apply the KLT to it to derive the orthonormal projection matrix $U_{LT\times LT}$. The quantization step results in a vector $\mathbf{Y}=U^{L^*}\mathbf{Z}+\mathbf{V}$ that is transferred to the node c with a rate R. The state vector elements $X_j^i[k]$ are reconstructed using linear estimation formulas and results in an error variance that is known. To enable the reconstruction of $\mathbf{Z}$, we need to send one time to node c the covariance matrices $\sum_{\mathbf{Z}}$ and $\sum_{\mathbf{V}}$ as well as the projection matrix U^{L^*} that has been used. However as these values have to be sent only a single time, the effect of the transmission of these data on the overall transmission rate vanishes with time. Can we do better and attain a lower MSE with the same transmission rate R? The answer is yes as we have not exploited all the existing correlation. There might be some correlation between the state vector of the nodes and by exploiting this correlation one can attain a better performance. We will present in next section how to achieve this.

3. Distributed compression

In the distributed setting we are assuming that the node i sends to the node c a noisy projection $\mathbf{Y}^i=C^i\mathbf{Z}^i+\mathbf{V}^i$. Our goal is to derive the projection C^i and the quantization noise parameters $\mathbf{V}^i$ such that the central node receiving at most mR bits per time steps can estimate the state vectors $\mathbf{Z}^i$ with a Minimal Mean Squared Error. At node i we have to choose the optimal projection C^i and the quantification noise $\mathbf{V}^i$ assuming that the node i has access to the information sent by other nodes. Let's define Z^i as the vector build by concatenating all local state vectors, $\mathbf{Z}^i$ of node c neighbours. We also define Y^i as the vector containing all information available at node c besides the information sent by node i, *i.e.* Y^i is built by concatenating all received vectors $\mathbf{Y}^j$, $j\neq i$ with the state vector of node c, X^c. Now let's assume that node i is not sending any data to central node c. Therefore the best estimate of $\mathbf{Z}^i$ at node c knowing the vector Y^i can be obtained using linear estimation formulas. However this estimation will entail an estimation error we denote by W^i. As the node c is already aware of Y^i, node i has just to send an approximation of W^i to improve the estimation of $\mathbf{Z}^i$. This observation is the key to reduce the amount of information that has to be given by node i to node c. We have therefore to estimate W^i at node i and to send this variable in place of $\mathbf{Z}^i$. In [distKLT] a derivation is provided that estimate W^i and that obtains the optimal projection C^i and quantization noise $\mathbf{V}^i$ to send information validating the transmission rate budget. By following this approach one can construct a distributed compression scheme to exchange node states with the central node. This is this last scheme that we are going to use and to adapt to our setting of distributed anomaly detection.

5. Experimentation

5.1 Case a1: Adaptive traffic sampling and management

5.1.1 Performance objectives, and evaluation criteria

Collected measurements can decide on how to tune the sampling rates inside the network. We consider for this purpose an important monitoring application, the estimation of the volume (in terms of number of packets or bytes) of some chosen network flows. A flow is a set of 5-tuple flows that share some common features as the source IP address prefix, the destination IP address prefix, the same ingress and egress routers inside the network, etc. At the limit, a flow can be one 5-tuple flow or even the entire network traffic. Given a set of flows to monitor, the machine learning engine should progressively tune the sampling rates in routers in such a way to minimize the global estimation error.

5.1.1.1 Scenario

Consider N traffic flows whose volumes in packets are labeled F_1, $F_2 \ldots F_N$. We denote by, $\hat{F}_1$, $\hat{F}_2 \ldots \hat{F}_{N,}$ the corresponding estimators. Let $P = (p_k)_{k=1..M}$ be the vector of sampling rates in the different monitors of the network (a monitor is equivalent to a router interface). There are in total M monitors. The target of the machine learning engine is then to find the vector P that minimizes the sum of normalized estimation errors:

$$\sum_i \frac{\mathrm{Var}(\hat{F}_i)}{(F_i)^2}$$

Each flow F_i is formed of a set of 5-tuple flows whose volumes are denoted by S_{ji}. Again, denote by $\hat{S}_{ji}$ the best estimator for the size in packets of each of these 5-tuple flows. One can then transform the optimization problem into minimizing the sum of the normalized estimation errors of the sizes of the 5-tuple flows:

$$\sum_i \sum_j \frac{\mathrm{Var}(\hat{S}_{ji})}{(F_i)^2}$$

This minimization is carried out in the following way. Suppose one decides to reduce (resp. increase) the sampling rate by δ in the least (resp. most) significant monitor. The task is to find the monitor k having the smallest (resp. largest) absolute value for the following partial derivative sum and to decrement (resp. increment) its sampling rate by δ:

$$\sum_i \sum_j \frac{\partial \mathrm{Var}(\hat{S}_{ji})}{\partial p_j} * \frac{1}{{F_i}^2}$$

In the following, we show how such estimators for the 5-tuple flow sizes are formed and how the partial derivatives of their variances are obtained. For the F_i themselves, which are unknown, we simply substitute them by their estimations, i.e. $\hat{F}_i = \sum_j \hat{S}_{ji}$.

Note that the volumes of flows are measured in packets. The passage to bytes can be made by multiplying the size in packets by the average packet size, which we suppose true for large flows (stationary packet size).

5.1.1.2 Local flow size estimation

Consider a 5-tuple flow S_{ji} crossing monitor k whose sampling rate is p_k. Let s_{kji} be the number of packets sampled from this 5-tuple flow in the monitor (this number could be zero). Using information, one can derive a first estimation for the flow size of the flow. The estimator that maximizes the likelihood is known to be:

$$\hat{S}_{kji} = s_{kji}/p_k$$

Under independent sampling of packets with probability p_k, the number of packets s_{kji} sampled from an original 5-tuple flow S_{ji} follows a binomial distribution whose variance is well known and equal to:

$$S_{ji}p_k(1-p_k)$$

It follows that this local estimator for the size of a 5-tuple has a variance equal to:

$$\mathrm{Var}(\hat{S}_{kji}) = (S_{ji}(1-p_k))/p_k$$

5.1.1.3 Combining measurements

We estimate the volume of a 5-tuple flow as being the sum of the weighted sum of the local estimators done in the monitors along its path. This gives the following global estimator for 5-tuple flow *j* belonging to aggregate flow F_i:

$$\hat{S}_{ji} = \sum_k \lambda_k \hat{S}_{kji} = \sum_k \lambda_k s_{kji}/p_k \text{ with } \lambda_k = \frac{1/Var(\hat{S}_{kji})}{\sum_l 1/Var(\hat{S}_{lji})}$$

Replacing the variances by their expressions given in the previous section, substituting the second equation in the first one, and simplifying by *Sji*, we get:

$$\hat{S}_{ji} = 1/\alpha_{ji} \sum_k \beta_{kji} s_{kji}$$

$$\text{With: } \alpha_{ji} = \sum_l \frac{p_l}{1-p_l} \text{ and } \beta_{kji} = 1/(1-p_k)$$

Note in particular how the α_{ji} and the β_{kji} are the same for all 5-tuple flows that follow the same path, which eases a lot the calculation. As for the variance of this estimator of 5-tuple flow sizes, it is simply equal to:

$$Var(\hat{S}_{ji}) = \frac{S_{ji}}{\alpha_{ji}}$$

5.1.1.4 Reconfiguring Monitors

The variance (or mean square error) of 5-tuple flow size estimation is very important for the determination of the global system accuracy and the monitor where to tune the sampling rate. For 5-tuple flow S_{ji} and monitor *k* we can write:

$$\frac{\partial \mathrm{Var}(\hat{S}_{ji})}{\partial p_k} = \frac{-S_{ji}}{(\alpha_{ji})^2 (1-p_k)^2}$$

This represents the marginal gain in accuracy (loss in the variance) when the sampling rate of monitor *k* is increased by a small step δ from the perspective of estimating the size in packets of flow S_{ji}. As expected, this gain is always positive when increasing the sampling rate (more sampling more accuracy). It also decreases when p_k increases, which suggests that the estimation error follows well a continuously decreasing and convex function with the sampling rate, a condition required for the uniqueness of solution in non-linear optimization theory. By plugging the expression below which sums the accuracy and normalize it over all 5-tuple flows forming the traffic of interest, one can easily find the total gain (resp. the loss) in accuracy when the sampling rate of monitor *k* is tuned up (resp. down) by some small step.

$$\sum_{\mathrm{i}}\sum_{\mathrm{j}}\frac{\partial \mathrm{Var}\left(\hat{S}_{ji}\right)}{\partial p_j} * \frac{1}{{F_i}^2}$$

This total gain (resp. loss) is from the perspective of the traffic accounting application. By testing all monitors, one can find the best sampling rate to tune up or down on the way to the optimal configuration.

5.1.2 Methodology: scenarios and tools

5.1.2.1 Validation scenario of our Adaptive Traffic Sampling and Management Platform

To emphasize the practical utilisation and the scalability of the developed platform, we choose to reproduce the GEANT backbone topology. The GEANT network is a pan-European backbone that connects Europe's national research and education networks via 23 routers. On the other hand, we choose to replay different traces collected at a transpacific line from the MAWI working group traffic archive (we use the traffic captured on some high speed link in a backbone transit network between 9:00AM and 11:00PM on 03/03/2009). Traffic traces are made by TcpDump, and then, IP addresses in the traces are scrambled by a modified version of Tcpdpriv (http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html/). For traffic dispatching over different access networks (European countries in this experiment), we associate different weights to the 23 Autonomous Systems (AS) connected to GEANT based on the importance of the traffic they are expectedly generating. We infer these weights from the populations of the countries represented by these ASes and the capacities of the links that connect them to their respective access routers in GEANT. Note that we run the three services composing our platform within three separated machines.

5.1.2.2 Experimental results (Platform validation results)

In the following paragraphs, we validate the effectiveness of the traffic emulation service and especially of the dispatching module it incorporates. We track the number of sampled 5-tuple flows and the number of packets across the different edge routers to check whether they follow the access network weights (or traffic matrix) predefined in our experiment configuration file.

Figure 21 depicts the evolution of the network-wide monitored number of flows in function of the sampling rate (variation is kept identical for all routers). As expected, the number of flows decreases linearly as we decrease the sampling rate in all the access routers. Indeed, if **S** is the size of a given flow, then the probability that this flow is sampled given a sampling rate **p** is equal to **1 – (1 – p)S**, which can be approximated by **p.S** for small **p**. The number of sampled flows **Np** can then be approximated by **p.E[S].N**, where **N** is the total number of original flows. This latter quantity clearly decreases linearly with the sampling rate.

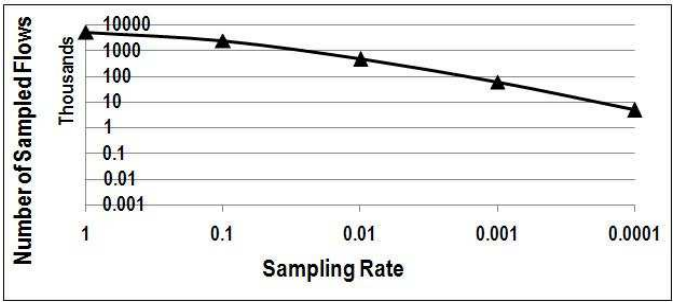

Figure 21. Number of Sampled Flows for different sampling rates

Next, we look at the effectiveness of the traffic emulation service and especially of its dispatching mechanism. Here, the purpose is to (try to) answer the following question: does each AS generate as much traffic as the importance of the weight associated to it? Towards that, we start by plotting in Figure 22 the number of prefixes per AS function of the weights associated to AS. The resulting curves remain linear for different prefix lengths. So, we conclude that our emulator dispatches the prefixes to AS without any bias.

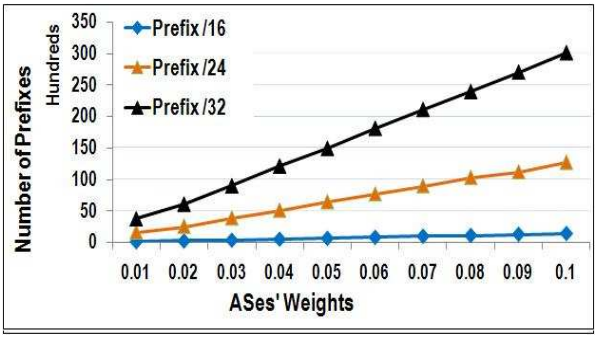

Figure 22. Prefixes dispatching

Then, we look at the number of generated flows, ingoing and outgoing packets per AS in Figures 23, 24 and 25, respectively. We notice that for the three prefix lengths (/16, /24 and /32), the number of generated flows as well as the number of ingoing/outgoing packets scale with the AS weights but do not fit a perfect line. Nevertheless, the fitting improves when the prefix granularity becomes finer. This improvement results from the presence of very large prefixes (ex: prefix /16 including different servers' IP addresses which are generating a large volume of traffic) of different volumes that, when dispatched over the AS, cause such deviations in the traffic; the coarser the prefix the more important this phenomenon. We illustrate it on prefixes of length /32 (we plot the distribution of the prefix /32 sizes, we mean by size the number of packets from a given ip address that belongs to the prefix /32), for which we plot the distribution of their sizes (in packets) on a log-log scale in Figure 26. Clearly, there is a power-law behaviour leading to very large prefixes compared to the average prefix size (these are servers, heavy users, etc). To improve further the fit, one can run the dispatching at another finer granularity, the 5-tuple level. This finer granularity enables our dispatching algorithm to split a large set of 5-tuple flows generated by (or destined to) a single /32 prefix to different AS. As it can be observed from Figures 23, 24 and 25, the number of generated flows and ingoing/outgoing packets better fits now a line. Even though it preserves the notion of connections, the 5-tuple dispatching still has the problem of altering the pattern of activity of servers and end hosts. The choice of the best prefix length should be decided by the tradeoff established between traffic realism and AS weight respect.

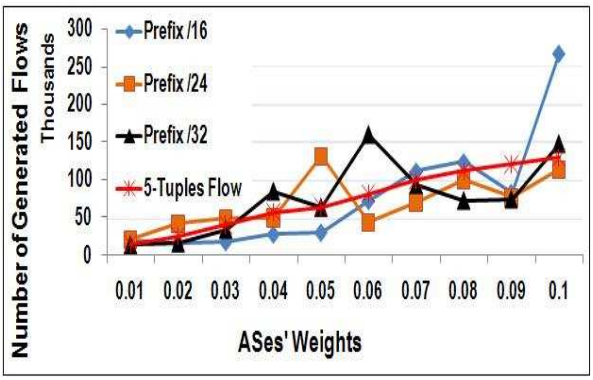

Figure 23. Flows dispatching

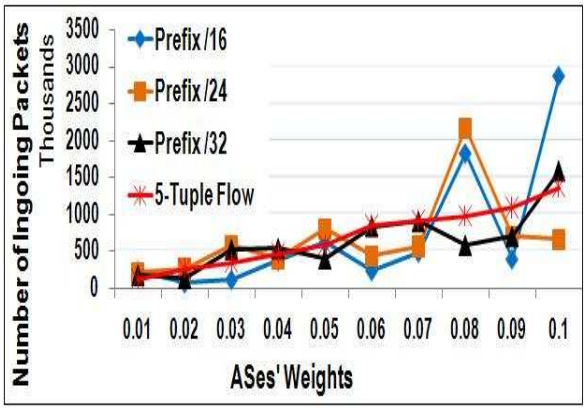

Figure 24. Ingoing packets dispatching

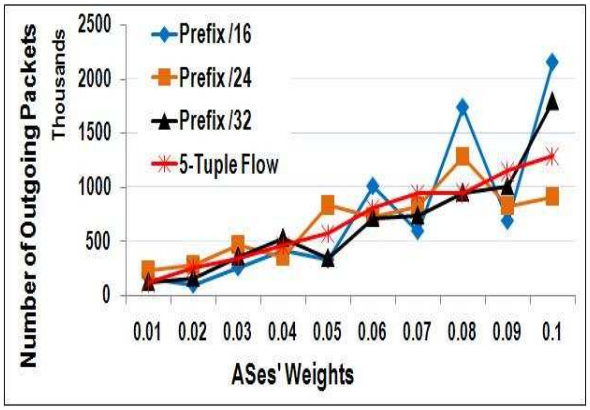

Figure 25. Outgoing packets dispatching

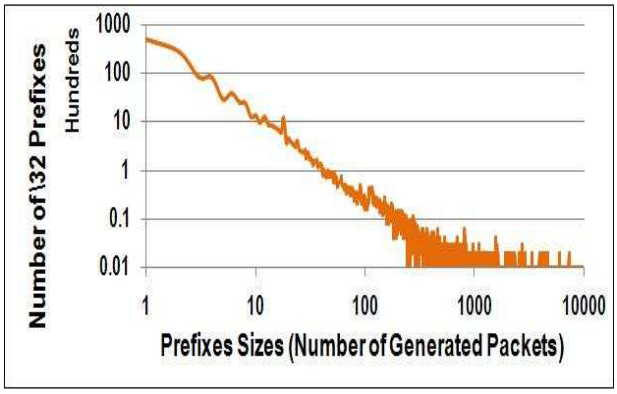

Figure26. Prefixes sizes distribution

5.1.3 Future work

As a future work, we intend to develop an API that introduces wide collaborative sampling methods within the traffic monitoring and sampling service. This API will be of a great help to us towards studying and developing such distributed sampling approaches.

5.2 Case a2: Global monitoring

5.2.1 Performance objectives, and evaluation criteria

The global monitoring has to be evaluated according to two dimensions:

1. It must be able to capture fully and accurately a trace of all packets passing through it. This means checking that all packets on the link have been captured and that the timestamp is accurate.
2. When exchanging information between monitoring nodes, the related traffic has to be as transparent as possible, i.e. as low as possible and not interfering with the background traffic of users.

5.2.2 Methodology: scenarios and tools

5.2.2.1 Validation of the capture module

The methodology for validating that the monitoring is able to capture accurately a trace of all packets requires comparing its resulting traces with a reference trace. For getting a reference trace, we are using a hardware capture technique which is guaranteed to provide perfect results: we use for this purpose a DAG system. The ECODE capture and the DAG systems are located on the same link and then are crossed by the same traffic. In addition, the times at which a packet is crossing the two capture system are very close as they are related to the physical propagation time on a cable or fibre (few centimetres between the two systems in our experiments). The difference is hardly measurable at the considered time scale (few microseconds). Then, it is sufficient to check that all packets captured by the DAG system are also in the trace captured by the ECODE one, and to compute the differences between their related timestamps.

5.2.2.2 Validation of the reporting system

The validation methodology has already been described in details in ECODE deliverable D3.1 [ECODE 2009]. To summarize, it consists in emulating different network topologies with monitoring nodes on some of the links, and to measure the reporting traffic generated as well as the global performance of the network (delay, loss, etc.). The reporting traffic consists of all information about raw traffic data or results of preliminary analysis of raw traffic exchanged between different monitoring nodes in different location for permitting all monitoring nodes to have a global view of all the traffic transiting in the network. Then, the reporting traffic and the network performance are closely analyzed to check whether the reporting traffic is transparent for the network behaviour.

5.2.3 Experimental results

As for the moment only the capture system is available (the reporting system has not been completed yet), we just performed few evaluations of it. It however appears that the traffic traces captured by our system are complete (all packets are present) and the timestamps are similar.

5.2.4 Future work

For the moment, the validation of the capture system has only been performed on actual operating links, i.e. links that are largely unused at their maximum capacity. Future work will then include similar evaluations but this time generating ourselves high traffic burst to measure how much the capture system is sensitive to link load. Future work will also include the evaluation of the reporting mechanisms once its development will be complete.

5.3 Case a3: Cooperative distributed anomaly detection

5.3.1 Attribute based local detection of anomalies

5.3.1.1 Performance objectives, and evaluation criteria

The performance of the ADS will be evaluated on the false negative and false positive rates. The false positive rate is the ratio between the number of undetected attacks over the total number of attacks. The false positive rate is the ratio between the number of false alarms over the total number of attacks.

5.3.1.2 Methodology: scenarios and tools

We use two datasets to validate our algorithm: the METROSEC project traces with artificially created anomalies and the MAWI traffic repository with anomalies seen on the wild. We concentrate on DDoS anomalies for their importance and the number of different shapes they can appear with. If we are able to successfully separate different DDoS anomalies from normal traffic and from other types of anomalies, it might follow that general automated classification of network traffic anomalies is possible. Note that because of space limitation, only the most significant results are presented. A full description of the validation process and results can be found in [Fernandes 2008].

The METROSEC traces consist of real traffic collected on the French operational network RENATER with simulated attacks performed using real DDoS attack tools. This dataset was created in the context of the METROSEC research project to, among other goals, study the nature and impact of anomalies on networks' QoS. This dataset has been used for validation by a number of different studies on anomaly detection (e.g. [Scherrer 2007]). For the validation of our algorithm, we use 14 METROSEC traces containing DDoS attacks of intensities ranging from very low (i.e. 4-10% of the whole traffic) to very high (i.e. 87-92%). The attacks also vary in type (i.e. from TCP SYN flooding to Smurf attacks), number of attacking hosts (i.e. 1-4) and duration.

On the other hand, the MAWI dataset has real undocumented anomalies. It is composed of 15 minutes packets traces collected daily at 2PM from a Japanese network called WIDE since 1999 to present. These traces are provided publicly after being anonymized and stripped of their payload data (see http://mawi.wide.ad.jp/). Although these traces are undocumented, the authors of [Dewaele 2007] started an effort to label anomalies found in this database. We randomly selected a total of 30 traces from 2001 to 2006 from which some had

already been identified by [Dewaele 2007] to contain DDoS anomalies. Using this second dataset is important to verify that our algorithm is not restricted to a single network or to artificial attacks.

The validation of our algorithm is divided in two parts. In the first part, a (proper) statistical validation is performed using the METROSEC traces for the classification of DDoS anomalies. Different levels of sensitivity of the detection algorithm are used by varying its K parameter from 1.5 to 6. The classification signatures used are the same for all values of K, but only DDoS related signatures are considered. In the second part, the classification performance of our algorithm is tested for different types of anomalies (i.e. DDoS, port and network scan, and attack response) on both of the datasets presented in the previous section. K is set to a constant value 2, and all the signatures are enabled (including the same DDoS signatures used in the first part). A granularity of 30 seconds and the levels of aggregation 0, 8, 16, and 24 are used in the detection algorithm for both parts.

For visualizing the performance of any anomaly detection tool it is possible to use ROC curves (Receiver Oriented Curves), in particular the probability of good detection depending on the probability of false alarm P_D vs. P_F. The ideal point for which all attacks would be detected and no false alarm raised is the left upper corner. The worst case is the diagonal, for which the results do not significantly differ from those obtained with random detection algorithms. ROC curve, among other solutions, provides a simple mean to compare the performances of different anomaly detection tools.

5.3.1.3 Experimental results

The classification performance for the first part of our validation was very similar for all values of K (i.e. the algorithm achieved a very high rate of correct classifications with a very small rate of misclassifications). The results obtained with K equal to 2 include 23 true positives (i.e. DDoS anomalies correctly classified), 2 false positives (i.e. non-DDoS anomalies misclassified as DDoS), 1 false negative (i.e. misclassified DDoS anomaly) and 455731 true negatives (i.e. non-DDoS anomalies classified as non-DDoS). Further analysis showed that one of the false positives was actually a real, unexpected DDoS ICMP reflector attack, and the attack responsible for the false negative was correctly classified in a subsequent anomaly.

The results for the second part of our validation were equally promising. On the METROSEC traces, the non-DDoS signatures found a total of 16 port scans, 13 attack responses and 2471 network scans. Manual analysis showed that all port scans and 10 attack responses were true positives. We were not able to identify the nature of the other 3 attack responses. Network scans were not manually analyzed, but the signature used (see Table 2) has a very low (if not inexistent) misclassification rate. Running the algorithm on the 30 fifteen minutes MAWI traces resulted in 22 DDoS, 4429 network scan, 5233 port scan, and 72 attack response anomalies in a total of 2.5 million anomalies detected. Manual analysis and cross-referencing with the results of [Dewaele 2007] revealed 19 true positives (of which 6 had not been detected by [Dewaele 2007]), 3 false positives that might be ICMP reflector attacks, and 9 (known) false negatives. The false negatives were mainly due to the detection algorithm used, and are not a limitation of our classification approach or of the signatures used. Preliminary analysis of the other type of anomalies showed that many of them were due to worm scans (and responses), with Sasser and Dabber variants being particularly common.

5.3.1.4 Comparison (for experiments targeting the same objective(s)

In order to compare the performance of our detection and classification algorithm based on the use of deltoids / signatures with other similar tools, we applied the same method with other tools having similar objectives. We then evaluated the performances of:

- NADA, i.e. an anomaly detection tool based only on the use of deltoids [Farraposo 2007]
- A Gamma-Farima tool [Scherrer 2007]
- PHAD (Packet Header Anomaly Detection) [Mahoney 2003]
- Our tool based on deltoid plus signature based classification we decided to call NewNADA

The ROC curves (see Figure 27) show True Positive vs False Positive rate of these tools for the considered traces and anomalies. It appears that PHAD is not very good as it appears as a little bit better than a random detection process. Gamma-Farima which uses more complex statistics than NADA exhibits better performance than NADA, but at the expense of attack constituting packet identification capabilities. Finally, it clearly

appears that our signature based classification algorithm added to the deltoid based detection has almost perfect results and is thus a good approach to follow for future work.

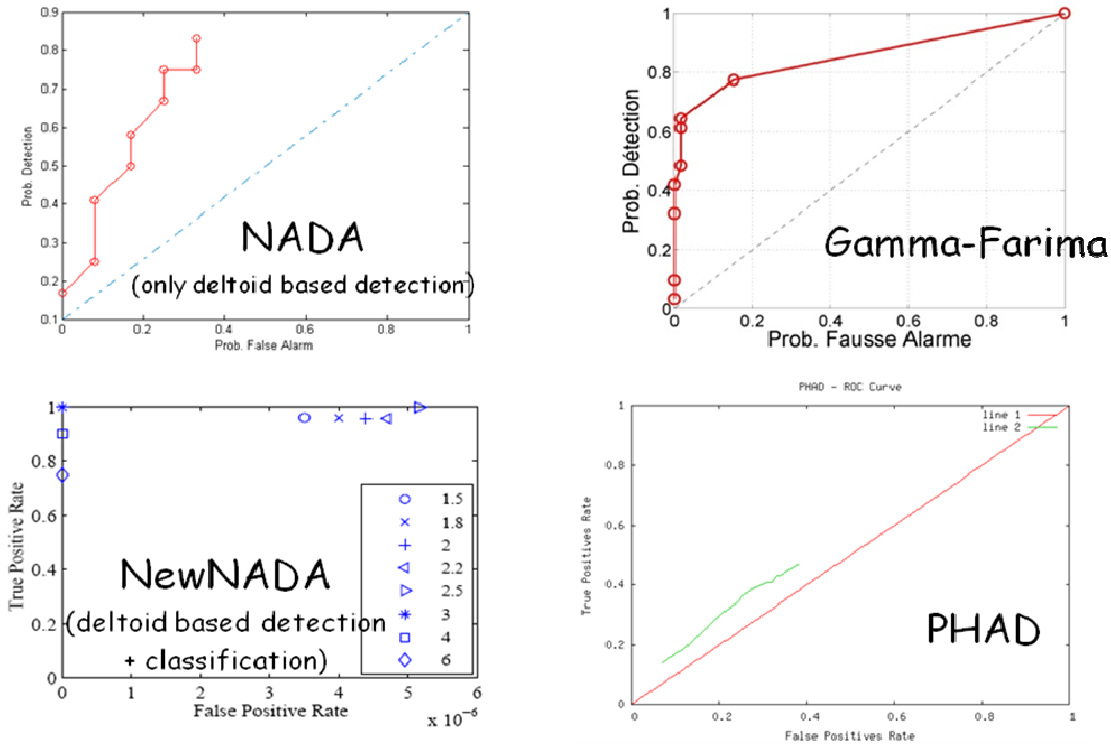

Figure 27. Comparison of the respective performances of four different anomaly detection tools.

5.3.1.5 Future work

At this stage, we have tested the principle of the signature based methodology which proved to be very promising. However, the rules are sill static. But, as described in Section 3, we are in the process of integrating in this algorithm some machine learning techniques for being able to autonomously fixing thresholds in the existing signatures, as well as producing new rules for 0d anomalies. Future work will then essentially deals with assessing these two new capabilities of the algorithm.

5.3.2 Distributed detection of anomalies

5.3.2.1 Performance objectives, and evaluation criteria

The aim is distributed anomaly detection is to achieve the best detection rate *vs.* false alarm trade-off. However, because of the distributed nature of our approach, we add another dimension that is the cost of the detection in term of information exchange rate between nodes.

5.3.2.2 Methodology: scenarios and tools

The analysed scenarios are constructed around a network with several monitoring nodes that want to jointly detect anomalies by exchanging their state data. As a toy example, we use IP-level traffic flow measurements collected over an academic backbone network. The data are derived by processing sampled flow data from every router of this network for a period of one week. In this dataset we distinguish between incoming and outgoing traffic, as well as UDP and TCP flows. For each of these four categories, we computed seven commonly used traffic features: byte, packet, and flow counts, source and destination IP address entropy, as well as unique source and destination IP address counts. The state variable to exchange there is of dimension 28 (7 parameters per 4 categories). All metrics were obtained by aggregating the traffic at 1 second intervals resulting in a 28x86400 data matrix per measurement day per interface. We analyzed 41 interfaces distributed in 11 different routers. All values are stored as integer over 32 bits, resulting in an overall rate of 28x41x 32=36768 bits per seconds for transmitting the data without compression.

5.3.2.3 Experimental results

In the first setting, we assume that all these routers have to send their state vectors to a central aggregation point and we evaluate the performance of the scheme in steady state in term of level of MSE attained as a function of overall rate in term of number of bits per sample of 28 x41=1148 state values. The sum of variance over all states amount to 8.8x 10^{16}. The best performance achieved by 1:2 sampling results in 18389 bits per seconds, transfer rate is 2.94x 10^{16}. By using the schemes proposed in our work, we can achieve performances shown in Figure 28.

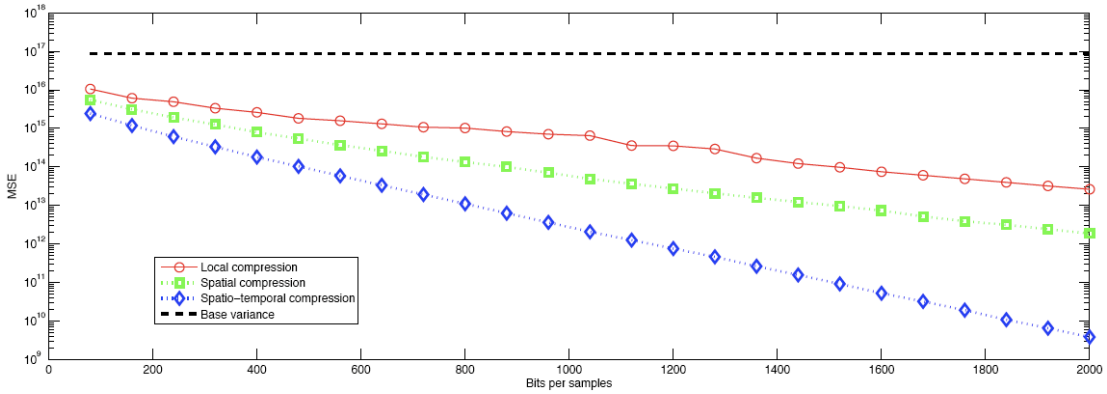

Figure 28: Overall MSE achieved in the single hop scenario for the temporal, the spatial and the spatio-temporal setting

Figure 28 shows three settings: in the first setting, we implement local compression; in the second setting, namely spatial compression, we implement the distributed scheme described previously but assuming that the temporal correlation horizon is 1 (that is not really the case); in the third setting, we implement the spatio-temporal compression scheme with a temporal correlation horizon T=5. One can observe that with just 2000 bits per sample (meaning a compression rate higher than 18) we can achieve an MSE that is 7 orders of magnitude less than the initial overall MSE, moreover the application of spatio-temporal compression results in an MSE that is 4 orders of magnitude lower than just doing a local compression. In particular, the spatio-temporal compression achieves with just 400 bits the performance that the local compression achieves at 2000 bits. It is noteworthy that the performance of the best sampling scheme (the 1:2 temporal sampling) was out of the graph and is not shown. It is worthy to discuss about the overhead involved in this scenarios. For this purpose, we show in figure the MSE achieved at one node as a function of the iteration number. Each iteration took 30 seconds as we estimated the covariance matrix using at least 30 samples to have an acceptable estimation. At each iteration, a new projection matrix has to be forwarded to the central point. We assume those projection coefficients are encoded into 16 bits. However the projection matrix for the spatio-temporal cases is as large as the dimension of the initial space is 5 times larger. The number of forwarded projection varies as the value L^* changes.

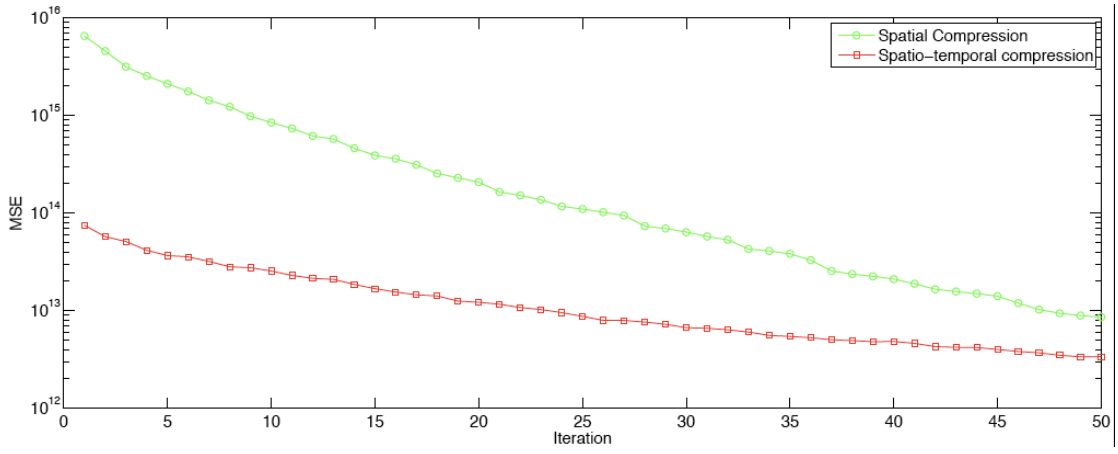

Figure 29: MSE achieved after a number of iteration of the distributed compression scheme for a global rate of 1200 bits per sample

Figure 29 shows that the number of needed iterations to stabilize the estimation is also different between the spatial and spatio-temporal scheme; the spatio-temporal scheme needs about 20 iterations where the spatial scheme needs about 40 iterations. Accounting all these sources of variation, for the spatial scheme we have observed an overhead around 119 Kbits, and for the spatial-temporal scheme an overhead around 962 Kbits. Accounting the difference in needed rate for achieving a given MSE between spatial and spatio-temporal scheme (for example, to achieve an overall MSE of $2x10^{14}$ we need 2000 bits per sample for spatial scheme and 400 bits per sample for spatio-temporal scheme), the difference in overhead between the two schemes is covered in 527 seconds (about 9 minutes of operation), and the difference between the uncompressed scheme and the spatio-temporal scheme is covered in 25 seconds, meaning that the use of the overhead is largely balanced with the benefit. The overhead analysis gives similar results for other values of global rate.

5.2.3.4 Future Work

Our work in the next steps will consists of developing a node state exchange module inside the ECODE architecture and to use the exchange states to implement anomaly detection. This will be achieved in the coming reports.

6. Recommendation for integration into common ECODE architecture (see D2.1)

6.1 Case a1: Adaptive traffic sampling and management

In the following paragraphs, we explain the way our architecture' key components fit within the global ECODE architecture. We provide both a specification of the interface between the machine learning engine (MLE) and the monitoring engine (ME) as well as the interface between two different MLE(s).

6.1.1 Communication between the MLE and the ME

The adaptive traffic sampling and monitoring architecture that we are proposing relies mainly on a passive monitoring solution. The latter consists on the traffic monitoring and sampling service which we have already described in the paragraph 4.1.2.

The traffic monitoring and sampling service will be incorporated within the ME (Monitoring Engine) component of the global ECODE architecture, and will relies on the CM (Cognitive - Monitoring) interface towards exporting the collected statistics to the MLE. The latter statistics consists on NetFlow Reports and more specifically on the Cisco Netflow(tm) version 5 packet format [Netflow].

A NetFlow report format can be described as follows: [**NF5_HEADER**, **NF5_FLOW**, **NF5_FLOW**, **NF5_FLOW** …]

Knowing that the **NF5_HEADER** describes the set of flows to report, it could be represented via the following structure:

```
➢ struct NF5_HEADER {
        u_int16_t version, flows *;
        u_int32_t uptime_ms, time_sec, time_nanosec, flow_sequence;
        u_int8_t   engine_type, engine_id, reserved1, reserved2;
        // Added to support the Traffic Emulation Service
        u_int32_t  emulator_monitor_adr *;
};
```

Regarding the **NF5_FLOW**, it describes a given 5-tuple flow and could be represented via the following structure:

```
➢ struct NF5_FLOW {
        u_int32_t  src_ip *, dest_ip *, nexthop_ip;
        u_int16_t  if_index_in, if_index_out;
        u_int32_t  flow_packets *, flow_octets *;
        u_int32_t  flow_start *, flow_finish *;
        u_int16_t  src_port *, dest_port *;
        u_int8_t pad1;
        u_int8_t tcp_flags *, protocol *, tos;
        u_int32_t  src_as *, dest_as *;
        u_int8_t src_mask *, dst_mask *;
        u_int16_t pad2;
};
```

Note that for the evaluation of our adaptive sampling technique, we only use the fields followed by *.

Once the NetFlow reports are received by the MLE component, The Translator block will take in charge the extraction of the fields followed by * and passing them to the Representation block. The latter will transform these statistics into tagged observations which will be stored later within the o*bservation Information*

Base (OIB). The stored information could be retrieved later by our ML algorithm by means of the Register (RL) or loaded (on-demand) by the Processing block.

Note that the main goal of the architecture we are proposing is to develop an autonomous system for network monitoring and traffic management. Starting from a measurement task, like for example the calculation of the traffic matrix, the estimation of flow sizes and rates, the prediction of flow rate increase/decrease, or the detection of anomalies, the system will configure the sampling rates in network routers so as to optimize the accuracy while limiting the overhead (volume of collected traffic, packet processing and memory access in routers). So at some point, the algorithm which we already described in paragraph 5.1.1 and which we will incorporate within the MLE will need to change the sampling rate of a given ME at a given network interface. The latter communication will be ensured via the CM interface. The format of the control message to be exchanged is as follows:

> [Command = change the sampling rate, Network Interface = a.b.c.d, New sampling rate = x/y (y > 0) we send the couple (x, y)]

This control message is sent from (triggered by) the MLE to the ME each time the algorithm we are proposing decides to change the sampling rate at a given network interface towards optimizing a given task.

6.1.2 Communication between two different MLE(s)

For the moment our ML algorithm is supposed to run within a collector node. However, as future work, we may envisage to deploy it in routers among which we will exchange information (between MLE(s)) to boost the convergence for the best measurement model and optimal router configuration. Such communication will be ensured by the **CCr** interface (sub- interface defined between representation modules of peering cognitive routers and used to exchange input to the machine learning processing).

6.2 Case a2: Global monitoring

The passive monitoring part of the monitoring engine (the active part being described in deliverable D3.4) is in charge of:

1) Collecting packet traces from the forwarding plane I/O. Packet traces consist in collecting for each packet its IP and TCP headers with a very accurate timestamp. Depending on privacy laws, more data could be eventually captured (e.g. application headers, applicative payload): the monitoring system developed is versatile enough for offering a wide range of possibilities.
2) Making the requested processing, with the result format matching the input format of the MLE' translator. Let's recall that our monitoring engine allows network designers and developers to configure the monitoring engine for performing any computing on the incoming traffic data: They just need to pass the pointer on the appropriate function (they have previously written) to the monitoring engine at any configuration phase.
3) Transmitting them to:
    a. the other modules implemented in the Machine Learning or forwarding Engines
    b. the other monitoring nodes in other locations of the network when these latest need to get a global view of all traffic or performance measurements made on the global network under consideration. This functionality is one of the bases for implementing distributed machine learning algorithms easily.

6.3 Case a3: Cooperative distributed anomaly detection

6.3.1 Attribute based local detection and classification of anomalies

For implementing the detection and classification algorithms, the following overall thread needs to be developed in the ECODE framework. A continuous process ensures that packet data is entering the monitoring engine which on its turn sends the required fields to the responsible module in the Machine Learning Engine. As previously, by default, IP and TCP headers are captured for each packet together with an accurate timestamp. But depending on privacy rules, more information per packet can be captured.

There, the two stages anomaly detection and classification process builds the needed traffic models (algorithm taken as an example in section 2.3.1 considers traffic deltoids, but any other traffic model can be considered for this detection process, e.g. Markovian, Gamma-Farina, etc.), isolates the possible anomalies and classifies them. Among the different kinds of anomalies that can be detected, we distinguish between legitimate (as flash crowd or alpha flows) and illegitimate anomalies (as attacks) regarding the management of the anomaly.

- For illegitimate anomalies, it clearly means that the corresponding packets represent a useless load for the network. They then can be discarded. The Machine Learning corresponding module will then provide to the forwarding engine the characteristics of the anomaly in order to help it discard the faulty packets. At this stage, we have a first list of parameters characterizing anomalies (see table 1), but it still needs to be completed, as well as the alarm formats.
- For legitimate anomalies which significantly impact network performance, a new way of managing the routing or the forwarding has certainly to be set-up. Therefore, the machine learning module concerned with anomaly detection provides to the routing and forwarding engines the characteristics of the anomaly issued from the detection and classification process, in order them to apply the appropriate counter-measures.

6.3.2 Distributed detection of anomalies

The integration of the distributed anomaly detection component into the global ECODE architecture is taking advantage of all parts of the architecture. Two major parts of the overall ECODE architecture cooperate during the distributed anomaly detection: the MLE (machine learning engine) and the ME (Monitoring Engine). However there is a major interaction between distant MLE in our case.

6.3.3 Communication between the MLE and the ME

In distributed anomaly detection, we assume that the ME is extracting continuously and at regular time interval a set of numerical traffic metrics that are going to be shared with other monitors or routers in the network. Indeed, the use of certain metrics are more valuable than others for the sake of anomaly detection however as our main goal is to deal with the distribution of anomaly detection we will assume that the right set of metrics has been chosen and these metrics are provided regularly to the MLE. This set of metrics is used in two ways: they are used along with estimates of the state metrics of other nodes for anomaly detection at the node itself, and they are also transferred along with remote states to node neighbours for propagation to nodes that have an interest in receiving these states. Therefore, the communication between MLE and ME consists simply in providing all the metrics of interest gathered by the ME during the last reporting period to the MLE that will thereafter apply the linear projection and forward it to the neighbours.

6.3.4 Communication between MLEs

Because of its distributed nature distributed anomaly detection make an intensive use of the distribution module inside the MLE. Routers exchange regularly two types of data: linear projection sent frequently to neighbours (around one projection per second rate), projection matrices, along with preference lists that are sent in a lower rate (every time these values are changed). The communication will involve therefore forwarding of linear projection messages along with projection matrices and preference lists update.

6.3.5 Communication between the MLE and the FE and the RE

Whenever an anomaly is detected the MLE should take actions and implement them through the interfaces to the forwarding and the routing engine. At this stage of the project we have not yet formalized the corrective actions so this communication is not still formalized.

7. Conclusion

The goal of the ECODE project is to develop, implement, and validate experimentally a cognitive routing system that can meet the challenges experienced by the Internet in terms of manageability and security, availability and accountability, as well as routing system scalability and quality. By combining both networking and machine learning research fields, the resulting cognitive routing system fundamentally revisits the capabilities of the Internet networking layer so as to address these challenges altogether.

This deliverable more specifically addresses the first technological objective dealing with adaptive sampling, path monitoring, and anomaly detection. It also focuses on the implementation of each of these use cases. For each of the use cases, this deliverable describes how machine learning techniques have been used, and an evaluation demonstrates its benefits. Note also that the anomaly detection use case has been divided into two axes as two aspects are under concern: (1) the attribute based detection and classification of anomalies, especially allowing the detection of 0d attacks, and (2) distributed detection of anomalies for a detection at the network scale. Of course, at the end of the project, these two contributions will be put together.

At this stage, the solutions proposed for the different use cases have been evaluated on dedicated local platforms by the partners in charge. They exhibit the gain in terms of performance and capabilities in using machine learning for network router functionalities. Last, section 6 indicates how the developments performed for the different use case will be integrated in the general ECODE router architecture.

The next objective is to migrate the developed code on the ILAB platform, perform the full integration, and provide a complete performance and functionalities analysis.

8. References

[Barford 2002] P. Barford, J. Kline, D. Plonka, and A. Ron, “A signal analysis of network traffic anomalies”, In *Internet Measurment Workshop (IMW’2002)*, Marseille, November 2002.

[Cormode 2005] G. Cormode and S. Muthukrishnan, “What’s new: finding significant differences in network data streams”, *IEEE/ACM Transaction on Networking*, 13(6):1219–1232, 2005.

[Dewaele 2007] G. Dewaele, K. Fukuda, P. Borgnat, P. Abry, and K. Cho. Extracting, “Hidden anomalies using sketch and non Gaussian multiresolution statistical detection procedures”, SIGCOM *Workshop on Large-Scale Attack Defense (LSAD’2007)*, Kyoto, Japan, 2007.

[ECODE 2007] ECODE consortium, “ECODE: Experimental Collaborative Distributed Engine”, Technical annex of the ECODE project proposal, October 2007.

[ECODE 2009] ECODE consortium, “D3.1: Detailed experimental plan and scenarios”, Deiverable of the ECODE project, February 2009.

[Farraposo 2007] S. Farraposo, P. Owezarski, and E. Monteiro, “A multi-scale tomographic algorithm for detecting and classifying traffic anomalies”, IEEE International Conference on Communications, Glasgow, Scotland, 24-28 June 2007.

[Fernandes 2008] G. Fernandes, P. Owezarski, “Automated classification of network traffic anomalies”, LAAS Report No 08468, 2008.

[Fernandes 2009] G. Fernandes, P. Owezarski, “Automated classification of network traffic anomalies”, SecureComm’2009, Athens, Greece, 15-17 September 2009.

[Lakhina 2004] A. Lakhina, M. Crovella, and C. Diot, “Characterization of networkwide anomalies in traffic flows”, *Internet Measurement Conference (IMC’2004)*, Taormina, Italy, October 2004.

[Lakhina 2005] A. Lakhina, M. Crovella, and C. Diot, “Mining anomalies using traffic feature distributions”, *ACM SIGCOMM*, Philadelphia, August 2005.

[Li 2006] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, A. Lakhina, “Detection and identification of network anomalies using sketch subspaces”, Internet Measurement Conference (IMC’2006), Rio de Janeiro, Brazil, 2006.

[Mahoney 2003] M. Mahoney and, P. Chan, “An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection”, Recent Advances in Intrusion Detection (RAID 20003), pp. 220 – 237, September 2003.

[Netflow] Cisco Netflow(tm) version 5 packet format
http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/nfc/nfc_3_0/nfc_ug/nfcform.htm

[Owezarski 2005] P. Owezarski, “On the impact of dos attacks on internet traffic characteristics and QoS”, 14th IEEE International Conference on Computer Communications and Networks (ICCCN’2005), San Diego, CA, USA, 17-19 October 2005.

[Salamatian 2008] K. Salamatian, “An Introduction to non-Supervised Learning Techniques”, ECODE tutorial, Anvers, Belgium, september 2008.

[Scherrer 2007] A. Scherrer, N. Larrieu, P. Owezarski, P. Borgnat, P. Abry, "Non Gaussian and long memory statistical characterization for Internet traffic with anomalies", IEEE Transaction on Dependable and Secure Computing, Vol. 4, No. 1, pp 56-70, January-March, 2007.

Annex 1: Supervised and semi-supervised learning

A1.1: Supervised Learning

Principle

Supervised learning is able to establish rules from labelled training data. The training data are made of pairs of objects. Each pair is composed of a multi-dimensional values and the associated output. The predictive model obtained through the learning phase is able to give an output value from any multi-dimensional input value.

Semi-supervised learning consists of two steps. The first phase is supervised learning which is used to create a rough scheme of what the model is going to be. The second phase is meant to improve the result of the first phase through the use of unlabelled data. Supervised learning can be realized through different algorithms such as decision tree, support vector machines, etc.

Example of supervised learning through decision tree

We want to build a model to describe the behaviour of the golf players according to of several parameters: temperature, humidity and wind.

We choose to use the decision tree algorithm. Figure 2 give the set of training data that will be used.

Play golf dataset

Independent variables				Dep. var
OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	FALSE	Don't Play
sunny	80	90	TRUE	Don't Play
overcast	83	78	FALSE	Play
rain	70	96	FALSE	Play
rain	68	80	FALSE	Play
rain	65	70	TRUE	Don't Play
overcast	64	65	TRUE	Play
sunny	72	95	FALSE	Don't Play
sunny	69	70	FALSE	Play
rain	75	80	FALSE	Play
sunny	75	70	TRUE	Play
overcast	72	90	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	80	TRUE	Don't Play

Figure 2. Training data

The algorithm uses all the data to create the rules that model the behaviour of the player according to these parameters. The result is the following (cf. figure 3), a tree that describes the player behaviour.

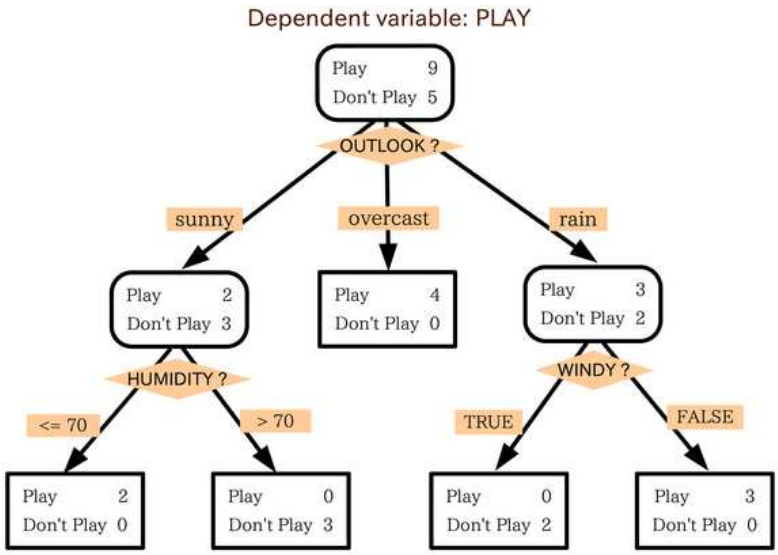

Figure 3. Model of the golf player behaviour

A1.2 Unsupervised Learning

Unsupervised learning aims at establishing how data is structured. In order to do so, unsupervised learning use unlabelled data and tries to find structures, key features or models that can describe the way the training data is organized and thus the way the system behaves.

There are different techniques of unsupervised learning among which we can quote dimensional reduction, density estimation, and clustering, etc. A priori, these techniques are the most suited for our use case.

Dimensional reduction tries to project the data from a set of great dimensions to a set with smaller dimensions. For example, PCA reduces the dimension of the dataset by replacing many correlated variables by a few ones. Let's quote few examples of methods of dimensionality reduction as for examples principal component analysis (PCA), multi-dimensional scaling (MDS).

Density estimation is a family of methods for "one-class" problems. We assume that there is a set of representative observations which is part of a single class. The general objective of this method is to estimate the distribution of the observations and then predict whether or not a new observation should be considered as an outlier or a "normal" member of the single class. For examples, Bayesian network, mixture network, etc. are density estimation methods.

Cluster analysis or clustering is the assignment of a set of observations into subsets (called clusters) so that observations in the same subset are similar in some sense. There are three types of clustering algorithms: hierarchical, partitional (e.g. k-means) and spectral.

Annex 2: Linear Estimation

In distributed anomaly detection we have frequently to deal with situations where we have access to a vector of random variables $\mathbf{Y} = (Y_1, ..., Y_l)$ that is correlated with another hidden vector $\mathbf{X} = (X_1, ..., X_m)$. We wish to find an estimator $\hat{\mathbf{X}} = f(\mathbf{Y})$) of $\mathbf{X}$ that minimizes the Mean Square Error defined as $\sum_e = E\{(\mathbf{X} - \hat{\mathbf{X}})^T (\mathbf{X} - \hat{\mathbf{X}})\}$. In general, the Mimimal Mean Square Error (MMSE) estimator of a random variable is the conditional expectation. In the case when $\mathbf{X}$ and $\mathbf{Y}$ are jointly Gaussian, the conditional expectation estimator can be easily derived and becomes simply

$$\hat{\mathbf{X}} = \sum_{\mathbf{XY}} \left(\sum_{\mathbf{Y}} \right)^{-1} \mathbf{Y}$$

Where, $\sum_{\mathbf{XY}}$ is the cross covariance matrix of $\mathbf{X}$ and $\mathbf{Y}$ defined as $\sum_{\mathbf{X}}$ and $\sum_{\mathbf{Y}}$ is the covariance matrix of $\mathbf{Y}$. It is noteworthy that the covariance's matrices $\sum_{\mathbf{XY}}$ and $\sum_{\mathbf{Y}}$ are finite size matrices that can be inferred using only a limited number of samples of $\mathbf{X}$ and $\mathbf{Y}$, when the above estimation formula can be used to infer all samples of $\mathbf{X}$. With this choice of estimator the MMSE of estimation (D^*) becomes equal to the trace of the error covariance matrix

$$D^* = tr\left(\sum_{\mathbf{X}} - \sum_{\mathbf{XY}} \left(\sum_{\mathbf{Y}} \right)^{-1} \sum_{\mathbf{XY}}^T \right)$$

A particular case of interest is when $\mathbf{Y}$ is a noisy projection of $\mathbf{X}$, $\mathbf{Y} = C\mathbf{X} + \mathbf{V}$ where C is a l x m projection matrix and $\mathbf{V}$ is a Gaussian noise vector with covariance matrix $\sum_{\mathbf{V}}$ that is independent of $\mathbf{X}$. Under this hypothesis the MMSE estimator becomes

$$\hat{\mathbf{X}} = \sum_{\mathbf{X}} C^T \left(C \sum_{\mathbf{X}} C^T + \sum_{\mathbf{V}} \right)^{-1} \mathbf{Y}$$

and the MMSE variance D^* is:

$$D^* = tr\left(\sum_{\mathbf{X}} - \sum_{\mathbf{X}} C^T \left(C \sum_{\mathbf{X}} C^T + \sum_{\mathbf{V}} \right)^{-1} C \sum_{\mathbf{X}} \right)$$

The above relations for the optimal estimator and the MMSE variance values raise some interesting comments. First the best estimator of $\mathbf{X}$ given $\mathbf{Y}$ is a linear projection of $\mathbf{Y}$ in a subspace determined by the cross-covariance $\sum_{\mathbf{XY}}$ and the covariance $\sum_{\mathbf{Y}}$. We will see later that this last observation can be generalized to more complex settings. Secondly, one can clearly evaluate the gain of using $\mathbf{Y}$ for estimating $\mathbf{X}$. If one did not have access to $\mathbf{Y}$ the variance of estimation would have been $\sum_{\mathbf{X}}$; now this value is reduced by $\sum_{\mathbf{XY}} \left(\sum_{\mathbf{Y}} \right)^{-1} \sum_{\mathbf{XY}}^T$, so that one can evaluate the gain obtained by sending a vector $\mathbf{Y}$ and balance it against the cost (in term of bandwidth) of sending $\mathbf{Y}$. A last interesting observation is that one can evaluate the impact of the added noise $\mathbf{V}$, into the overall estimation. We will see later that this noise is directly related to the quantization used for transmission of $\mathbf{Y}$ and with the bit rate transmission.

Annex 3: Source Compression

Very frequently, we have to apply a lossy compression to a sequence of n independent and identically distributed observations of a L-dimension random vector $\mathbf{X}=(X_1,...,X_m)$, *i.e.* we want to represent n observed vectors $\mathbf{X}[k], \quad k=1,...,n$ using only nR bits. These nR bits are used to reconstruct an approximation $\hat{\mathbf{X}}[k], \quad k=1,...,n$. The estimation error resulting from this estimation is:

$$D_n(R)=\frac{1}{n}\sum_{i=1}^{n}E\left\{\left|\mathbf{X}[k]-\hat{\mathbf{X}}[k]\right|^2\right\}$$

It is well known from the source compression literature when the multidimensional process $\mathbf{X}$ is Gaussian, an optimal compression scheme can be build by applying a Karhunen-Loeve Transform to the vector $\mathbf{X}$ followed by a quantization step. By optimal scheme we mean a compression scheme that results in a Mimimal Mean Squared Error (MMSE) for the given rate R.

The optimal local compression scheme consists of two stages: a projection stage where the state vector is projected linearly into an orthonormal space and a quantization phase that consists of assigning the bit budget to different projection dimensions following a water filling argument. The basic ingredient in finding the optimal compression scheme is the covariance matrix $\sum_{\mathbf{X}}$. This matrix is real, symmetric and positive semi-definite, and it has a Singular Value Decomposition (SVD):

$$\sum_{\mathbf{X}} = U\Lambda U^T$$

where U is a unitary matrix whose columns are the eigenvectors of the matrix $\sum_{\mathbf{X}}$ and Λ a diagonal matrix, whose diagonal entries are the eigenvalues λ_i of $\sum_{\mathbf{X}}$ in decreasing order. The matrix U^T maps the correlated vector $\mathbf{X}$ into a zero mean vector $\mathbf{Z}$ of uncorrelated components (that are also independent as $\mathbf{X}$ is assumed to be jointly Gaussian). The random component Z_i is a zero mean Gaussian random variable with variance $\mathrm{var}\{Z_i\}=\lambda_i$. By choosing just K columns of the matrix U, resulting in the matrix U^K, we can map the L-dimension vector $\mathbf{X}$ into a K-dimension orthonormal space with $\mathbf{Z}^k=U^k\mathbf{X}$ where $\mathbf{Z}^k$ contains the K first components of $\mathbf{Z}$. It can be easily shown that:

$$\mathbf{X}^k=\left(U^k\right)^T\mathbf{Z}^k$$

is an approximation of $\mathbf{X}$ with a MSE equal to $\sum_{k=K+1}^{L}\lambda_k$.

The second stage of the optimal compression scheme is quantization. As the components Z_i are uncorrelated and Gaussian with variance λ_i, they can be encoded separately. The quantization step consists in assigning bits from the budget of R. bits to each independent source. The bit assignment follows a water-filling technique defined as follows:

Let's define $L^*\le L$ as the largest integer satisfying

$$d_{L^*}\overset{\Delta}{=}2^{\frac{2R}{L^*}}\left(\prod_{i=1}^{L^*}\lambda_i\right)^{\frac{1}{L^*}}\le\lambda_{L^*}$$

The value L^* determines how many components of the vector $\mathbf{Z}$ are going to be sent. Now the number of bits k_i assigned to transmitting each component Z_i is derived as:

$$k_i = \frac{R}{L^*} + \frac{1}{2}\log_2 \lambda_i - \frac{1}{2^{L^*}} \sum_{j=1}^{L^*} \log_2 \lambda_j$$

This step results in a quantization noise for each chosen component that can be modeled by a zero mean Gaussian term V_i with variance d_{L^*}, the signal sent after projection and quantization can be represented as :

$$\mathbf{Z}^{L^*} = \left(U^{L^*}\right)\mathbf{X} + \mathbf{V}$$

where $\mathbf{V}$ is a Gaussian noise with a diagonal covariance. One can reconstruct/estimate the initial vector $\mathbf{X}$ by using linear estimation formula as:

$$\hat{\mathbf{X}} = \sum_{\mathbf{X}} \left(U^{L^*}\right)^T \left(\left(U^{L^*}\right) \sum_{\mathbf{X}} \left(U^{L^*}\right)^T + \sum_{\mathbf{V}} \right)^{-1} \mathbf{Z}^{L^*}$$

The MMSE resulting from this compression is obtained by the distortion-rate function $D(R)$ that defines the MMSE as a function of R

$$D(R) = L^* d_{L^*} + \sum_{j=L^*+1}^{L} \lambda_j$$

where d_{L^*} depends indeed on R.

The above analysis can be easily extended to minimizing a weighted MSE

$$D_n(\mathbf{w}, R) = \frac{1}{n} E\left[\left(\mathbf{X} - \hat{\mathbf{X}}\right)^T diag(\mathbf{w}) \left(\mathbf{X} - \hat{\mathbf{X}}\right) \right]$$

where $diag(\mathbf{w})$ is a diagonal matrix with weight values on it diagonal. This consists simply of applying the above analysis to the vector $\sqrt{diag(\mathbf{w})}\mathbf{X}$ where $\sqrt{diag(\mathbf{w})}$ is a diagonal matrix with square root of weights on its diagonal. In particular if we want to exclude a variable from the minimization it is enough to set its weight to 0.