Seventh FRAMEWORK PROGRAMME
FP7-ICT-2007-2 - ICT-2007-1.6
New Paradigms and Experimental Facilities

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D3.3

Experimental Evaluation of TO1

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-D3.3 Title: Experimental Evaluation of TO1 Responsible: INRIA Editor(s): Amir Krifa, Imed Lassoued, Chadi Barakat Contributor(s): Amir Krifa, Imed Lassoued, Chadi Barakat, Pedro Casas, Philippe Owezarski, Johan Mazel, Yann Labit Reviewer: Dimitri Papadimitriou Dissemination level: Public (PU) **Date of preparation:** ***31st May 2010*** **Version: 1.0**

D3.3 – Experimental Evaluation of TO1

Executive Summary

This deliverable is part of a series of 3 deliverables – one per technical objectives (TO) [ECODE2007] – which aims at describing the experimental evaluation and results of the different use case functions. More specifically, this deliverable addresses the evaluation and experiments performed in the framework of TO1 related to i) adaptive sampling and ii) cooperative intrusion and attack/anomaly detection

For each technical objective, the deliverable exposes the technical problems that need to be solved and outlines the techniques that have been developed. The proposed techniques have been introduced in the previous deliverable D3.2 titled "Design & Implementation of TO1". 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, advanced performance evaluations are detailed. At this stage, evaluation of the concepts and their implementation are done by means of simulation and/or small scale experiments (emulation most of the time, but can be also experimented in real environments).

This deliverable is organized as follows. Part 1 describes the use-cases main architectures and algorithms; part 2 describes the experimental facilities used towards the evaluation of the different use-cases. Finally, part 3 presents and discusses the evaluation results.

Table of Contents

List of Authors

Affiliation	Author
INRIA	Amir Krifa
INRIA	Imed Lassoued
INRIA	Chadi Barakat
LAAS/CNRS	Philippe Owezarski
LAAS/CNRS	Pedro Casas
LAAS/CNRS	Johan Mazel
LAAS/CNRS	Yann Labit
Univ of Lancaster	Kavé Salamatian

List of Figures

List of Tables

1. Introduction

This deliverable focuses on Technical Objective 1 (TO1) and presents the results of experimentations for the two use cases of this deliverable a1 and a3. Before exposing the results, the deliverable describes the technical problems that need to be solved and recalls the methods that have been proposed to handle these problems. 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, advanced performance evaluations are detailed. At this stage, evaluation of the concepts and their implementation are done by means of simulation and/or small scale experiments (emulation most of the time, but can be also experiment in real environments).

The deliverable is organized as follows. Part 1 describes the use-cases' main architectures and algorithms; part 2 describes the experimental facilities used towards the evaluation of the methods proposed within the different use-cases. Finally, part 3 presents and discusses the validation results.

2. Use case a1: Adaptive traffic sampling

2.1 System Architecture

Figure 1 depicts the basic functional components of the proposed monitoring system and the interactions among them. This system relies on local NetFlow-like measurement tools (Monitoring Engine (ME)) deployed in network routers as well as on the reporting capabilities for exchanging information and decisions with the central unit (Cognitive Engine (CE)). The targeted application provides the input to the system in the form of a set of attributes, for example the set of IP prefix to track, to apply on the collected data at the central unit (Machine Learning Engine (MLE)). 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.

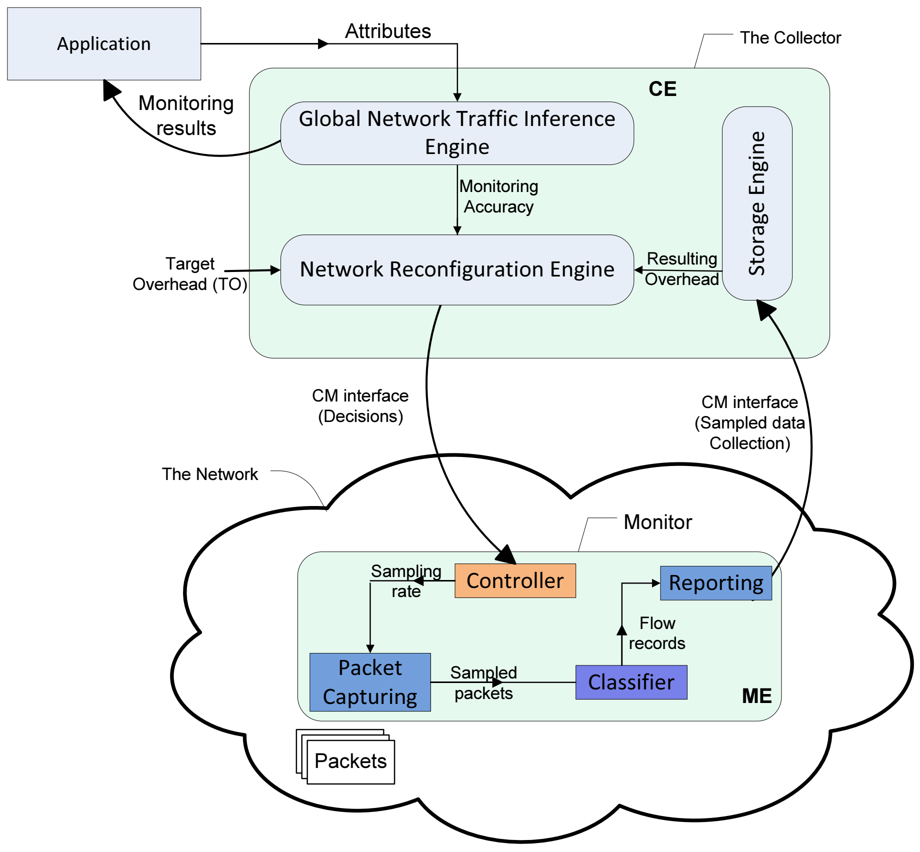


Figure 1: Architecture

2.2 Monitoring Engine (ME)

This engine comprise a set of monitoring points that run 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 samples data at a given sampling rate. This sampling rate is configured each time by the Machine Learning Engine (MLE) 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). 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 to the collector together with the statistics on those flows. A flow is declared finished by the classifier in one of three cases: (i) when observing a FIN or an 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 requiring to release memory.

- Reporting: once collected, flow records are exported using UDP messages to the Machine Learning Engine via the CM (Cognitive-Monitoring) interface.
- **Controller:** based on the collected data applies machine learning methods, the Machine Learning Engine takes a decision on how to tune the sampling rates and then sends it back to the ME in each router. The router controller receives the decision and updates its sampling rate accordingly.

2.3 Machine Learning Engine (MLE)

This component is motivated by the need to extend the local existing monitoring tools (MEs) with a network wide cognitive engine able to:

- Investigate the measurements collected from the different routers (local views) and then construct a global view of the traffic and the network state.
- Automate and enhance network-wide monitoring control 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 resulting overhead.

The MLE is composed of two main modules:

- Global Network Traffic Inference Engine: Given a measurement task T to realize, this inference engine investigates the local measurements made by the different routers to have a global and more reliable view. A typical task T would be the estimation of the volume of traffic between each pair of edge routers. The engine takes as inputs the sampling rate vector of network routers as well as the local estimations of $T,(\hat{T}_k)_{k=1..M}$, calculated from the reports sent to the collector by the different routers and stored in the Storage Engine. M is the number of router interfaces in the network. The inference engine then tries to combine the local estimators and to derive a better estimation of T. This combination is motivated by the need to minimize the variance of the global estimation error. For this purpose, we construct the global estimator of the task T as a weighted sum of the different local estimators. This weighted summation of local independent estimators is known to be the best linear combination in terms of mean square error [Duffield05]. Note that the summation is only done over monitors that have seen the traffic of interest. The weights are inversely proportional to the estimation error of the local estimators, which in their turn are inversely proportional to the configured sampling rate. Thus, local estimates with smaller error variance have a larger impact on the global estimator than those with larger errors.
- Network Reconfiguration Engine: To dynamically reconfigure the network monitors and in order to give a solution that allows optimizing measurement accuracies while respecting the monitoring constraints, we resort to a dynamics inspired from the one used by TCP for the adjustment of its congestion window. Starting from an initial sampling rate P_{init} for all interfaces, the Network Reconfiguration Engine is fed with the estimation

of task T and the variance of this estimation $VAR(\hat{T})$ from the Inference Engine, as well as the resulting overhead from the Storage Engine, i.e. rate at which flow records arrive. If the resulting overhead is less than the TO (Target Overhead), the system keeps increasing the sampling rates of the different monitors. Once the TO is reached, the system triggers decrease the sampling rates of the least significant monitors. In this way the system strives to keep the reporting overhead at TO flow records per second and fully profits from the available resources while seeking the best estimation of measurement task T. To increase or to decrease sampling rates, we use increments in the logarithmic scale in order to give more flexibility to our system. Another advantage of using the logarithmic scale is that sampling rate adjustments depend on the value of the sampling rate itself: small adjustments when the sampling rate is low and large adjustments when the sampling rate is high. The least significant monitors are identified using the GPM as already described. From the perspective of the task T, the least significant monitors are the ones providing the least increase in the variance of the estimator of T, i.e. $VAR(\hat{T})$, when the logarithmic of its sampling rate is decreased. To be identified, one has first to write analytically the expression of $VAR(\hat{T})$ as a function of the sampling rates in routers of interest then calculate the utility function of the different monitors by differentiating this expression with respect to log (p_k), where p_k is the sampling rate of the monitor k: $U_k = \partial Var(\hat{T}) / \partial \log(p_k), k = 1..M$. Given the current configuration of sampling rates, we choose the least significant monitors as being those having utility function values less than the average of the utility function value over all the monitors. In the next section, we provide an example of this control using an accounting application.

2.4 Case Study: Traffic Accounting

We explain in this section using a concrete example how the machine learning engine, based on the collected measurements, can decide on the way to tune the sampling rates over the network. For this purpose, we consider an accounting application: the estimation of the volume of some chosen network flows. 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.

2.5 Definitions

Consider N traffic aggregate (set of attributes) flows whose volumes in packets are labeled $F_1, F_2, F_3, ...F_N$. Denote by $\hat{F}_1, \hat{F}_2, \hat{F}_3, ...\hat{F}_N$ the corresponding estimators. Let P = (p_k) be the vector of sampling rates in the different monitors of the network. There are in total M monitors. The target of the system is to find the vector P that minimizes the sum of normalized estimation errors $\sum_i Var(\hat{F}_i) / F_i^2$.

Each aggregate flow F_i is formed of a set of 5-tuple (source IP, destination IP, source port, destination port, protocol) flows whose volumes are denoted by S_{ji} . Again, denote by $\hat{S}_{ji}$ the best estimator for the size 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 $\sum_i \sum_j Var(\hat{S}_{ji}) / F_i^2$.

As long as there are available resources, the monitors' reconfiguration engine periodically increases all sampling rates to improve results' accuracies. Once the TO value is reached the system triggers a decrease in the sampling rate of the least significant monitors. This continues until the overhead is again below this TO value. The least significant monitors are the ones having the smallest absolute values for the following partial derivation sum:

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

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. $F_i = \sum_j \hat{S}_{ji}$. Note that we consider the volumes of flows as measured in number of packets. The translation to bytes can be made by multiplying the size in packets by the average packet size, which we suppose true for large flows.

2.6 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). With this information, one can derive a first estimation for the flow size of the flow. The estimator that maximizes the likelihood is known to be [Duffield02]: $\hat{S}_{kji} = \frac{s_{kji}}{p_k}$. 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 flow has a variance equal to $Var(\hat{S}_{kji}) = S_{ji} * (1 - p_k) / p_k$.

2.7 Combining measurements

The information on a 5-tuple flow comes from all monitors along its path. Though, some of them may not sample any of the packets of the flow, either because their sampling rate is low, or because the volume of the 5-tuple flow is small in terms of packets. We propose to identify these monitors related to a 5-tuple flow with the help of routing information. Largely deployed link-state protocols like OSPF and IS-IS provide such information. If such routing information (the list of monitors on the path that should see the flow) is not available at the central unit, one has to limit the observations to monitors that have seen the flow taking into account that this might cause a bias against unsampled 5-tuple flows that got. However, this bias is expected to be small when aggregating over aggregate flows F_i.

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,

$$\hat{S}_{ji} = \sum_{k \in \varphi_{ji}} \lambda_k s_{kji} / p_k, \lambda_k = \frac{\frac{1}{Var(\hat{S}_{kji})}}{\sum_{l \in \varphi_{ji}} \frac{1}{Var(\hat{S}_{lji})}}$$

where φ_{ji} is the set of monitors on the path followed by S_{ji}. Replacing the variances by their expressions given in the previous section, substituting the second equation into the first one, and simplifying by S_{ji}, we get,

$$\hat{S}_{ji} = \frac{1}{\alpha_{ji}} * \sum_{k \in \varphi_{ji}} \beta_{kji} s_{kji}, \alpha_{ji} = \sum_{l \in \varphi_{ji}} \frac{p_l}{1-p_l}, \beta_{kji} = \frac{1}{1-p_k}$$

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 $Var(\hat{S}_{ji}) = \frac{S_{ji}}{\alpha_{ji}}$. The original flow size being unknown, we can simply substitute it by its global estimator $\hat{S}_{ji}$.

2.8 Reconfiguring monitors

As shown in the previous section, the variance (or mean square error) of 5-tuple flow size estimation is very important for the determination of the global system accuracy (note that we can increase the Target Overhead TO to satisfy a given threshold of measurements accuracy) and for the identification of the monitors that should be reconfigured. For 5-tuple flow S_{ji} and monitor k we can write,

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

This represents the marginal gain in the accuracy (loss in the variance) when the logarithm of the sampling rate of monitor k is increased by a small step and this is from the perspective of estimating the size in packets of flow S_{ji}. As expected, this gain is positive when someone increases the sampling rate p_k (more sampling means more accuracy). It also decreases when p_k increases, which suggest that the estimation error follows a continuously decreasing and convex function with the sampling rate, a condition required for the uniqueness of solution in non-linear optimization theory.

By using the above expression we obtain the utility function of the monitor k, which sums the accuracy and normalizes it over all 5-tuple flows forming the traffic of interest. Thus, the total gain (resp. the loss) in accuracy when the sampling rate of monitor k is tuned up (resp. down) by a multiplicative step (additive in the logarithmic scale) is given by the following expression:

$$\sum_i \sum_j \frac{-S_{ji} * p_k}{\alpha^2{}_{ji}(1-p_k)^2} * \frac{1}{F_i^2}$$

By testing all monitors, we can find the best sampling rates to tune down in case of saturation. We choose to decrease the monitors having utility function values less than the average over the different monitors. Note that the sum can be calculated online as long as more reports are received. The parameters α_{ji} can be calculated only once for each configuration. These parameters will be used for all possible paths across the network.

3. Use case a3: Cooperative traffic anomalies and attack detection

The use case a3 consists of two approaches followed separately by LAAS and Lancaster University. On one hand, LAAS targets to develop a two-steps technique to tackle both the anomaly detection and the anomaly classification problems. The technique is based on a clustering technique adapted to traffic anomalies and attack detection. On the other hand, Lancaster University tries to develop a cooperative anomaly detection scheme that is based on information exchange between monitor nodes. In this section these two efforts will be described.

3.1 System Architecture

Figure 2 depicts the basic functional components of the proposed distributed monitoring system and the interactions among them. This system assumes that a local Monitoring Engine (ME) deployed in network routers or any other monitoring boxes are capturing regularly (at fix time intervals for example) some metrics (like NetFlow measurements, or traffic volume related metrics).

We assume that metrics measured by the monitoring engine generates multidimensional (one dimension per metric measured) time series. We also assume that some of the network routers (or eventually all of them) have a MLE that will contain two components relative to our application:

1. A distribution component that has the responsibility of forwarding needed metrics or eventually an approximation of them to nodes that have demanded the metrics.
2. An anomaly detection component that will use information coming from the local ME as well as the information forwarded by other nodes by the forwarding task in order to detect anomalies occurring locally and eventually remotely.

The anomaly detection task will run only in nodes that have a need to detect anomalies, i.e. we might consider cases where only a single central node (master node) do anomaly detection using information coming from other nodes as well as scenarios where several anomaly detectors are distributed in the network. However the distribution task should be present more frequently.

We will consider the distribution task as a distributed application running in the MLE of participant routers that will run on a subset of routers (or eventually all). This application will sit in the MLE, as it will have to learn from all demands crossing that node which information (which projection of information) should be forwarded to other nodes. We are therefore considering a distribution overlay that consists of routers running the distribution task into their local MLE. These nodes are to be connected by an underlay consisting of normal routers.

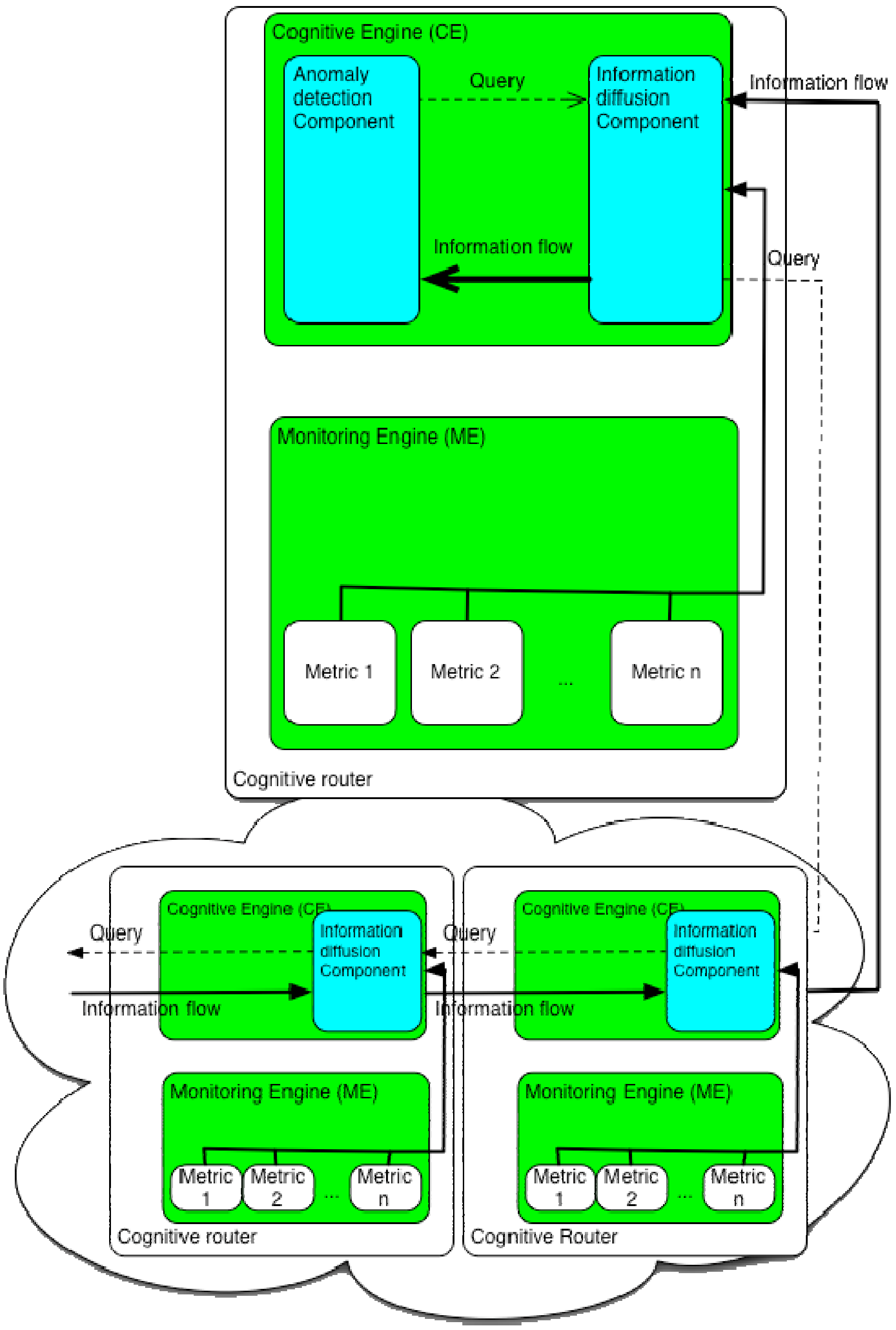


Figure 2: Architecture

The anomaly detection and the distribution task are interacting through a client-server relationship: the anomaly detector queries some remote metric of interest to the distribution component that will take care of retrieving this information for the anomaly detector and returning a flow containing the queried metrics at fixed time intervals. For this purpose the distribution component will launch queries to its neighbors' in the overlay. A neighbor node combines the queries it receives and creates new queries that will accommodate the demand with its own constraints and forward them to next steps. In the forthcoming we will describe with more details these components.

3.2 Information Distribution Component

The role of this component is to provide information from remote and local Monitoring engine to the local anomaly detector. This information might be directly observed and collected by the local Monitoring Engine (ME) or eventually received from remote distribution components. The remote information might be information coming from local ME at remote location, or information processed by an intermediate component (for example anomaly detection results obtained by another anomaly detection component on a remote router) at the remote location.

Indeed, a node would like to obtain the best approximation of remote variables. However this comes at the cost of consuming network resources, as processor, bandwidth and power resources that could be used for other more important purposes. We assume that the nodes in the network are selfish, *i.e.* a node wants to get the best approximation of the state of other nodes while giving away the smallest amount of information about its own state. This selfishness can result from several reasons; for example generating state messages and sending them is consuming processor, bandwidth and power resources that could be used for other more important purposes, or the node is secretive and does not want to give more precision about its state than needed. This means that we need an incentive/punishment cooperative mechanism to deal with node selfishness and to drive nodes to use their resource to exchange information.

We are assuming here that nodes are observing correlated information, as there will be no gain in sending non-correlated information to a remote point for anomaly detection. In other terms, exchanging information will not give any gain compared to detecting anomalies locally. For example two routers might observe locally the volume of traffic, the number of flows on each egress/ingress, the number of entries in the routing table, *etc*. These pieces of information are correlated as some part of the traffic crossing one router is crossing another one.

We have described in deliverable D3.2 an exchange mechanism based on the distributed Karhunen-Loeve transform. The scheme assumes that we have a network $(\boldsymbol{N}, \boldsymbol{E})$ with M nodes. Each node $\boldsymbol{i}$ observes at time $\boldsymbol{k}$ a vector of metrics (called in the forthcoming the state vector) $\mathbf{X}^i[k] = \left(X_1^i[k],\ldots,X_L^i[k]\right)^T$. All vectors are assumed to be column vectors. The estimated value of the state vector of node $\boldsymbol{i}$ at node $\boldsymbol{j}$ (the value received from the distribution component at node $\boldsymbol{j}$ about a queried metric at node $\boldsymbol{i}$) 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.

The main idea of the distribution approach developed in D3.2 is the following (we refer the reader to D3.2 for complete details) instead of transmitting the L values of the state vector $\mathbf{X}^i[k]$ at each time step, we transmit linear projection of it $\mathbf{Y}^i = C^i\mathbf{X}^i + \mathbf{V}^i$, where $\mathbf{Y}^i[k] = \left(Y_1^i[k],\ldots,Y_K^i[k]\right)^T$ with $K<L$ and $\mathbf{V}$ being a random error vector representing the truncation error (when we represent a projection value with a finite number of bits). When a node receives truncated linear projection, it can retrieve an estimate $\hat{\mathbf{X}}^{i,j}[k]$ of the initial state vectors $\mathbf{X}^i[k]$ using classical techniques from linear estimation theory (described fully in deliverable D3.2) as

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

where $\sum_{\mathrm{X}}$ are the covariance of the local state variable and $\sum_{\mathrm{V}}$ the covariance of the truncation noise. Moreover, we can give an expression for the resulting estimation error as

$$D^{*} = tr\left(\sum_{\mathrm{X}} - \sum_{\mathrm{X}} C^{T}\left(C\sum_{\mathrm{X}} C^{T} + \sum_{\mathrm{V}}\right)^{-1} C\sum_{\mathrm{X}}\right) \qquad (2)$$

Sending the linear projection in place of the value results in two outcomes:

1- The amount of data sent per time step is reduced as we send K truncated values in place of the initial L values. By setting the dimension of the projection K and the number of bits used for representing each projection one can control the compression ratio.

2- By sending the linear projection, we give a negotiation tool to the selfish node. If one sends directly the state vector $\mathbf{X}^{i}[k]$, the node will be left with an all or nothing choice; or all the available data is given or no data is given. With linear projection, the node can implement a full palette of cooperation scheme going from giving all the data to its counterpart to giving nothing and passing by intermediate points, where the node gives an approximation of its state. This last point is also of interest when the node has to mitigate concurrent transmission demands and to fit them into a given transmission constraint.

We developed in document D3.2 a full method for determining the optimal projection matrix C^{i}, and the dimension of the projection K. We also provide the number of bits used for representing each projection in order to fit into an overall transmission budget of K bits per time step while attaining a minimal mean squared error on the estimated states. The optimal projection depends indeed on the correlation between the states at the remote node with the states at the receiving node; if the correlation is high the amount of needed information to send is small and *vice-versa*.

Nevertheless, the scheme presented in deliverable D3.2 was generic and worked for the case where all nodes where interested to know about the state of all other nodes. In the case of interest in ECODE project we wish to let the node to define what his states of interest are and to give to them a relative weight. We wish also this query to interact with intermediate node selfishness and make possible the node to mitigating between different queries he is receiving.

In the forthcoming, we will describe how to achieve this query behavior. In the general situation, we are assuming that a node i has access to two types of data: its own state vector condensed in the vector $\mathbf{X}^{i}[k]$ and the projection sent by other nodes about their own state vectors represented as Y^{i}. The node has to forward some information to its neighbors (in the real topology or in an overlay) validating a transmission rate constraint. The main issue is that if one does not take care, the length of Y^{i} increases and because of the rate constraints the precision about the state variables decreases, or equivalently the amount of noise introduced into the projection processing. We have therefore to develop a scheme to control the propagation of noise resulting from the joint effect of the increase in the amount of information to forward and the forwarding rate constraints. We will assume that each node i have a noise limit $D_{\max}^{i}$ that represents the amount of overall noise energy (coming from the sum of mean squared error of variables passing by this node) that the node can accept. A node will not be able to accept state variables that will drive its overall MSE beyond $D_{\max}^{i}$.

We assume a node that wish to access to remote states generate a query in form of a list of state variables L^{i} that it is interested in receiving with the highest precision possible. However, as it might not be possible to fit the query of all nodes with the bandwidth constraints, so the node along with its wish list provides a weight vector $\mathbf{W}^{i}\left(\mathrm{L}^{\mathrm{i}}\right)$. In summary, a node i stores 3 vectors:

- A vector of local states $\mathbf{X}^{i}$ with realizations $\mathbf{x}^{i}[k]$. The vector $\mathbf{X}$ contains all local states observed up to time k in the network.
- A vector of state preferences L^{i} along with a weighting vector $\mathbf{W}^{i}\left(\mathrm{L}^{\mathrm{i}}\right)$ describing the relative importance of the different state values in the list. A vector of estimated variance $V\left(\mathrm{L}^{\mathrm{i}}\right)$ for the elements in the list is also maintained along with a value $D_{\max}^{i}$ that represents the maximum variance the node can afford.
- A vector of received projections Y^{i}. We assume that the projection matrix $\boldsymbol{C}^{i}$ used for generating Y^{i} as well as the characteristics of the quantization noises $\operatorname{var}\left\{\mathrm{V}^{i}\right\}$, $\mathrm{Y}^{\mathrm{i}}=\boldsymbol{C}^{i}\mathbf{X}+\mathrm{V}^{\mathrm{i}}$ are known at node i.

The node broadcasts periodically to his neighbors its list L^{i}, its associated weights $\mathbf{W}^{i}\left(\mathrm{L}^{\mathrm{i}}\right)$, along with the estimated variance of the members of this list $V\left(\mathrm{L}^{\mathrm{i}}\right)$. Using this information, the neighbor can estimate the overall noise energy at node I as $D^{i}=\mathbf{W}^{i}\left(\mathrm{L}^{\mathrm{i}}\right)^{T}V\left(\mathrm{L}^{\mathrm{i}}\right)$. Moreover, whenever the projection matrix $\boldsymbol{C}^{i}$ used for forwarding changes, its new value is forwarded to its neighbors as well as the new parameters of the associated noises $\operatorname{var}\left\{\mathrm{V}^{i}\right\}$. We assume moreover that a node has a bandwidth constraint R on the amount of information that it can forward.

3.2.1 Incentive for selfish nodes

In a well-behaved world, a node will forward to its neighbors linear projection that will help him estimating the needed set of state variables, ensuring that the overall variance will not go beyond $D_{\max}^{i}$. However, as said previously, the node might be selfish; so, they might be inclined toward forwarding less data than needed or even worst not forwarding anything. We have therefore to ensure there is a benefit for the nodes to cooperate with their neighbors by forwarding information helping them for inferring their needed information. In a nutshell, the solution consists in systematically sending to neighbor's projections involving our estimates of the state variables in L^{i} along with the variables in the neighbors list. At a neighbor node j, the MSE of the variables in L^{j} depends therefore on the MSE of the estimate of variables in L^{i}; nodes have an incentive to improve the estimation of their neighbors, as it will help to improve their own estimation.

3.2.2 Reception processing

A node i receive from its neighbors in information diffusion overlay the projections resulting from combining the neighbor's state vectors and the projections received from the 2-hops neighbors. The received projections fill the vectors Y^{i}.

We have assumed also that the projection matrix used for generating the projections as well as the quantization noise variances are also known by node *i*. This knowledge results from exchange of control messages that are broadcasted to neighbors anytime there is change in these parameters.

Upon reception of a message from its neighbors the node has to apply four types of processing to it:

- Extracting node and state variable IDs used in the message and assigning each received value to the correct variable. This step essentially consists of parsing the received packet to find the bits relative to each specific variable.
- Whenever a new projection or a new remote state variable is observed, the node has to update the vectors Y^i and the data structure it uses for storing the projection matrix $\boldsymbol{C}^i$. The node has also to update its preference list $\boldsymbol{L}^i$. We will discuss this point further.
- Estimating the covariance $\sum_{\mathbf{X}}$ of matrix $\mathbf{X}$ that is needed for estimating the values of the states from the projections. This point will also be discussed further.
- Knowing $\sum_{\mathbf{X}}$, the estimation of remote state variables proceed following formula (1) and the error resulting from it can be estimated using formula (2)

Estimation of the covariance of $\mathbf{X}$ can be done using the following approach. The element of this covariance matrix would fall in one of three cases:

1. The covariance is between local state variable in node *i*: this covariance can easily be obtained at node *i* as all information are local.
2. The covariance is between state variables in node *i* and remote node *j*: in this case let's define $\mathbf{Y}^j = C^j\mathbf{X}^j + \mathbf{V}^j$ the projection received from node *j*. Let's $\sum_{\mathbf{X}^i\mathbf{Y}^j} = E\left\{\mathbf{X}^{i^T}\mathbf{Y}^j\right\}$, one can estimate $\sum_{\mathbf{X}^i\mathbf{X}^j}$ as:

$$\hat{\sum}_{\mathbf{X}^i\mathbf{X}^j} = C^j \sum_{\mathbf{X}^i Y^j} \qquad (3)$$

   where $\sum_{\mathbf{X}^i\mathbf{Y}^j}$ can be calculated as $\mathbf{X}^i$ and $\mathbf{Y}^j$ are available at node *i*.
3. The covariance is between state variables in remote nodes: in this case one can estimate $\sum_{\mathbf{X}^k X^j} j,k \neq i$ as:

$$\hat{\sum}_{\mathbf{X}^j\mathbf{X}^k} = C^{j^T} \sum_{Y^j Y^k} C^k \qquad (4)$$

3.2.3 Query processing

The way the preference list L^i is updated, *i.e.* the query is propagated depends on the particular application of the node state exchange. We will present here a generic updating mechanism. Let's assume that at time $k=0$ the node *i* introduces in L^i the IDs of the nodes and state variables it wishes to get and assign to them high weight. This query list is forwarded to neighbor nodes. After reception of its neighbor's preference list, node *i* updates its preference list by merging it with the neighbor preference list; however, a lower weight is assigned to the new values. This inclusion is done as the variables needed by neighbors can be bartered later with the variables needed by node *i*. This new preference list is periodically forwarded to neighbor nodes.

However, in order to ensure that the preference list is not lengthening without bound, we are using the maximum variance $D^i_{\max}$. We have to ensure that the weighted sum of the variance of elements in the preference list does not go beyond $D^i_{\max}$, *i.e.* $D^i = \mathbf{W}^i\left(\mathsf{L}^{\mathrm{i}}\right)^T V\left(\mathsf{L}^{\mathrm{i}}\right) < D^i_{\max}$. This means that increasing the length of the preference list is done at the cost of loosing acceptable precision on all variables in the preference list. A node might therefore prefer to exclude some elements from its preference list in place of degrading the overall quality of its needed variables. This would happen in particular if a particular variable is not anymore in the preference list of neighbors and if this variable is not of interest for the node itself. Another setting that leads in the retraction of a variable from the preference list is when a particular variable have a very high correlation (close to 1) with all its neighbors[1]. It is noteworthy that the weight of a particular variable in the preference list can be adjusted in order to improve its quality of estimation and to increase its bartering values.

3.2.4 Forwarding scheme

At time *k*=0 each node *i* has just its local state vector, the received projections vector Y^{i} is empty as well as the vector **X**. After reception of a preference list from neighbors where one of our local variables is included, the node first implements a local compression with the constraint rate *R* and forward an approximation of its own state vectors that are in the neighbors queries. The node send along with this transmission a node ID, the projection matrix resulting from the local Karhunen-Loeve Transform, and a vector containing the variance of quantization noise.

With reception of preference list variables from the neighbors, the node will forward variables (or estimates of variables) from the preference list of neighbors weighted by their preference weights, as well as the estimate of some variables from its own preference list. The inclusion of the variables from its own preference list is to give an incentive to the neighbors to help the node *i*; if node *i* has a bad estimate of the variable in its preference list, it will introduce more noise in the projections sent to its neighbors. If we have not yet received any information about a variable in our preference list, we will send a zero value.

The scheme will have in fact two major phases: a first phase of propagation of preference list and preference weights that terminates when the preference reaches a neighbor of the source maintaining the state variable of interest, followed by a second phase of propagation of the variable of interest that will follow the inverse diffusion path.

After estimating the matrix $\sum_{\mathsf{X}}$ node *i* can implement the distributed compression scheme described in report D3.2 and derive the global projection matrix that integrates correlation between nodes and send only projections relevant to the variables in the preference list of neighbors. As the nodes will choose asynchronously their projection matrix, the node needs to recalculate its projection matrix every time it receives a new projection matrix from its neighbors. This recalculation of projection matrix implements the iterative

[1] The correlation of variables is derived from matrix $\sum_{\mathsf{X}}$

algorithm suggested in the discrete Karhunen-Loeve Transform. This iterative algorithm is proven to converge to the optimal projection.

After a transient period the preference lists become stable and no exchange is anymore needed. Moreover, the projection matrices converge to a stable point and there is not anymore the need for sending them. In Steady state, the node applies the converged projection to calculate the projected values to be exchanged with the neighbors.

One of the main aims of the proposed scheme is to reduce the volume of information needed to spread the node state information. We achieve this by implementing relatively complex processing. However, the compression scheme needs to send parameters in addition to the projections. One has to evaluate the overhead resulting from the transmission of these parameters and to check if it is not killing the purpose. We will provide here an evaluation of this overhead.

The overhead coming for the exchange of the preference lists is resulting from the hypothesis that the node has no topology knowledge. Any scheme that has to discover the topology will have such an overhead. This means that transmitting the projection matrices is the main overhead. However, as explained above the projection matrices converges to a stable point and they do not change afterward, meaning that the overhead resulting from this exchange become negligible with time.

The above analysis is done assuming the state variables are stationary. However in a real network one can expect to observe changes in the distribution. Whenever such a change happens the projections have to be recalculated and the scheme returns back in the transient state. However, one can expect that the time between changes in the distribution is much larger than the transient time and the overhead effect stays negligible.

In practical settings, we have observed that the overhead induced by exchanging the projection matrices is covered in the order of a minute by the reduction in the transmitted data, meaning that the use of the overhead is largely balanced with the benefit.

Another point of interest is related to convergence in realistic settings. Indeed, the proposed iterative scheme is proved to converge to a steady state. However this proof is based on an assumption of perfect knowledge of the cross-correlation between the exchanged variables. However in practice we have not such a perfect knowledge and we have to resort to estimating the correlation. This ends up in some instability in the convergence. Nevertheless, by setting a threshold on the amount of variation in total noise energy between two iteration one can easily set a termination rule that his stable enough for realistic operation of the scheme.

3.3 Joint Anomaly Detection and Classification

In deliverable D3.2 we presented NewNADA, a two-steps algorithm to tackle both the anomaly detection and the anomaly classification problems, providing a flexible framework for network operators to understand and manipulate the anomaly treatment process.

The first step of the algorithm consists in anomaly detection and information retrieval based on absolute deltoids [Cormode 2005] of volume traffic metrics and minimum sets search. The second step consists in anomaly classification based on traffic attributes signatures and filtering rules [Fernandes 2009].

The classification of anomalies requires labeled data of already known anomalies to produce very specialized signatures for low misclassification. However, this approach is not appropriate to detect 0-day anomalies, for which we cannot have labeled data by definition.

As we explained in deliverable D3.2, we have integrated a new machine learning module into NewNADA to detect unknown anomalies and to automatically produce new filtering rules for these 0-day anomalies. Additionally, the module can be used to update already known signatures, producing soft instead of static signatures. This module is based on unsupervised learning techniques, more precisely, based on clustering techniques and clusters assessment.

The algorithm that we present takes as input a set of unlabelled traffic data and attempts to find unknown anomalies buried within the data. The proposed method does not assume any anomaly signature or particular model for anomaly-free traffic, which allows for detection of previously unseen attacks. After an anomaly is detected, we tackle the problem as a classification supervised learning problem (i.e. labeled data), selecting the best traffic attributes to automatically produce filtering rules (i.e. signatures).

Our algorithm makes three assumptions about network traffic: (i) The majority of the network traffic corresponds to normal-operation traffic [Portnoy01], (ii) the attack traffic is statistically different from normal-operation traffic [Denning87], and (iii) it is possible to find a traffic aggregation in which anomalous traffic lies in small-size clusters. Instead of detecting anomalies based on outliers detection [Eskin02, Leung05], we will find anomalies based on small-size clusters detection. This is extremely important to accurately define robust filtering rules. In simple words, it is impossible to define a robust anomaly signature based only on outliers, which are by definition isolated instances. If any of these three assumptions fail, the performance of the algorithm will deteriorate.

3.4 Data Clustering for Anomaly Detection

Data clustering represents a very challenging problem. The objective of clustering is to partition a set of unlabelled objects into homogeneous groups or “clusters” with similar characteristics. The notion of cluster is generally defined in terms of dense regions: clusters are dense regions in the data space that are separated by regions of lower density.

While hundreds of clustering algorithms exist [Jain99, Jain10, Duda01, Kaufman90, Everitt93, Theodoridis99], it is extremely difficult to find a single clustering algorithm that can handle all types of cluster shapes and sizes, or even decide which algorithm would be the best one for a particular data set [Dubes76, Fraley98]. Different clustering algorithms produce different partitions of data, and even the same clustering algorithm produces different results with different initializations and/or different algorithm parameters. This is in fact one of the major drawbacks in current cluster analysis techniques: the lack of robustness.

To circumvent the limitations of current clustering techniques, we integrate in our algorithm a multiple clustering combination approach, via the notion of “clustering ensemble” [Strehl02]. A clustering ensemble consists in a set of multiple partitions produced for the same data. The idea is to combine the multiple **partitions** or clustering’s into a single data partition, obtaining better quality and more robust clustering results. Each partition of the clustering ensemble provides an independent evidence of data organization, which can be exploited to find a proper separation between normal traffic and anomalies.

There are many different ways of generating a clustering ensemble. For example, multiple data partitions can be generated by using different clustering algorithms, or by applying the same clustering algorithm with different values of parameters and/or initializations. There are also many different ways of combining the information provided by the different partitions. It is interesting to see that the clustering ensemble approach can take advantage of the lack of robustness of a certain algorithm to produce better results.

In our particular problem of unsupervised anomaly detection, we generate multiple partitions by changing the feature space used to represent the data, applying the same clustering algorithm on each of the modified spaces. Remember that the feature space is simply the space generated by the different traffic attributes used to describe the traffic. Our algorithm falls within a more general field of clustering, known as Sub-Space Clustering [Parsons04].

As we discussed in D3.2, any modification that we shall make to the feature space must keep “human interpretable” and tractable attributes, easing the traffic analysis instead of obscuring it. For this reason, the modification of the feature space simply consists in constructing sub-subspaces of the original space, considering combinations of k features from the d features that compose the complete feature space. The number of features $k < d$ represents the dimension of each feature sub-space, and it basically permits to dig the feature space for unknown anomalies with different depths.

This sub-space clustering approach has an extremely beneficial side-effect: by taking small values of k, the clustering is performed in low-dimensionality spaces, reducing computational cost and getting around the “curse of dimensionality” problem. When the data is high-dimensional, the feature space is usually sparse, making it difficult to distinguish high density regions from low density regions. Briefly, sub-space clustering algorithms overcome this limitation by finding clusters embedded in low-dimensional subspaces.

3.5 Sub-space Clustering and Multiple Clustering Combination

Without loss of generality, let $X = \{x_1, x_2,…, x_n\}$ be the representation of the n traffic objects or patterns $Y = \{y_1, y_2,…, y_n\}$ found in the detected anomalous slot (detected by NewNADA). We use the generic term pattern because its nature will depend on the level of aggregation used in NewNADA. For example, y_i can represent all the IP packets with a particular IP network destination address IPdsti, or all the packets with the same origin IP address. Each vector $x_i \in R_d$ is a d-dimensional vector of attributes that describe y_i, taken from the list of attributes previously described in D3.2. These d attributes define the feature space previously mentioned.

Any general clustering algorithm takes X as input and organizes the n patterns into j different clusters, forming a data partition P. In multiple clustering combinations, we generate N different partitions of the same n patterns, building a cluster ensemble $P = \{P_1, P_2,…, P_N\}$. Producing a clustering ensemble leads to an exploration of distinct views of inter-pattern relationships. From a computational perspective, multiple partitions produced in an independent way facilitate efficient data analysis.

In our algorithm, the N partitions are generated by applying a particular clustering algorithm to each of the sub-spaces X_i c X that result from the combinations of k attributes taken from the d original attributes.

Each partition Pi is obtained by applying DBSCAN [Ester96] to the sub-space Xi. DBSCAN is a powerful density-based clustering algorithm, which permits to accurately discover clusters of arbitrary shapes [Jain10]. In this context, N is simply the number of k-combinations obtained from d:

$$N(d,k) \;=\; C_k^d \;=\; \frac{n!}{k!(n-k)!}$$

Each vector xj ϵ Xi is now a k-dimensional vector of attributes that describe y_j, which provides a clear advantage from the clustering side: as we will generally consider small values of k, i.e. k << d, we shall be working with low-dimensional data. Density-based clustering algorithms such as DBSCAN provide better results in this case [Agrawal98]. However, this poses a clear trade-off between data dimensionality d, sub-spaces dimension k, and computational cost. The bigger the value of d and the lower the value of k, the bigger the cluster ensemble P becomes. This trade-off may render the problem computationally infeasible, but for the values of d and k that we are working on, this is not really an issue. For example, if we consider the complete list of attributes defined in D3.2, we have an initial dimension d = 20. To set the value of k, we take into account a very useful property of monotonic in clustering sets, known as the downward closure property: "if a collection of points is a cluster in a k-dimensional space, then it is also part of a cluster in any (k-1) projections of this space" [Agrawal98]. This directly implies that, if there exists any evidence of density in the data Y, we are sure that this evidence will be present in low-dimensional spaces. For this reason, we have decided to use a small value of k, usually k = 2. For d = 20 and k = 2, the number of partitions N is equal to 190, i.e., a reasonably small cluster ensemble P. In addition, doing clustering in low-dimensional data is more efficient and less time-consuming than clustering in bigger dimensions, which partially reduces the increased cost of multiple clustering. Even more, the computation of multiple partitions can be done in parallel, which certainly speeds-up analysis.

3.6 Evidence Accumulation for Anomaly Detection

In [Fred02, Fred05], authors introduced the concept of Evidence Accumulation Clustering (EAC). EAC is a multiple clustering combination algorithm that combines the clustering results of the multiple partitions P_i into a new single partition P*, using each clustering result as an independent evidence of data organization. EAC summarizes the inter-pattern structure perceived from the multiple partitions in P into a new similarity measure between patterns. In this sense, we expect that the new data partition P* will better explain natural groupings of the patterns, compared to the individual clustering results.

Given n patterns Y = {y_1, y_2,…, y_n}, the EAC algorithm follows a split-combine-merge approach to discover the underlying structure:

- **Split**: in this step, the d-dimensional representation of Y, namely X, is independently split in N consecutive clustering runs, building the N partitions P_i that compose P. In [Fred02, Fred05], authors use the K-means algorithm to perform this decomposition, producing the N partitions of X by random initializations and selecting randomly the number of clusters K to produce in each clustering run. As we explained before, in our algorithm we apply DBSCAN to the N different feature sub-spaces X_i. The reader should note that in our case, we make no assumptions on the number of clusters K in each data partition P_i, which represents an important advantage with respect to [Fred02, Fred05].
- **Combine**: in order to cope with partitions containing different numbers of clusters, we use a voting mechanism to combine the clustering results, leading to a new measure of similarity between patterns. The underlying assumption is that patterns belonging to a "natural" cluster are very likely to be co-located in the same cluster in different partitions. Taking the co-occurrences of pairs of patterns in the same cluster as votes for

their association, the N partitions are mapped into a (n∗n) co-association matrix C:

$$C(i,j) = \frac{n_{ij}}{N}$$

where n_{ij} is the number of times the pattern pair (y_i, y_j) is assigned to the same cluster through the N partitions P_i. The evidence accumulation mechanism thus maps the partitions in the clustering ensemble P into a new similarity measure between patterns (summarized in the co-association matrix C), intrinsically performing a non-linear transformation of the original feature space X into a new representation.

This voting mechanism for evidence accumulation can be improved for our particular task of unsupervised anomaly detection. In fact, according to our assumptions, traffic anomalies lie in small-size clusters. In this sense, it would be beneficial to assign a different weight to the vote that a pattern pair (y_i, y_j) receives, taking into account not only the membership to the same cluster, but also the size of the cluster where both patterns lie together.

Given a certain partition P_i formed by k=1,…,K clusters, and assuming that n(k) is the number of patterns inside cluster C_k, we define the evidence importance function w_k(n(k)) as a weighting function that takes bigger values for small values of n(k), and tends to zero for large values of n(k). Using this function, we modify the voting mechanism described before, multiplying the assigned vote by wk when two patterns are found in the same cluster C_k of size n(k). We shall use the following pseudo-code of the evidence accumulation voting mechanism to better explain this simple idea:

```
1:  **Initialization:**
2:      (a) Set co-association matrix $C$ to a null $n \times n$ matrix.
3:      (b) Set minimum cluster size $n_{\min}$.
4:      (c) Set evidence weighting factor $\gamma$.
5:  **for** $t = 1 : N$ **do**
6:      $P_t$ = DBSCAN $(X_t, n_{\min})$
7:      Update $C(i,j)$, $\forall$ pattern pair $(\mathbf{y}_i, \mathbf{y}_j) \in Cluster_t(k)$:
8:          (a) $w_k \leftarrow e^{-\gamma \frac{(n_t(k) - n_{\min})}{n}}$
9:          (b) $C(i,j) = C(i,j) + \frac{w_k}{N}$
10: **end for**
11: Compute a similarity matrix $S$ from $C$:
12:     $S \leftarrow 1 - C$
```

Figure 3: Evidence Accumulation for Unsupervised Anomaly Detection

The parameter n_{min} simply specifies the minimum number of patterns that can be classified as a cluster by the DBSCAN algorithm. The evidence weighting factor permits to set the slope of wk, but for us it will be a fixed parameter.

After going through the complete clustering ensemble, the final task of the Combine step consists in transforming the co-association matrix C into a similarity matrix S, introducing an idea of proximity between patterns. Noting that each element in C lies in the interval [0,1], the similarity matrix S is simply the complement of C:

$$S(i,j) = \begin{cases} 1 - C(i,j), & i \neq j \\ C(i,j), & i = j \end{cases}$$

- **Merge**: the core of the EAC technique is the mapping of partitions into the similarity matrix S. We can now apply any clustering algorithm over this new similarity matrix in order to find a consistent data partition P*. In [Fred02, Fred05], authors apply a minimum spanning tree (MST) algorithm, cutting weak links at a pruning threshold t_{prun}. This is equivalent to cutting the dendrogram produced by the hierarchical single-link (SL) clustering method over the similarity matrix S at threshold t_{prun}.

  A well known difficulty of the SL method is its quadratic space and time complexities, related to the processing of a (n*n) similarity matrix. To circumvent this, we use the similarity information provided in S, coupled with the idea of what we are looking for: a small cluster. As we have assumed, if there is an anomaly present in Y, it must be represented by a small-size cluster with a well defined structure. If this is not the case, we claim that it is always possible to find a traffic aggregation with NewNADA in which this is true.

  Thus, the detection of an anomaly consists in finding a small and compact cluster, and this can be achieved by looking into S for those pattern pairs (y_i, y_j) that have the smallest dissimilarity. In other words, we look for those pattern pairs with the lowest values in S.

3.7 Automatic Rules Generation

Selecting the best attributes for sub-space clustering is a difficult task. Many papers in the literature attempt to do so by using different search heuristics [Liu98] (greedy forward and greedy backward, pruning, bottom-up, iterative top-down, etc), additionally using some measure of goodness of clustering to assess the relevance of a particular feature. In this sense, they generally adapt the feature selection problem [Liu98], usually directed to the field of supervised learning, to the case of unsupervised clustering.

In our case, we do not intend to perform attributes selection for the sub-space clustering step, but for automatically generating filtering rules that permit to clearly identify the anomalous cluster detected by EAC. The basic idea is to produce robust filtering rules that can be further integrated into a signature-based detection system, so as to detect in the future the new anomalies detected by our unsupervised approach.

We follow a similar approach to those proposed in the literature of feature selection, using the already generated clustering ensemble P. Basically, we select those partitions Pi in which the anomalous cluster is clearly isolated from the rest of the traffic patterns.

We shall define two different types of filtering rules: absolute rules and splitting rules. Absolute rules do not depend on the relative separation between clusters. This kind of rules corresponds to the presence of dominant attributes in the patterns of the anomalous cluster. For example, if one of the attributes in X corresponds to the strong presence of SYN packets in a certain traffic aggregation specified for Y (e.g., 90% of the packets are SYN packets), it is likely that in the detection of a SYN port scan attack, the majority of the patterns in the anomalous cluster will have a value equal to 1 for this attribute. Absolute rules are then rules of the type <attribute == value>.

On the other hand, splitting rules consist in isolation rules that depend on the relative separation between clusters. Briefly speaking, if the anomalous cluster is well separated from the rest of the clusters in a certain partition P_i, then the attributes of the corresponding sub-space X_i are good candidates for defining a filtering rule. Splitting rules are rules of the type <attribute > threshold> or <attribute < threshold>.

Absolute rules are important rules, because they define inherent characteristics of the anomaly. As regards splitting rules, it is clear that some of them will be more important than others, based on the degree of separation between clusters. In order to assess the importance of splitting rules, we use the notions of Linear Discriminant Analysis (LDA), through the computation of the Fisher Score. Given two clusters C_1 and C_2, where C_1 corresponds to the anomalous cluster and C_2 to the closest cluster in a particular dimension defined by attribute i, the Fisher score F(i) can be computed as follows:

$$\begin{aligned}
F(i) &= \frac{(\bar{x}_1(i) - \bar{x}_2(i))^2}{\sigma_1(x(i))^2 + \sigma_2(x(i))^2} \\
\bar{x}_k(i) &= \frac{1}{n(k)} \sum_{x(i) \in C_k} x(i) \\
\sigma_k(x(i)) &= \sum_{x(i) \in C_k} (x(i) - \bar{x}_k(i))^2
\end{aligned}$$

where $x_k(i)$ is the value of attribute i for the patterns that belong to cluster C_k, and n(k) is the size of the cluster.

The Fisher score is a measure of the separation between clusters, relative to the total variance within each cluster. A big value of F(i) means that both clusters are well separated in the direction of attribute i. Therefore, in order to select the most important splitting rules, we shall keep those features with largest Fisher score.

4. Experimental setup

4.1 Use case a1: Adaptive traffic sampling

We choose to validate our architecture over a real platform that we developed for the purpose of the study. This platform detailed in [Krifa10] and briefly described in the following paragraph, has the following main features: (i) it is fed by real traffic captured on a transit link then spread and played over an emulated network topology, (ii) it includes real NetFlow-like tool for traffic monitoring on all router interfaces of the emulated topology, and (iii) it implements the machine learning engine as it should be operating in real-life setting.

In addition, to validate the efficiency of the algorithms, we are particularly concerned by their feasibility and their practical deployment. During the performance evaluation study, we focus on the evaluation of the accuracy of traffic estimation for the accounting application and on the convergence of the monitoring configuration under an overhead constraint.

4.1.1 Emulation platform

Our validation platform, described in detail in [Krifa10], is composed of three services: (i) the traffic emulation service, (ii) the traffic monitoring and sampling service, and (iii) the data collection and analysis service. It is meant at emulating traffic sampling and monitoring functionalities in a backbone network. Routers can be virtual nodes connected by virtual links, but they can also be real routers connected by real links over which the monitoring platform runs. The first service of the architecture is responsible of generating the emulated traffic across the network routers. The second service implements on each router interface packet sampling and flow monitoring. This latter functionality is provided by SoftFlowd [Softflowd], open source free software capable of NetFlow measurements in high speed networks. The third service implements the MLE mainly consists of the Flowd tool in the SoftFlowd package. It is a centralized service that collects NetFlow records, correlates them to better estimate network traffic, and then runs our adaptive algorithm to decide on which sampling rates to update. SoftFlowd runs on each router interface and requires network traffic in the TcpDump format. Unfortunately, obtaining real traffic data from an entire backbone network is a hard issue, if not impossible. This has the other problem of being limited to one specific topology.

To go around this problem, we proceed in the following way. We first seek for unsampled packet level traces collected on high speed transit links. Many of such traces exist; we consider for this study the ones coming from the Japanese WIDE project [Mawi]. Each trace is assumed to be one instance of the entire emulated network traffic. We parse the trace for the IP prefixes, and we dispatch them over the Autonomous Systems (ASes) connected to the edge routers of the emulated topology. The dispatching is done randomly according to some predefined weights that determine the importance of each stub AS.

Furthermore, the dispatching preserves the IP prefixes in the traffic: two IP addresses belonging to the same prefix are assigned to the same AS. We leave it to the user to define the length of the prefix as a function of the granularity of the dispatching he wants to achieve.

This can range from all in one AS (/0) to one IP address per AS (/32) passing by prefixes of length /16 and /24. For this work, we consider the /16 prefix as the basic unit for IP address assignment to ASes since we believe it is a good representation of how the IP address space is allocated in the Internet. Any other allocation, mainly finer grained, should lead to a traffic distribution between ASes close to what we obtain with the /16 prefix, which should result in close performances and should provide the same insights on the cognitive control we present in this paper.

Once addresses are allocated, the packets in the TcpDump trace are split accordingly between the different ASes connected to the emulated topology. Shortest routes are calculated for this latter topology. Then packets in the main TcpDump trace are associated to the different monitors over their respective paths across the network with the correct timestamps derived from the main trace. In this way, packets per monitor can be grouped into a new sub TcpDump trace and fed into SoftFlowd, can now sample the packets in that monitor, form the flows and send them back to the central collector.

This sampling and monitoring is done in parallel on all network router interfaces. The different parameters of the architecture are set via an XML configuration file. The most important parameters are the topology of the studied network, the prefix length for IP address assignment and the weights of the ASes for the dispatching of packets. Note that different main TcpDump traces can be considered for the experimentation. In general, the larger the trace, the better the repartition of the emulated traffic over the network and the more meaningful the results are.

4.1.2 Validation scenarios

Our platform requires the definition of a network topology over which it dispatches and replays real traffic. This topology is supposed to connect an AS at each of its POP (Point-Of-Presence) routers. We chose to experiment over network topologies similar to the one of well known tier-1 transit networks. Two topologies, described in the following Figures (Figure 4 and Figure 5) were chosen for their widely use, the Geant topology (TOPG) [Geant] and the Abilene one (TOPA) [Abilene]. The weights of ASes needed for traffic dispatching set according to the sizes of stub ASes in Geant and Abilene and we make sure these weights sum to 1. An AS of weight ***w*** will then see itself attributed **100*w%** of the prefixes available in the trace and will see its traffic (ingoing or outgoing) being around **100*w%** of the total trace traffic, both at the flow and packet levels (random prefix allocation).

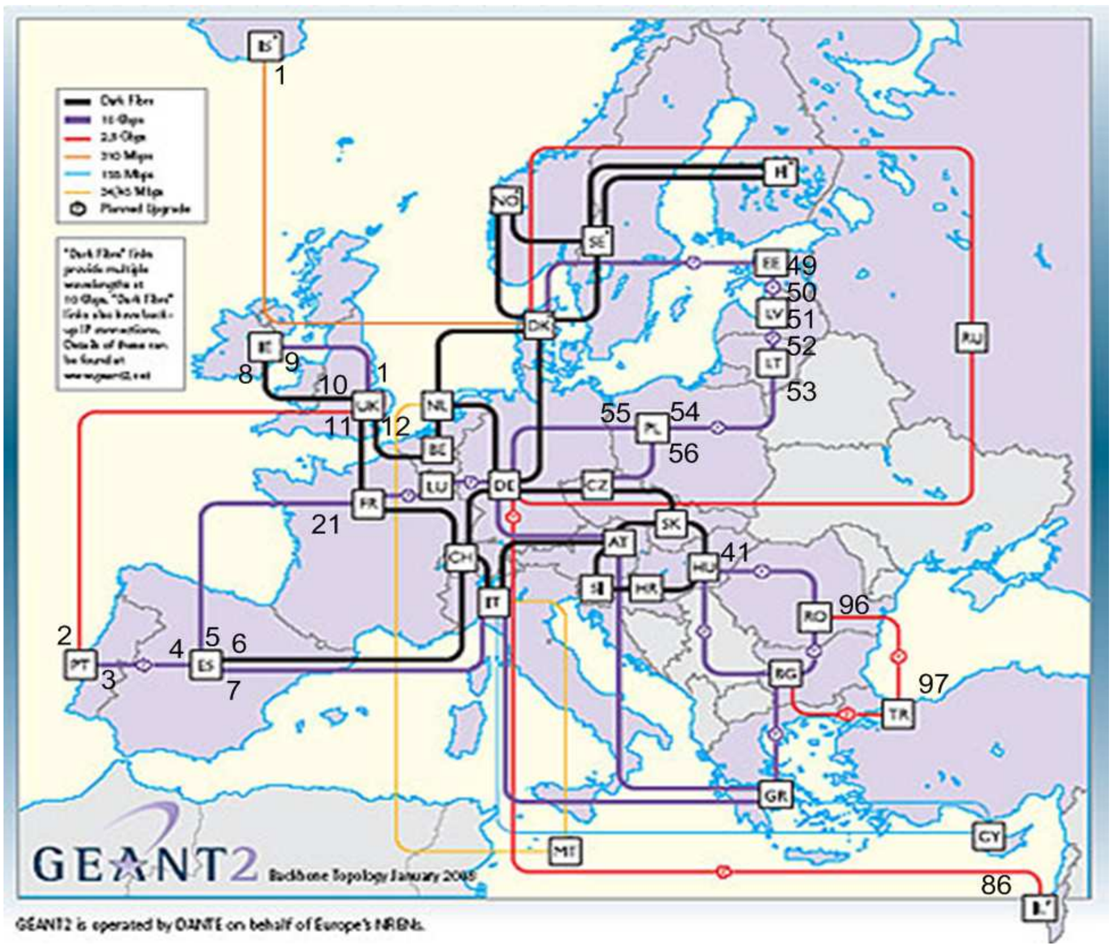

GEANT2
Backbone Topology January 2008
GEANT2 is operated by DANTE on behalf of Europe's NRENs.
IS
NO
SE
FI
EE
LV
LT
RU
DK
IE
UK
NL
BE
LU
DE
PL
CZ
FR
CH
IT
AT
SK
SI
HR
HU
RO
BG
TR
GR
PT
ES
MT
CY
IL

Figure 4: Geant Topology

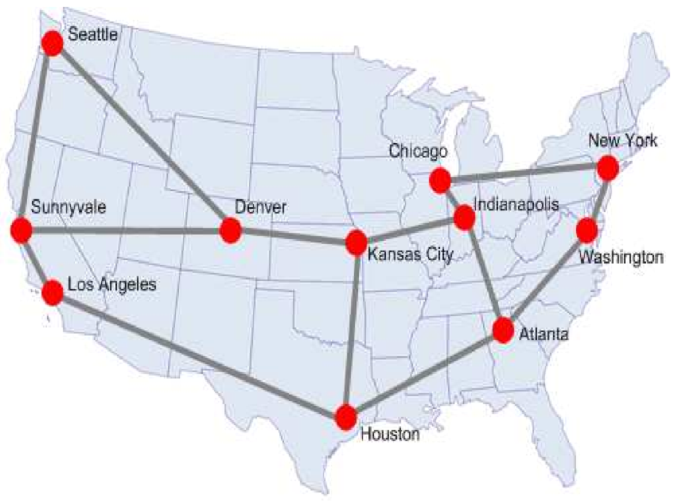


Figure 5: Abilene Topology

Once topology and weights are set, we replay over each emulated topology different traces collected at a transpacific link by the Japanese MAWI working group [Mawi]. Traffic traces are made by TcpDump, and then, IP ad-dresses in the traces are scrambled by a modified version of Tcpdpriv [Tcpdpriv]. The default scrambling configuration preserves network prefixes and IP address classes.

Later, we present results for two traces among the many ones in this data archive: Trace **S** collected on 03/03/2006 during the night making the traffic relatively smooth, and Trace **V** collected on 03/03/2006 during the day featuring more important traffic variability.

Each of the three services of the platform runs on a separate computer. Computers are fast enough to follow in real time the stream of packets in the replayed TcpDump traces. There is one computer for dispatching and replaying traffic, a second computer for topology emulation and flow monitoring, and a third computer for measurement collection. The third computer emulates the central unit; it collects NetFlow reports and implements the sampling rate adaptation algorithm.

As target application, we consider the estimation of flow sizes. We recall that a flow $\mathbf{F_i}$ is the set of 5-tuple flows that share the same AS source and AS destination. All AS-to-AS flows are jointly considered, which is often called in the literature the traffic matrix. The objective is to minimize the sum of normalized mean square errors for these flows. Initially, we set the sampling rates of the different monitors at $\mathbf{P_{init}}$ and we set the maximum time interval between two measurements at **t**. Records are received by the collector and stored in a buffer of size **C** chosen in such a way that **C = TO*t** records. This means at the equilibrium when the **TO** is reached, the collector will receive **C** flow records every time **t**. At every update of sampling rates, the buffer **C** is emptied and records are stored on a hard disk for later analysis. Practically, a new measurement and sampling rate update is triggered in the following two cases:

- Timer **t** expires and the buffer **C** is still not full. In this case, the reporting overhead is less than the **TO**. The MLE increases different sampling rates and the timer t is rescheduled.
- Buffer **C** overflows before timer t expire. In this case the reporting overhead **O** is larger than the **TO**. The **CE** decreases the sampling rates of the least significant monitors without waiting for the timer t to expire. A least significant monitor is a one causing a marginal loss in accuracy less than the average loss over all monitors. The timer t is then rescheduled.

To reconfigure the monitors and get a fast scan of the interval **[0, 1]**, we use increments in the logarithmic scale. For reconfiguring the sampling rate of e.g. monitor **k**, we set **log(p$_k$) to log(p$_k$) ± c**. This gives in the normal scale $p_k = p_k(\gamma)^{+-1}$ where γ = exp(c). In our experiments, we measure the reporting overhead O and we set the value of γ to **min{1+**$\sigma * \left|\frac{TO-O}{TO}\right|$**,3}**, so that this value varies between **1** and **3**. The reporting overhead is the number of flow records received since the last update, divided by the time since this last update. It is immediately noticed that the value of γ depends on the reporting overhead. σ is a constant parameter of the control that represents a balance between convergence speed and stability. Thus, using a small value of σ gives small multiplicative factors γ allowing the system converging smoothly while large value of σ gives large values of γ so that the system will converge rapidly.

4.2 Use case a3: Cooperative traffic anomalies and attacks detection

4.2.1 Cooperative distributed anomaly detection

Our validation architecture consists of a diffusion overlay over a real network. For our purpose we will be using the Abilene Network architecture (shown in Figure 4) as the main architecture and emulate an information exchange between 11 nodes connected by an overlay with 41 links. The diameter of this topology is 5 and the topology is well interconnected with at least two different paths between each two nodes. We set arbitrarily the weight of initial node states added to the preference list at time *k*=0 equal to 10. Weight of node states asked by neighbors are set equal to 2, i.e. we give 5 time more weight to our own wish list compared to neighbors wishes. We assume that all nodes have the same transmission rate constraint on each one of the links attached to them.

4.2.2 Joint anomaly detection and classification

We shall evaluate the ability of the Unsupervised Anomaly Detection algorithm to detect and to automatically generate a signature for a distributed network attack in real traffic data from the Japanese MAWI traffic repository [Wide00]. This real-traffic dataset comes from the WIDE Internet operational network, a test-bed network developed under the WIDE project (http://www.wide.ad.jp/). The WIDE network provides interconnection between different research institutions in Japan, as well as connection to different commercial IPSs and universities in the U.S.

The MAWI traffic repository consists of 15-minutes-long raw packet traces collected daily at 14:00 (Japan time) since 1999. These traces are provided publicly after being anonymized and stripped of payload data.

The trace we shall work with consists in traffic captured in January 2004 at one of the trans-pacific links between Japan and the U.S., measured at sample point-B. The line is a 100-Mbps link with 18-Mbps CAR (Committed Access Rate).

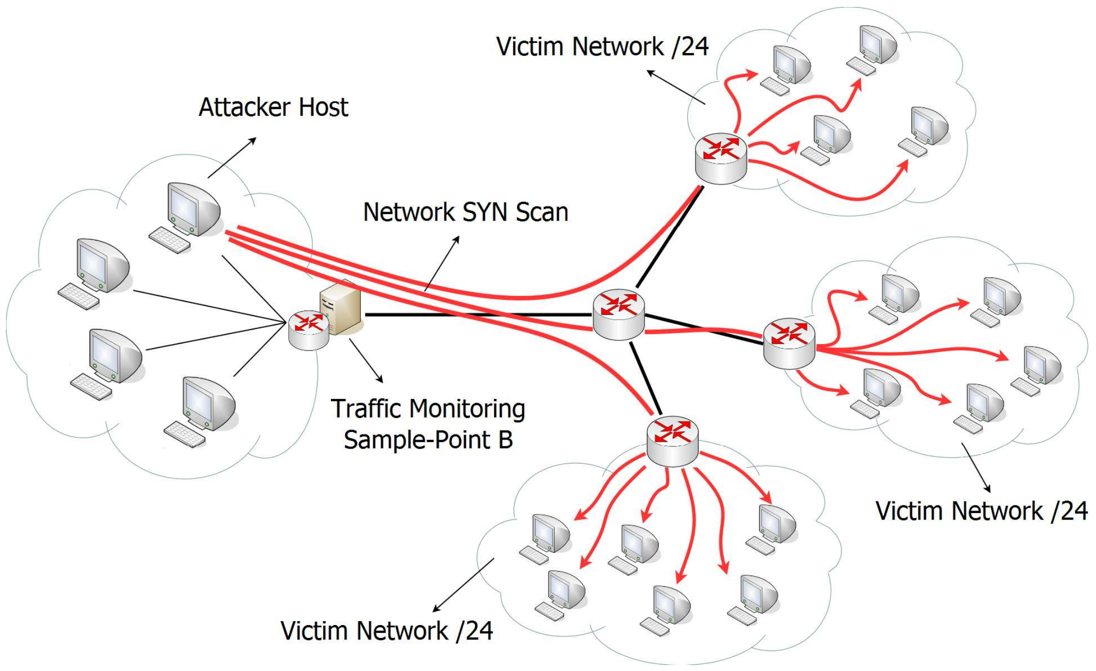


Figure 6: Distributed Network Attack in WIDE - Network SYN Scan.

In the first scenario, we analyze a distributed network SYN scan attack directed to many victim hosts under IP network address 162.225.0.0/16, originated at an attacker host with origin IP address 204.243.26.29. The attack starts at time 14:06:43 and it is directed towards multiple hosts and destination ports. Figure 6 depicts this situation.

Traffic is aggregated on a destination /24 network basis, using packet destination IP address and bitmask 255.255.255.0. The sliding-window time-scale granularity used by NewNADA to compute absolute-deltoids is 20 seconds. Absolute-deltoids are computed for three different volume metrics, namely number of packets (# pkts), number of bytes (# bytes), and number of SYN packets (# SYN) [Fernandes 2009].

The attack that we analyze consists in a distributed network SYN scan directed to many victim hosts under IP network address 162.225.0.0/16, originated at an attacker host with origin IP address 204.243.26.29. The attack starts at time 14:06:43 and it is directed towards multiple hosts and destination ports. Figure 6 depicts this situation.

The performance analysis of the unsupervised anomaly detection method is done in two different frameworks: prototype evaluation in Matlab, and implementation evaluation in iLAB. In the case of prototype evaluation, we shall analyze the MAWI real traffic trace in an off-line fashion, but using a temporal sliding-window to analyze traffic in consecutive temporal bins. In the case of implementation evaluation, we shall replay the real traffic from the MAWI trace and perform the analysis in an on-line basis.

The implementation setup in iLAB is composed of several sources (at least two), a monitoring host and a gateway that will announce itself as the next hop for all the outgoing traffic of the monitoring host. Figure 7 depicts the proposed setup.

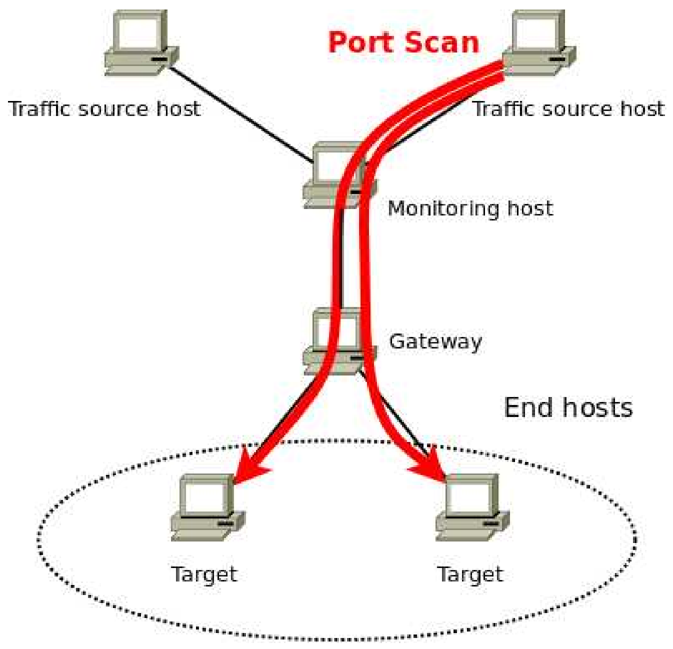


Figure 7: iLAB setup for implementation evaluation in real traffic from the WIDE project. A distributed network SYN scan is re-played.

We shall reproduce the network SYN scan by replaying the traffic traces from MAWI. We replay a trace of normal traffic in every source host, so as to simulate background traffic. In order to simulate the attack, here a network scan, we replay a trace composed of only traffic packets from a real network scan, extracted from the MAWI trace. This trace will be replayed from only one of the traffic sources. We extract the real anomaly from the MAWI trace in two steps: firstly, we filter the original trace containing the anomaly, keeping only the anomalous packets; secondly, we re-forge the trace in terms of IP addresses and time-stamps so as to fit to our scenario.

The scenario will then simply consist of replaying a normal trace to create background traffic through the monitoring host and the gateway. We then launch the attack by replaying the anomalous trace. Our system running on the monitoring host will then detect the unknown anomaly and characterize it, automatically producing a signature for the a-priori unknown anomaly.

We expect that the detection results provided by the iLAB experimentation will be exactly the same as those provided by the prototype evaluation in Matlab. The important issue will be then to assess the performance of the iLAB implementation as regards execution time, aiming at an on-line implementation of the algorithms.

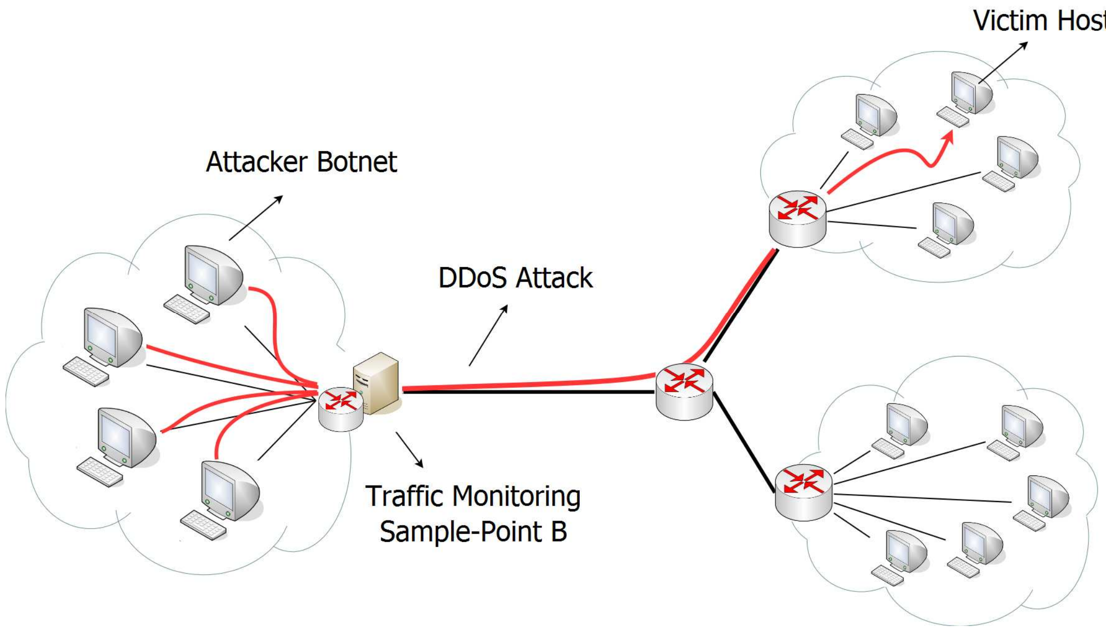


Figure 8: Distributed Network Attack in WIDE – TCP SYN DDoS.

In the second attack scenario, we shall analyze a distributed denial of service attack directed to a single victim host with destination IP address 95.5.63.90, originated at an attacker botnet under IP address 165.35.115.0/24. Figure 8 depicts this situation.

In this case, traffic is aggregated on an origin host basis, using packet origin IP address and bitmask 255.255.255.255. The sliding-window time-scale granularity used by NewNADA to compute absolute-deltoids is the same as before. The evaluation for this traffic scenario is only presented for the prototype implementation under Matlab, as results in the iLAB experimentation are exactly the same.

5. Experimental results

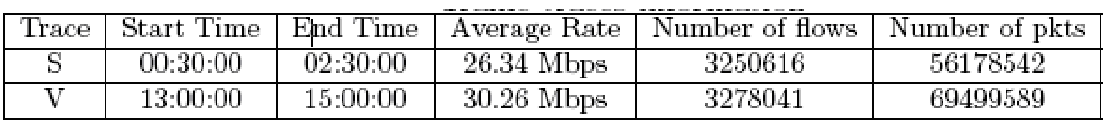
5.1 Use case a1: Adaptive traffic sampling

We divide the validation results into three parts. First, we study the efficiency and convergence of our adaptive solution and its ability to adapt to the heterogeneity of flow rates and to the predefined collected traffic overhead. Second, we show the practical benefits of deploying our optimization approach by comparing it to the common static configuration approach where sampling is only performed at the edge of the network. For fairness, the comparison is done at equal overhead. Last but not least, we present a global sensitivity analysis of the importance of the different parameters of our algorithm and we calculate their influence on the system behavior. This analysis will confirm our focus on the overhead threshold in the first two parts.

5.1.1 System efficiency, adaptability and convergence

The proposed monitoring system should satisfy the following properties:

- **Convergence:** Starting from any initial configuration value $\mathbf{P_{init}}$ of all sampling rates (usually a low value), we want to know if our system is able to converge to an equilibrium in its configuration and hence in the realized monitoring accuracy. To detect this equilibrium, we will experiment the same scenario for different initial sampling rates and check the final state of the system. The system converges when the mean relative error stabilizes and stops improving.
- **Reactivity:** Any change in the network traffic should bring the system temporary out of its convergence state before it converges again toward a new equilibrium. We want to observe if our system can detect changes in the network traffic and if it is able to find quickly this new equilibrium.
- **Target Overhead:** We want to test the capability of our system to respect the imposed constraint on the rate of measurement records. Among the set of configurations resulting in a rate of records equal to the imposed constraint, the system should be able to find the one minimizing the error on the target measurement task.



Trace	Start Time	End Time	Average Rate	Number of flows	Number of pkts
S	00:30:00	02:30:00	26.34 Mbps	3250616	56178542
V	13:00:00	15:00:00	30.26 Mbps	3278041	69499589

Table 1: Traffic traces information

The above points will be addressed next by real experiments over the two network topologies Geant-like and Abilene-like and by the help of the two traces **S** and **V** described in the Table 1.

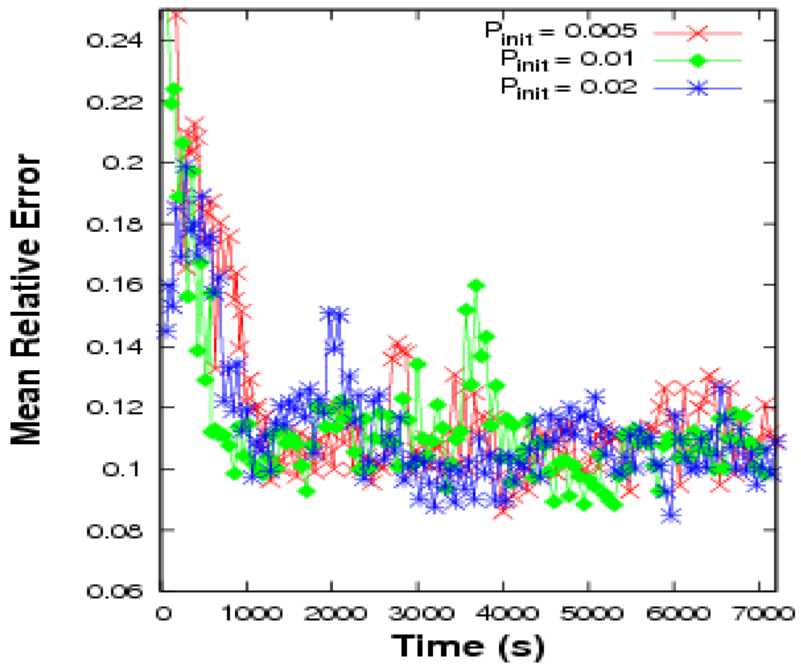


Figure 9: Mean relative error vs. time using Trace S

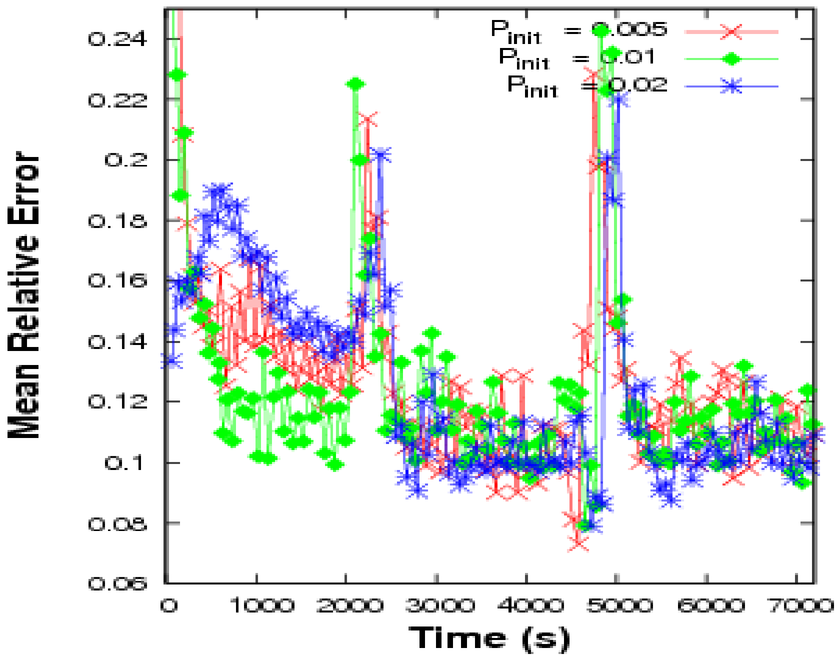


Figure 10: Mean relative error vs. time using Trace V

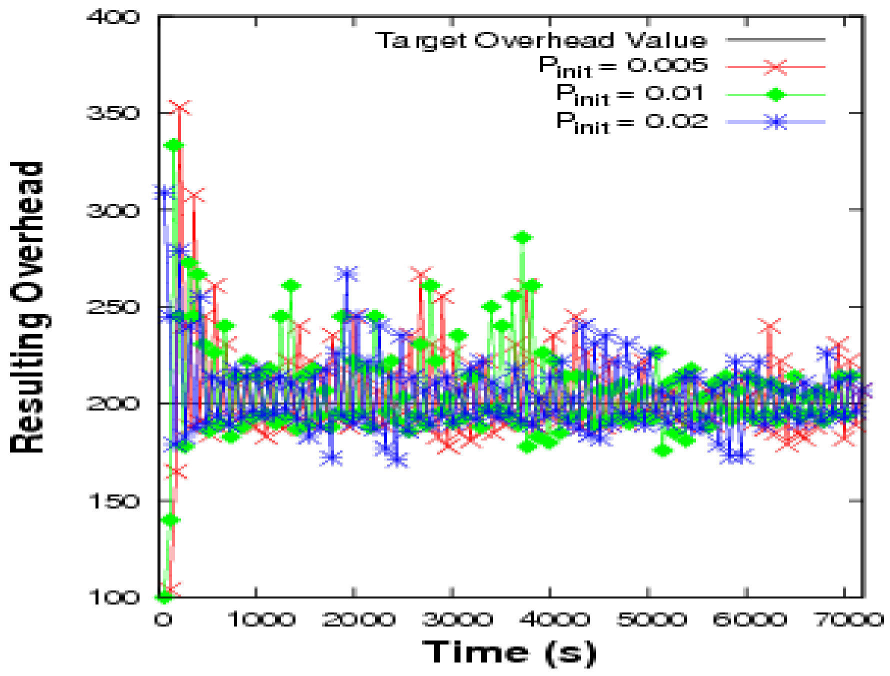


Figure 11: Resulting overhead vs. time using Trace S

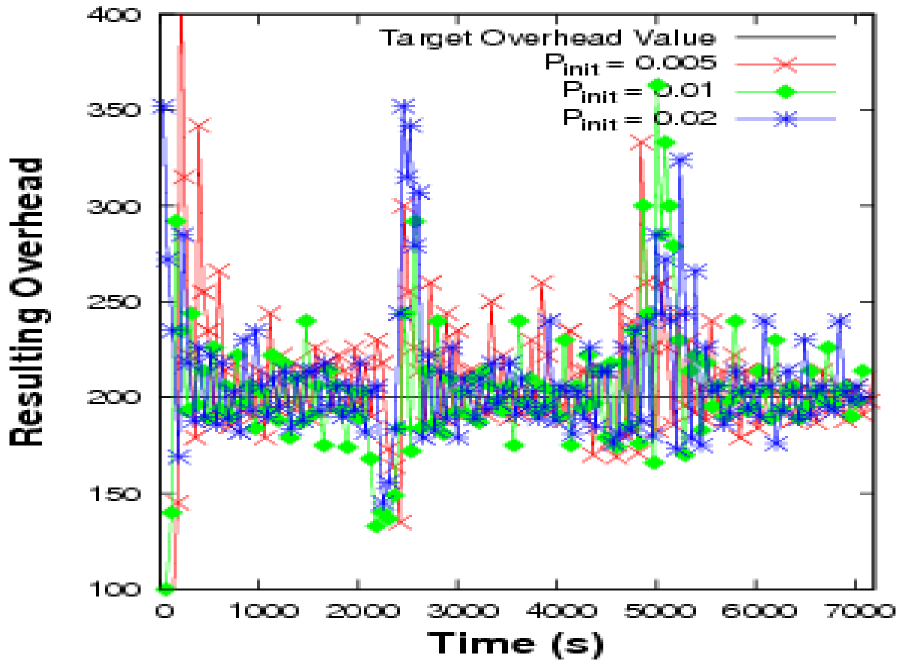


Figure 12: Resulting overhead vs. time using Trace V

In Figure 9, Figure 10, Figure 11 and Figure 12, we plot the evolution of the mean relative error obtained over all AS-to-AS flows (on the left hand side) and the resulting overhead in NetFlow-records/s (on the right hand side) over time using the two traces **S** and **V**. Each point in the graphs corresponds to an update of the sampling rates, either in the increase (overhead **O** less than the target value) or in the decrease (overhead **O** larger than the target value and the buffer **B** is full). For this experiment, we set the timer **t** for updating sampling rates to **1** minute, the regulator σ to **2**, the minimum possible sampling rate $\mathbf{SR}_{min}$ to **0.0005** and the maximum possible one $\mathbf{SR}_{max}$ to **1**. The **TO** is set to **200** NetFlow-records/s.

Three initial sampling rates are considered: **0.005**, **0.01** and **0.02**. We can immediately observe that the system keeps improving the global accuracy while fully profiting from the available resources for measurement collection. At the beginning, the system exponentially increases sampling rates until the **TO** is reached. Once done, it keeps improving the accuracy of the estimation while maintaining the overhead around its target value. After little iteration, the system reaches equilibrium where the mean relative error tends to oscillate around its minimum value. For the smooth trace **S**, the equilibrium does not change much along the trace. For the other variable trace **V** however, we can see in the middle of the trace sudden increases in the error caused by sudden changes in the traffic. The system adapts to these changes by recalculating a new optimal configuration, always at a constant overhead. Note how the behavior is almost identical for the three initial sampling rates illustrating the stability of our system and its ability to converge in small number of iterations (few minutes here) to an equilibrium that only depends on traffic conditions and monitoring target and not on the initial configuration of sampling rates.

These results are illustrated in Figure 13, presenting the evolution of some sampling rates over the time starting from an initial configuration **Pinit** equal to **0.005** and using the trace **V**. We can observe the ability of our system to converge to equilibrium in its configuration. Once done it keeps oscillating around this optimal configuration until the network conditions change. Moreover, we notice the capability of our system to track any change in the network conditions as well as to adapt sampling rates to move smoothly towards a new optimal configuration. To show the spatial distribution of the sampling rates of the different monitors labeled in Figure 4, we plot in **Figure 14** their sampling rates at time instant 2023s.

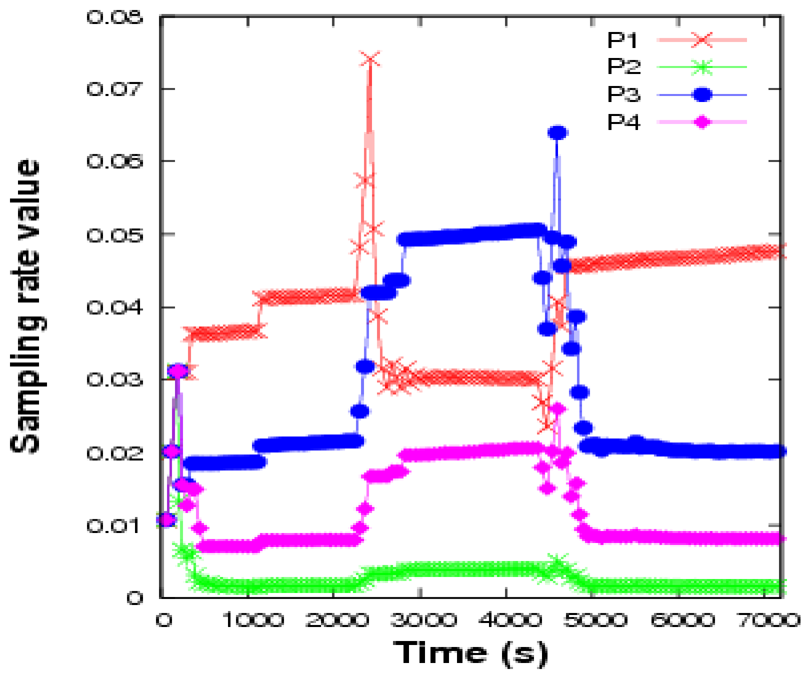


Figure 13: Evolution of some sampling rates vs. time using trace V and TOPG.

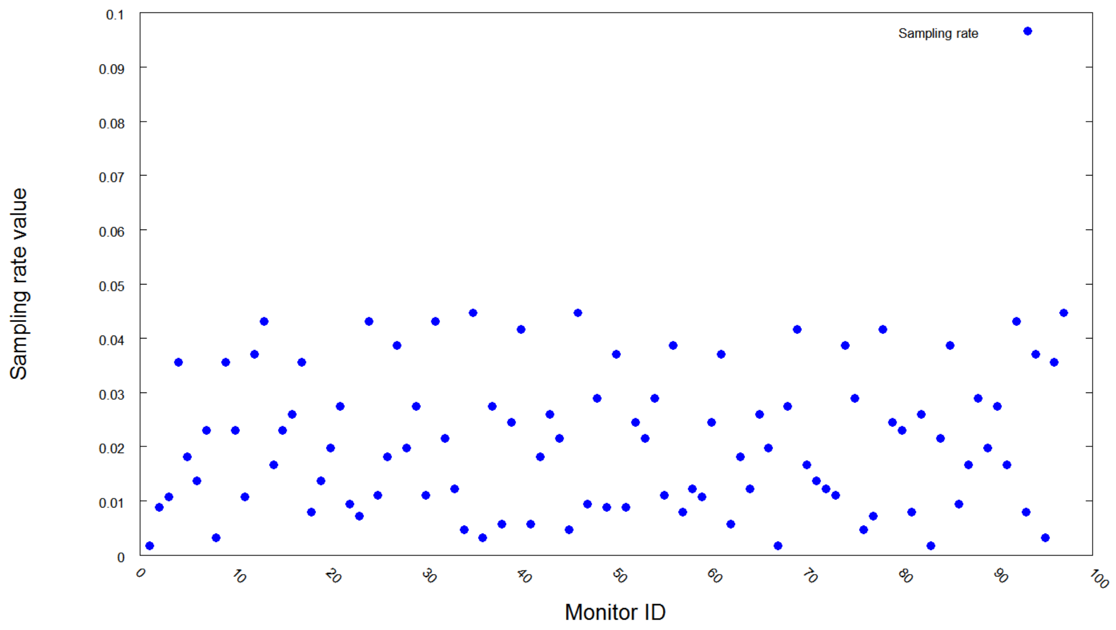


Figure 14: Sampling rates of the different monitors at time instant 2023s.

Figure 15 shows the value of the mean relative error for different target overhead values. These results are for topology **TOPG** and Trace **V**. One can immediately notice the impact of the **TO** on the traffic estimation accuracy. There is a clear reduction of the overall measurement error from **0.402** for a **TO** equal to **100** NetFlow-records/s, to **0.08** for a **TO** equal to **300** NetFlow-records/s. Indeed, for each TO value, the system tries to find the best configuration that minimizes the traffic estimation error. When **TO** is low, the system has to lower the sampling rates in the least significant monitors with the objective to reduce the rate of collected measurement records without much compromising the estimation accuracy. Allowing more overhead gives the system more freedom in increasing the sampling rates of the most significant monitors looking for better estimation of the sizes of the target flows.

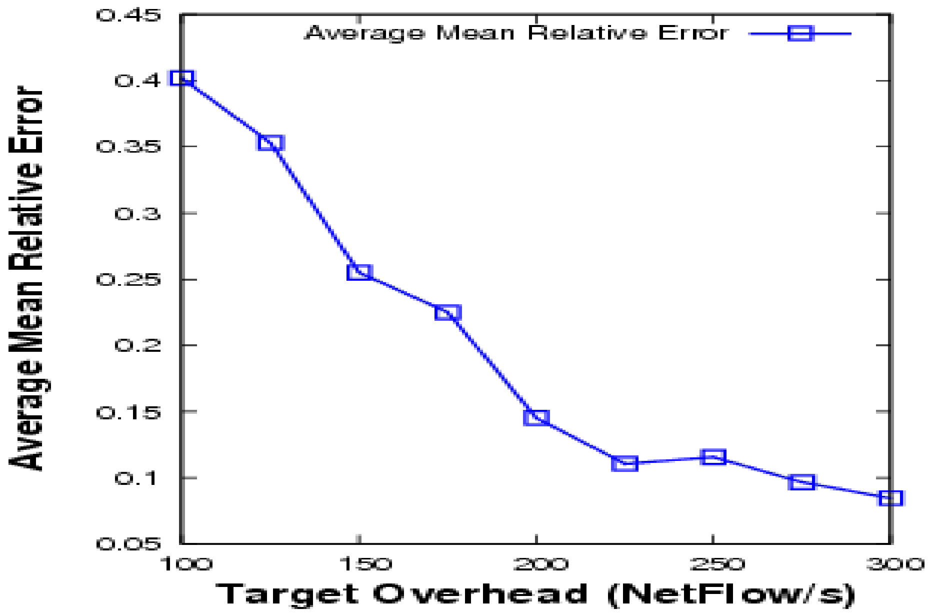


Figure 15: Average mean relative error for different TO values.

The main strength of our system is that it is able to cope with any **TO** value and provides for this value the best configuration of monitors. Now, this configuration might not satisfy the administrator in terms of the accuracy of the measurement, in this case the only remaining solution is to increase the value of **TO**. In a future research we will be working on an enhanced version of our system that adapts the **TO** in such a way to realize the measurement task with some predefined minimum accuracy. For now, we suppose the **TO** is a constraint set by the administrator and we let our system find the best configuration that maximizes accuracy.

To illustrate the capacity of our system to maintain the measurement overhead around the **TO**, we plot in Figure 16 the measured overhead in terms of collected NetFlow-records/s as a function of experimentation time and this is for three **TO** values. We can clearly see how for each experiment, whose traffic estimation accuracy is reported in Figure 15, the real overhead is maintained around the target value and how our system is able to converge and adapt to variations in traffic conditions along the trace lifetime.

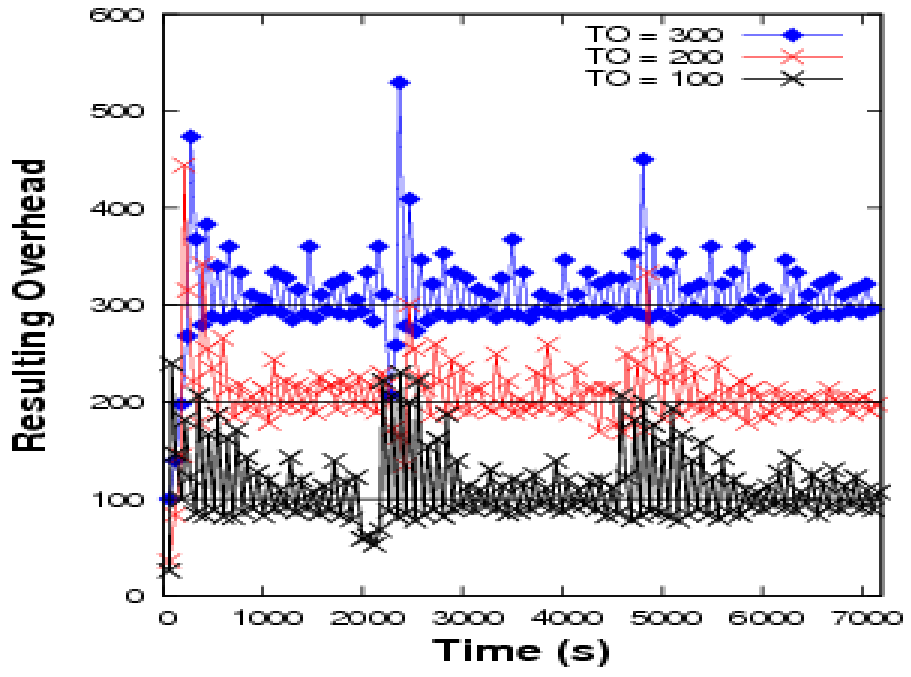


Figure 16: Resulting overhead vs. time using three different TO values.

The experiments over the Abilene-like topology confirm the same findings about the performance of our system. To give a sample of the obtained results, we plot in Figure 17 the mean relative estimation error averaged over all flows as a function of the **TO**. This figure is the equivalent of Figure 15 for the Geant-like topology. We can notice how the two figures look the same. The error for the Abilene-like topology is slightly smaller which comes from the smaller size of this topology and hence the larger volume of flows. Note that both experiments are conducted at equal total traffic driven by the same Trace **V**.

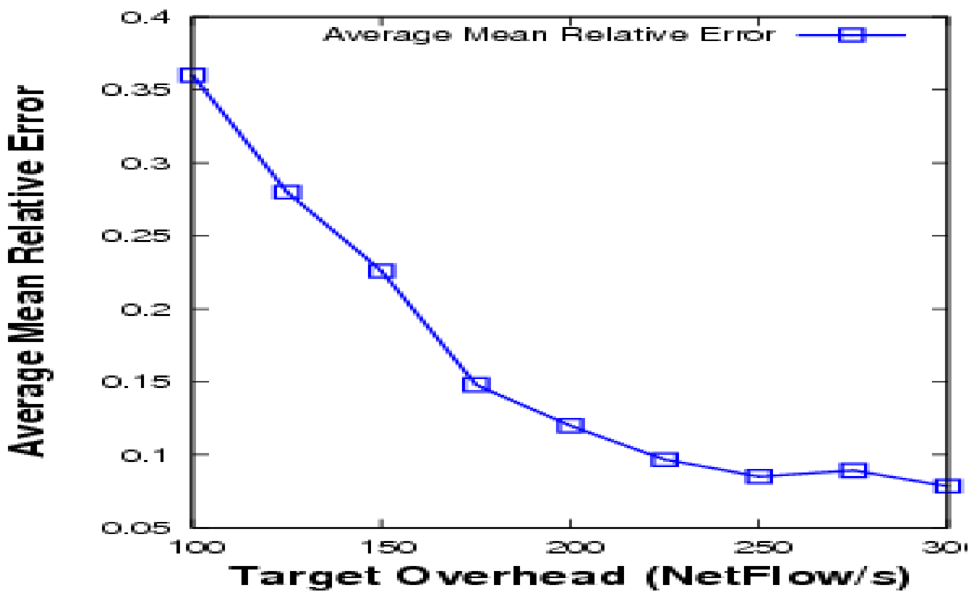


Figure 17: Average mean relative error vs. target overhead for the Abilene-like topology.

5.1.2 Fairness and comparison with the local static method

In the last part, we argued that an adaptive system that coordinates sampling responsibilities between the different monitors can considerably improve the flow monitoring capabilities of the network. In this paragraph, we are interested in comparing our adaptive solution with the standard static configuration of NetFlow in order to assess the ability of our system to avoid unnecessary measurements while tracking efficiently the target flows at constant overhead.

By minimizing the sum of flow relative errors, our system gives the same weight to each flow independently of its volume. This should naturally lead to a fair allocation of sampling rates that homogenizes estimation errors over target flows. This ability to track fairly (same for all flows independently of their sizes) small and large flows constitutes one of the main strengths of our system. Any static configuration of sampling rates does not provide this fairness feature. Note that we are talking here about aggregate flows **Fi**. Each aggregate flow is composed of a set of 5-tuple flows whose total volume is estimated. In order to illustrate the fairness of our approach with respect to target flows, we plot in Figure 18 the evolution of the mean relative error of all the flows over time.

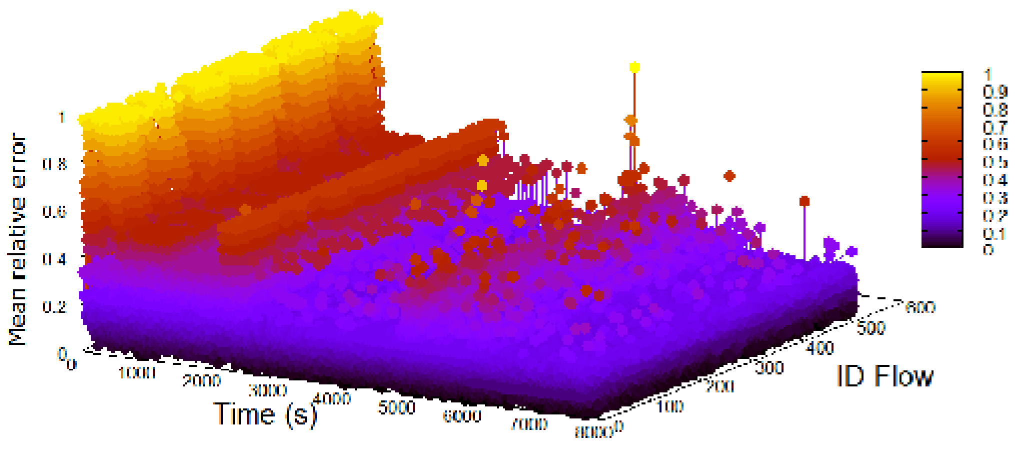


Figure 18: The evolution of the mean relative error of all the flows vs. time.

Starting from a large mean relative error, we can clearly note how our system keeps reducing this estimation error for all flows at almost the same rate even though these flows span different volumes. Some of the flows unfortunately still suffer from a large relative error because of their very small volume.

Then, we move to the comparison of the performance of our system with the widely deployed NetFlow solution (the local static solution), which consists of monitoring traffic at the edge of the network with static sampling rates. For this latter solution, each flow is monitored only one time at the input interface of the edge router of its originating AS.

This solution has the advantage that every sampled packet belongs to one of the flows of interest thereby flows can be easily formed at the collector. The problem is that this offers few options to sample a flow, and thus small flows that get mixed at their input interface with large flows suffer from a low sampling rate. Our approach has the nice feature of giving more choices for where to sample a flow, hence the protection of small flows. One has to add the dynamic feature of our approach and its ability to combine multiple measurements for the same flow and to limit the overhead. For comparison purposes, we use two specific accounting applications: 1) **Traffic matrix estimation**: all AS-to-AS flows are considered, and 2) **AS traffic estimation:** the total volume of traffic generated by each stub AS.

For this experiment, we use the Geant-like topology and the variable traffic trace **V**. The parameters of the experimentation are set as in the previous sections. For the sampling rate in the case of the local static configuration, denoted by **p**, we set it in such a way that the resulting reporting overhead is the same as in our network-wide adaptive case, and this for the main purpose of fairness between the two approaches. If $\mathbf{N_S}$ is the total number of 5-tuple flows in the trace, **D** the duration of the trace, $\pi(S)$ the probability to sample a 5-tuple flow of size **S** packets, **S** being a random variable, then the sampling rate p is given by: $\frac{N_S * \pi(S)}{D} = \frac{N_S * E\left[1-(1-p)^S\right]}{D} = TO$ The term on the right-hand side is no other than the target overhead of our adaptive architecture. The term on the left-hand side is an estimation of the rate of collected records in the local static configuration.

1) **Traffic matrix estimation:** While giving on average close performance to our approach, the local static solution presents sampling bias against small flows as we can see in Figure 19 for the case of the smallest 20 flows.

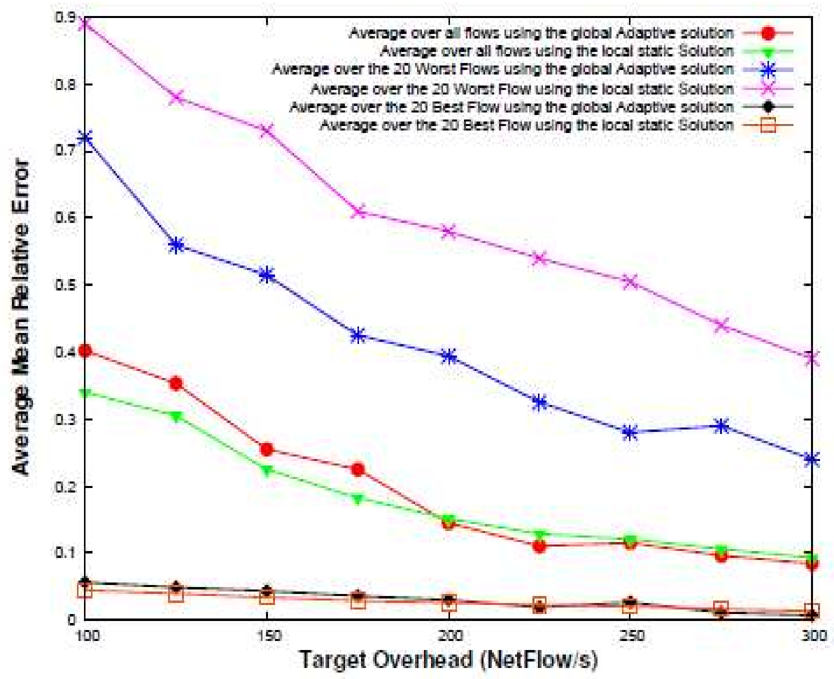


Figure 19: The mean relative error of flow measurements: Our global adaptive approach vs. local static one.

This figure plots the average mean relative error as a function of the **TO.** With the local static solution, small flows dilute within large flows and suffer from low estimation accuracy. If this happens, no other choices are available to sample them elsewhere. However, with our approach, we are able to track small flows on other lightly loaded links inside the network and combine measurements from different routers together without incurring much overhead on the system. As we can see, in order to track small flows using the local static solution with a similar accuracy to the one we obtain using the adaptive solution, we have to use a **TO** value larger than **150%** of the value used by the adaptive solution.

2) **AS traffic estimation:** we change our objective and instead of defining a flow as being the volume of traffic from one stub AS to another stub AS, we define it as the total volume of traffic generated by each stub AS. We count both the outgoing and ingoing traffic for each AS. The best configuration is the one that minimizes the sum of mean square relative errors of AS traffic estimators. Changing target measurement is very easy in the context of our approach; one has to correctly define an aggregate flow, the configuration of sampling rates follows automatically. In this scenario, the traffic volume of an AS should be proportional to the weight attributed to it during the trace dispatching phase. To further prove the generality and efficiency of our approach in compared to the standard local static one, we perform two tasks within this scenario: In a first time, we estimate the traffic volume of the different ASes (All ASes task). Then, we estimate the traffic volume of ASes contributing to more than some percentage of the total traffic trace (Large ASes task). We present results for a **6%** threshold. Some ASes are smaller than this threshold but there traffic might still be reported to the central collector, yet it is not included in the optimization loop and is not returned to the monitoring application. The main purpose of our architecture is to reduce this volume of undesirable traffic. The All ASes task is a particular case of this second general task and can be obtained by setting the threshold to **0%.**

AS	Affected weight	Without sampling		Edge sloution		All ASes		Large ASes	
		AS ratio	# of pkts	AS ratio	# of pkts	AS ratio	# of pkts	AS ratio	# of pkts
2	10	9.926	13797278	8.91	14273284	10.88	12937707	9.35	14040661
5	8	6.66	9258345	5.31	8499160	7.08	8415835	6.568	9860137
4	7	8.13	11302633	6.616	10590567	8.74	10387119	7.928	11901672
7	6	7.23	10049975	5.782	9256026	7.704	9155527	7.1	10663023
8	5	6.03	8381660	4.749	7602165	6.376	7577020	5.968	8959994
11	4	3.04	4228675	1.807	2892413	2.975	3535172	0.45	676588
9	2	1.87	2599884	0.633	1013954	1.542	1832918	0.155	233989
Total number of packets		138999178		160057553		118830397		150119112	

Table 2: Comparing AS traffic volume estimations

Table 2 presents a summary of the experimental results for a selection of ASes. The first two columns present the AS number and its associated weight. The other columns present the AS traffic volume estimation in number of packets and the ratio of this volume with respect to the total estimated network traffic. Four configurations are presented, the one without sampling as a reference configuration, the local static one, the adaptive All ASes one, and the adaptive Large ASes one. A set of observations can be made from these results.

The first observation is that the All ASes adaptive configuration provides more accurate results for all AS traffic volumes independently of their sizes. The traffic volume of ASes is better estimated than with the local static configuration, especially for small ASes whose traffic get diluted within the traffic of large ASes if only sampled at the edge. The second observation we can make is that for large ASes, one can even get a better estimation by only focusing in the optimization on the sizes of these large ASes. In fact, by only focusing on large ASes we can avoid unnecessary data coming from small ASes and thus profit from the monitoring resources to improve the accuracy of large ASes.

As requested, small ASes contributing to less than **6%** of the total network traffic get ignored by our optimization, hence the decrease in their accuracy. Indeed, the overhead these small ASes generate with the All ASes configuration and the local static one is used to better sample the large ASes and to better estimate their traffic volumes. These results illustrate the adaptive nature of our approach and its capacity to cope with the monitoring application needs, always at constant monitoring overhead.

5.1.3 Global sensitivity analysis

In the previous two paragraphs, we give a particular attention to the impact of the overhead target value. Yet, the system has other parameters and it is important to evaluate their impact as well. In this paragraph, we demonstrate indeed that, apart from the overhead target value, the other parameters have minor impact on system performance.

The goal of global sensitivity analysis is to characterize, qualitatively or quantitatively, what impact an input parameter has on a system output and how it compares with the impact of the other parameters. Fourier Amplitude Sensitivity Test (FAST) [Cukier73, Cukier75] is considered to be one of the most efficient methods in sensitivity analysis [Duffield02, Duffield05]. Among its advantages are: fast implementation, possibility to deal with no monotonic models, arbitrary large variations in input parameters, and no need for the knowledge of the mathematical model.

The main idea of FAST is to assign to each parameter a distinct integer frequency (characteristic frequency). Then, for a specific parameter, the variance contribution can be singled out of the model output with the help of the Fourier transformation. Therefore, FAST is also referred to as variance based sensitivity analysis. Specifically, let us consider a nonlinear model $y = f(x_{1,} x_2, x_3, ..., x_n)$ where x_n are parameters. We emphasize that the FAST method does not require the analytic knowledge of the function f(·). Various search functions have been proposed. The search function must let the parameter x_i to oscillate with frequency w_i. For instance, the authors of [Saltelli99] have proposed the search function $x_i = \frac{1}{2} + \frac{1}{\pi}\arcsin(\sin(w_i s))$, which is a particular case of a more general search function $x_i = F_i^{-1}(\frac{1}{2} + \frac{1}{\pi}\arcsin(\sin(w_i s))$ (1), where $F_i^{-1}(.)$ is the inverse cumulative distribution function for x_i. To make more efficient use of the model evaluations, the authors of [Saltelli99] have suggested the following slight modification $x_i = \frac{1}{2} + \frac{1}{\pi}\arcsin(\sin(w_i s + \varphi_i)$ (2), where φ_i is a random phase-shift chosen uniformly in the interval [0, 2π). The model output becomes a periodic function with period 2π. Thus, we can represent the model with a Fourier series, $y = f(x_1, x_2, ..., x_n) = A_0 + \sum_{k=1}^{\infty}[A_k \cos(ks) + B_k \sin(Ks)]$

If we denote a sample of size N as S = {s_1, s_2... s_N}, then, using either (1) or (2) as a search function, we can obtain the sampled values of the parameters X_i = {xi_1, xi_2... xi_N}, and the discrete Fourier transform coefficients

$$A_0 = \frac{1}{N}\sum_{j=1}^{N} f(s_j)$$

$$A_k = \frac{2}{N}\sum_{j=1}^{N} f(s_j)\cos(s_j k) \text{ and } B_k = \frac{2}{N}\sum_{j=1}^{N} f(sj)\sin(s_j k)$$

where $f(s_j) = f(x_{1j}, x_{2j}, ..., x_{nj})$ and $k = 1, ..., (N-1)/2$.

The variance of the model output can be decomposed into variance components at the integer frequencies,

$$V = \frac{1}{2}\sum_{k=1}^{(N-1)/2}[A_k^w + B_k^2]$$

By summing the spectrum values $A_k = [A_k^2 + B_k^2]/2$ for the characteristic frequencies w_i and their higher harmonics, the partial variance in model output arising from the uncertainty x_i, V_i can be estimated by $V_i = \sum_p \Delta_p w_i$ where $pw_i \leq (N-1)/2$. The ratio $\frac{V_i}{V}$ measures the contribution of parameter x_i. This ratio is also referred to as the first-order sensitivity index.

Because the characteristic frequencies are integers, there will be an aliasing effect if one frequency is a linear combination of the others.

It is said that a frequency set is free of interferences to an order M if

$$\sum_{1}^{n} a_i w_i \neq 0, \sum_{i=1}^{n} |a_i| \leq M + 1$$

where a_i an integer, and M is a design integer (usually 4 or 6). In order to avoid the interference effect, the maximal value of p in calculating V_i should be M. In [Cukier73], the authors have proposed the following empirical formula for calculating the characteristic frequencies free of interference up to order M = 4: $w_1 = \Omega_n$, and $w_i = w_{i-1} + d_{n+1-i}, i = 2...n.$. The parameters Ω_n and d_k can be found in a table provided in [Cukier73]. Below we give several lines from that table.

Dimension, n	Ω_n	d_n	Minimal number of points, N_{min}
1		4	
2		8	
3	1	6	38
4	5	10	78
5	11	20	142
6	1	22	182

Table 3: Frequencies

Then, for instance, for the case of six input parameters we obtain the following values of the characteristic frequencies:

$$w_1 = \Omega_6 = 1; w_2 = w_1 + d_5 = 21; w_3 = w_2 + d_4 = 31; w_4 = w_3 + d_3 = 37; w_5 = w_4 + d_2 = 45;$$
$$w_6 = w_5 + d_1 = 49$$

We have applied the method FAST to our system in order to characterize the impact of the different parameters used in experimentations on results. Table 3 summarizes the set of evaluated parameters with their ranges. The last column presents the impact of each parameter on the system output. It is immediately noticed that the parameter having the most important impact on the system output is the target overhead TO, while the other parameters have a relatively low impact on results, in the order of 1% or less. Indeed, for some value of TO, there is an optimal configuration of monitors and our system will converge to this optimal configuration in a robust manner with respect to the other parameters. It is only by changing the value of TO that the system will converge to another optimal configuration yielding another measurement precision.

Parameter	symbol	range	impact
Target Overhead	TO	$[20, 500]$	0.58
Increasing sampling rates timer	t	$[60s, 300s]$	0.0142
γ regulator	σ	$[1, 10]$	0.00747
Initial sampling rate value	P_{init}	$[0.005, 0.02]$	0.01179
Minimum sampling rate value	SR_{min}	$[0.0005, 0.005]$	0.00691
Maximum sampling rate value	SR_{max}	$[0.02, 1]$	0.00721

Table 4: Parameters of the experiment

5.2 Use case a3: Cooperative traffic anomalies and attacks detection

5.2.1 Cooperative distributed anomaly detection

In this part, we will present the results of the experimental setting in a realistic network scenario. The data used for validation are coming from IP-level traffic flow measurements collected over the Abilene 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 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 28x41x32=36768 bits per second for transmitting the data without compression.

In the first setting, we assume that all these routers have to send their state vectors to a central aggregation point. 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}. By using the diffusion scheme described here, we can achieve performances shown in Figure 20.

We show here three settings: in the first setting we just implement a local compression without taking into account the spatial correlations; in the second setting named spatial compression we implement the fully distributed scheme, but assuming that the temporal correlation horizon is 1 (that is not really the case); in the third setting we are fully with a temporal correlation horizon *T*=5.

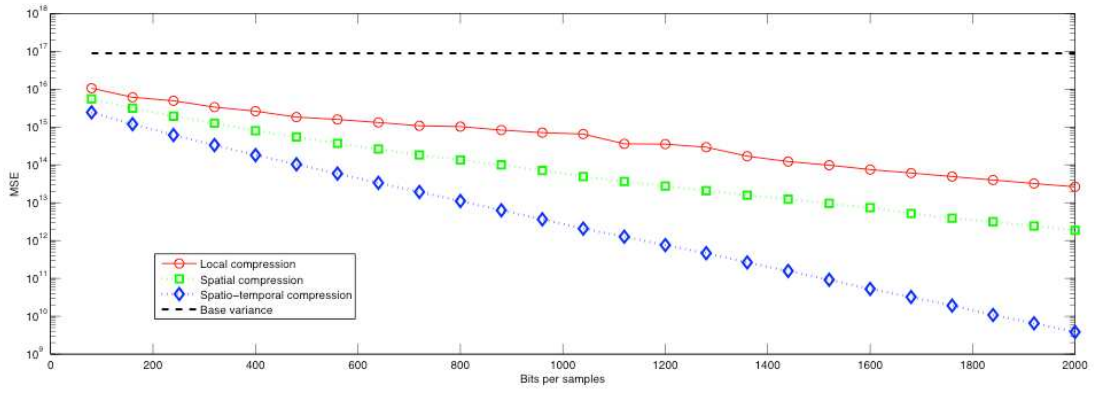


Figure 20: Overall MSE achieved in the single hop scenario for the temporal, the spatial and the spatio-temporal setting

One can observe that with just 2000 bits per samples (meaning a compression rate higher than 18) we can achieve an MSE that is 7 order of magnitude less than the initial overall MSE, moreover applying spatio-temporal compression results in an MSE that is 4 order of magnitude lower that just doing a local compression. In particular, the spatio-temporal compression achieves with 400 bits the performance that the local compression alone achieves at 2000 bits.

It is worthy to discuss about the overhead involved in this scenarios. For this purpose, we show in Figure 21 the MSE achieved at one node as a function of the iteration number. Each iteration took 30 seconds as we estimate the covariance matrix applying the method described above 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 that projection coefficient is encoded into 16 bits. However, projection matrix for the spatio-temporal case is larger as the dimension of the initial space is 5 times larger. The number of forwarded projection varies as the optimal number of projections changes.

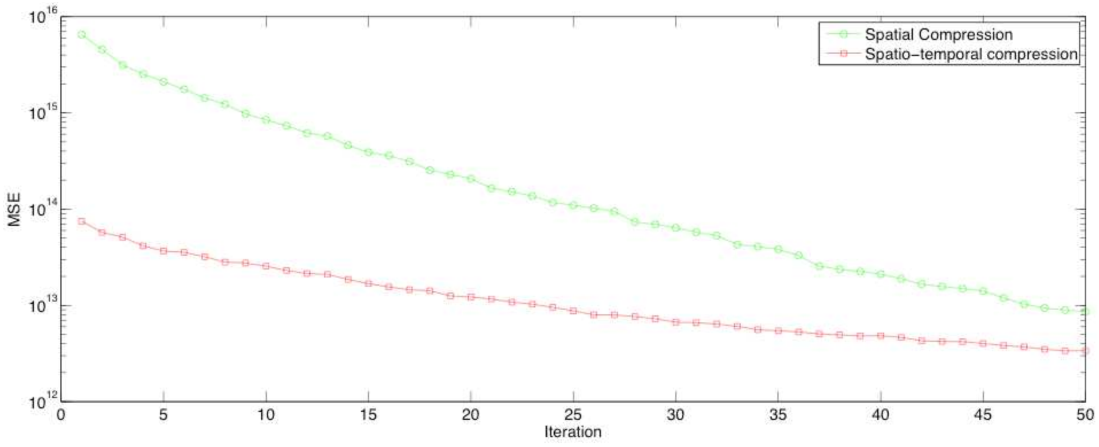


Figure 21: MSE achieved after a number of iteration of the distributed compression scheme for a global rate of 1200 bits per sample}

Figure 21 shows that the number of needed iteration to stabilize the estimation is also different between the spatial and spatio-temporal scheme; the spatio-temporal scheme need about 20 iterations where the spatial scheme need about 40 iterations. Accounting all these source of variations, we have observed around for the spatial scheme and overhead around 119 Kbits and for the spatial-temporal scheme an overhead around 962 Kbits. Accounting the difference in needed rate for achieving a giving 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 scheme is covered in 527 sec (about 9 mins of operation), and the difference between the uncompressed scheme and the spatio-temporal scheme is covered in 25 secs, 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.

Now we validate our proposed cooperative scheme by evaluating two behaviors: the first behavior is asocial, *i.e.* the node does not try to help his neighbors in improving the estimation of the preference list of its neighbors; the second behavior is social, *i.e.* the node plays the game and help his neighbors in improving their estimation. To evaluate we run two times the same simulation; the first time with all nodes having a social behavior and a second time with one node that is asocial. The preference list of an asocial node will become different from a social node, as the asocial node will not include the neighbors preference list. To make the comparison fair, we will calculate the MSE on the intersection of the asocial and social node preference list as a function of its neighbors' transmission rate. We choose the initial member of the preference list of the asocial and social node to be some states from nodes several hops away from the node.

Figure 22 compares the performance of the asocial and the social nodes. It can be seen that the asocial node reach fast an MSE floor that it cannot puncture. This is coming from the fact that the neighbors add their own preference list to the projection to send. As the neighbors cannot improve their estimation because of the asocial node selfishness, the estimation of the asocial node is affected. However when the node becomes social the performance improves significantly. This validates the proposed cooperative scheme.

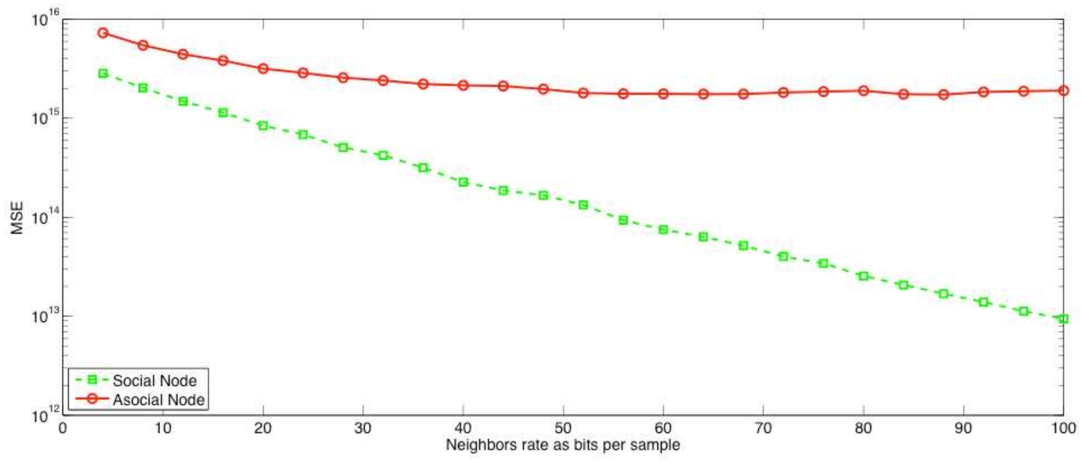


Figure 22: MSE as a function of neighbor rate achieved for a social and an asocial node

As a last validation, we are checking how the constraint on $D^i_{\max}$ works. We are observing that all nodes in our network have a weighted distortion on their preferred list equal to $D^i_{\max}$ irrelevantly of the rate constraint on connected links. However, the number of elements in the preference list depends strongly on the rate constraint. With increase in the rate constraint, the number of elements in the preference list increases. This means that a larger number of nodes will get a specific node state and the diffusion scope of a particular state expands. This shows that the MSE constraint plays his role well, as it is there to control the number of variables and to control that the entire network is not flooded by weak approximation of all node states.

5.2.2 Joint anomaly detection and classification

Let us first present the prototype evaluation of our method in Matlab. The NewNADA algorithm detects an anomalous sliding-window at time 14:07:00 (i.e., at the end of the 21-st sliding-window) due to an anomalous absolute deltoid in the three volume metrics, i.e., ***# SYN***, ***# pkts***, and ***# bytes***. Figure 23 depicts the brutal modification in all of the metrics when the attack is deployed. The Unsupervised Anomaly Detection algorithm is therefore *fed* with all the traffic that belongs to the 21-st sliding-window. Given the traffic aggregation level and the time-scale used by NewNADA, each pattern Y_i ***ϵ Y*** consists of all the IP packets directed to a certain IP network destination address ***IPdst$_i$/24***.

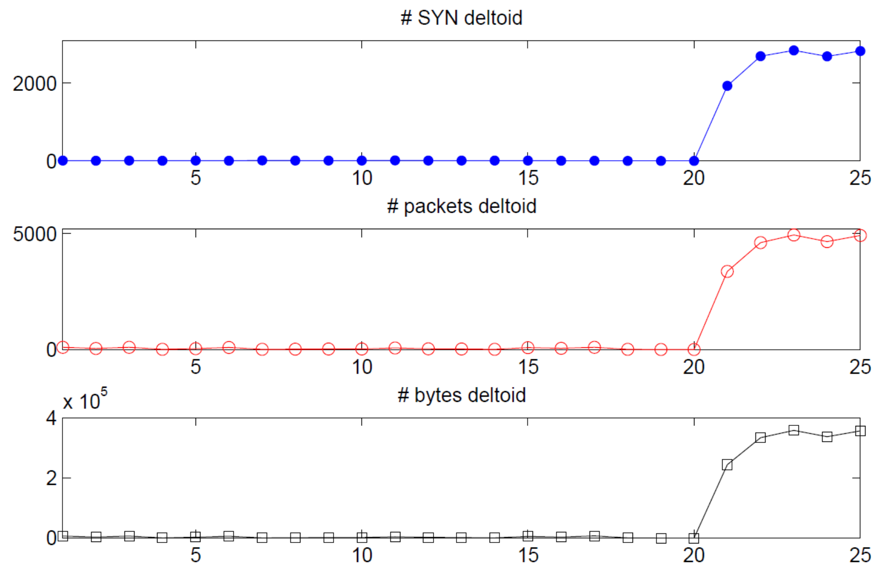


Figure 23: Anomaly detection based on absolute-deltoids change detection in WIDE, using destination IP address /24 as traffic aggregation.

In order to describe each of these patterns, we shall use some of the attributes that were used in [Fernandes 2009] to define classification signatures for general network attacks (DoS, DDoS, and Scans). The main idea of this evaluation is to show that our unsupervised algorithm can automatically detect and build a signature without any previous knowledge about the attack under analysis. Table 5 presents the set of attributes.

Id	Attribute	Description
1	***nDsts***	Nº of different destination IP addresses
2	***nSrcs***	Nº of different source IP addresses
3	***nPkts/nDstPorts***	Ratio of total packets to Nº of different dest. Ports
4	***nSrcs/nDsts***	Ratio of number of sources to number of destinations
5	***nICMP/nPkts***	Proportion of ICMP packets
6	***nEcho/nPkts***	Proportion of Echo Request/Reply packets
7	***nSYN/nPkts***	Proportion of SYN packets
8	***nRST/nPkts***	Proportion of RST packets

Table 5: Set of Attributes for the Feature Space.

The input feature space is therefore $X = \{x_1, x_2, \ldots, x_n\}$, where each x_i is a ***8***-dimensional vector that summarizes the characteristics of the set of IP packets directed to a certain IP network destination address $IPdst_i/24$. The value n is simply the number of different destination address $IPdst_i/24$ found in the anomalous sliding-window.

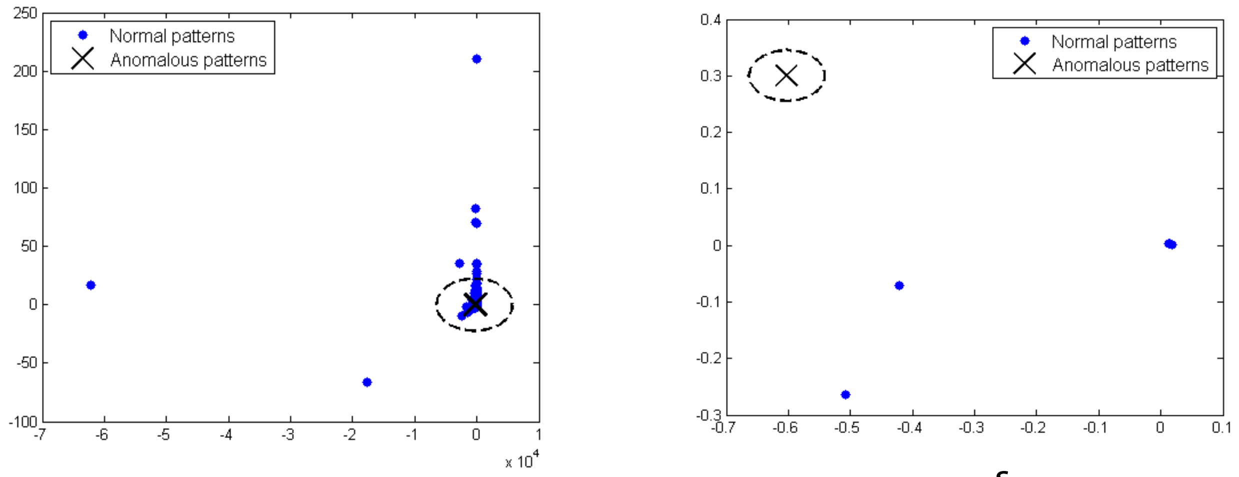
The Unsupervised Anomaly Detection algorithm runs in three consecutive steps. The first step consists in sub-space clustering and evidence accumulation, the second step consists in identifying the anomalous cluster, and the last step consist in automatically generating the filtering rules and selecting the top-M most relevant clusters to build a signature for the detected anomaly. Let us analyze each of the steps.

The sub-space clustering algorithm produces a clustering ensemble $P = \{P_1, P_2, \ldots, P_N\}$ that contains the $N = (8*7)/2 = 23$ partitions. The information provided by these N partitions is used by the evidence accumulation algorithm to produce a similarity matrix we shall refer to as ***S***S_{EAC}, which represents some notion of distance between the ***n*** patterns Y_i ϵ ***Y***.

In order to appreciate the great advantage of using S_{EAC} w.r.t. a traditional clustering approach, based on the information provided by the complete feature space ***X***, we shall compute a similarity matrix $S_{traditional}$ for the ***n*** patterns represented in ***X***. The ***(n * n)*** similarity matrix $S_{traditional}$ accounts for the inter-pattern distance between the ***n*** patterns, where element ***(i,j)*** stands for the Euclidean distance between patterns ***i*** and ***j***:

$$S_{traditional}(i,j) = \sqrt{\sum_{k=1}^{d} (x_i(k) - x_j(k))^2}$$

Figure 24 presents a two-dimensional plot of the information provided by both similarity matrices S_{EAC} and $S_{traditional}$, using a Multi-Dimensional Scaling (MDS) analysis. MDS permits to explore similarities in high-dimensional data, assigning a location to each couple of patterns described in a similarity matrix into a low-dimensional space. In order to assess the power of discrimination provided by each similarity matrix, we assume in Figure 24 that the anomalous patterns are known in advance.



(a) MDS of $S_{traditional}$ (b) MDS of S_{EAC}

Figure 24: Multi-Dimensional Scaling for (a) $S_{traditional}$ and (b) S_{EAC}. The presence of the anomalous cluster becomes evident in the S_{EAC} similarity matrix.

In the case of $S_{traditional}$, we can appreciate that the anomalous patterns are mixed-up with the normal ones, and the discrimination using all the attributes at the same time becomes difficult. In the case of S_{EAC}, the anomalous patterns are perfectly isolated from the rest of the patterns, providing a powerful discrimination measure of similarity.

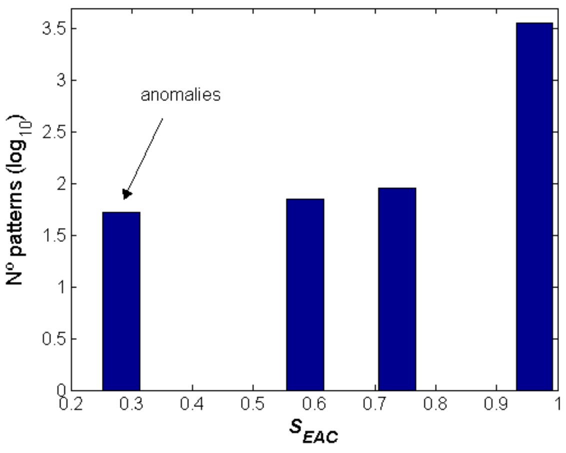


Figure 25: Distribution of inter-pattern similarity in S_{EAC}.

The second step of the algorithm consists in detecting the anomaly. As we explained before, once we have built S_{EAC} we can apply any clustering algorithm to this new similarity matrix so as to find a consistent data partition. In our anomaly detection application, we are not particularly interested in building a partition of the data but in identifying the anomalous patterns, which by definition will lie in small and well-formed clusters. The reader should remember that S_{EAC} gives high priority to small clusters, through the importance function $w_k(n(k))$. Therefore, the detection simply consists in identifying the most similar patterns in S_{EAC}.

Figure 25 shows a histogram on the distribution of inter-pattern similarity, according to S_{EAC}. Select the most similar group of patterns results in a compact cluster of 53 patterns. A further analysis of the IP packets that compose each of these patterns reveals the same origin IP address, which corresponds to the attacker's host 204.243.26.29. Each individual pattern corresponds to a flow of SYN packets directed towards a different destination IP network address 162.225.xxx.0/24.

The reader should note the astonishing effectiveness of this unsupervised detection approach, which has perfectly detected and isolated the distributed attack in a completely *blind fashion*, without assuming any particular traffic model, detection threshold, significant clustering parameters, or even clusters structure beyond a basic definition of what an anomaly is.

The last step of the Unsupervised Anomaly Detection algorithm consists in automatically building easy-to-understand and easy-to-visualize discrimination rules that can be used to construct a signature for the anomaly that has been detected and isolated. As we explained before, we take advantage of the low-dimensionality clustering ensemble $P = \{P_1, P_2, \ldots, P_N\}$ built during the sub-space clustering step to produce both absolute and splitting rules.

Figure 26 depicts some of the partitions P_i where both absolute and splitting rules were found. The partition depicted in Figure 26.a presents three clusters, one of them exclusively composed of anomalous patterns. This partition builds two rules; the former is a splitting rule, relative to the number of different destination IP addresses found in the IP packets aggregated into each pattern. Given the distributed nature of the network scan attack, it is clear that the number of destination IPs must be much larger than in the case of normal-operation traffic. The latter is an absolute rule, in which almost every pattern of the anomalous cluster has the SYN flag activated for all of its IP packets. This rule makes perfect sense, because the network scan is a SYN scan.

The signature produced by these two rules can be simply expressed as ***<(nDsts >*** λ_1 ***) & (nSYN/nPkts == 1)>***. Note however that not every anomalous pattern depicted in Figure 26.a will be correctly classified using this signature, basically because there are some patterns for which attribute ***nSYN/nPkts*** is not strictly equal to 1.

The partition depicted in Figure 26.b shows two clusters, but in this case the smallest one contains not only anomalous patterns but also some other patterns not identified as such. Given that the filtering rules are defined in a per-cluster basis, a new splitting rule two separate both clusters is generated for attribute ***nSYN/nPkts***, which permits to *relax* the absolute rule previously defined. The new rule for ***nSYN/nPkts*** is therefore ***<(nSYN/nPkts >*** λ_2***)>***. Given that this new rule covers the previous one, the absolute rule ***<(nSYN/nPkts == 1)>*** is discarded. Another rule is also identified in this partition, consisting in an absolute rule for attribute ***nSrcs***. This absolute rule specifies that all the packets come from the same origin IP address, which corresponds to the attacker's IP address.

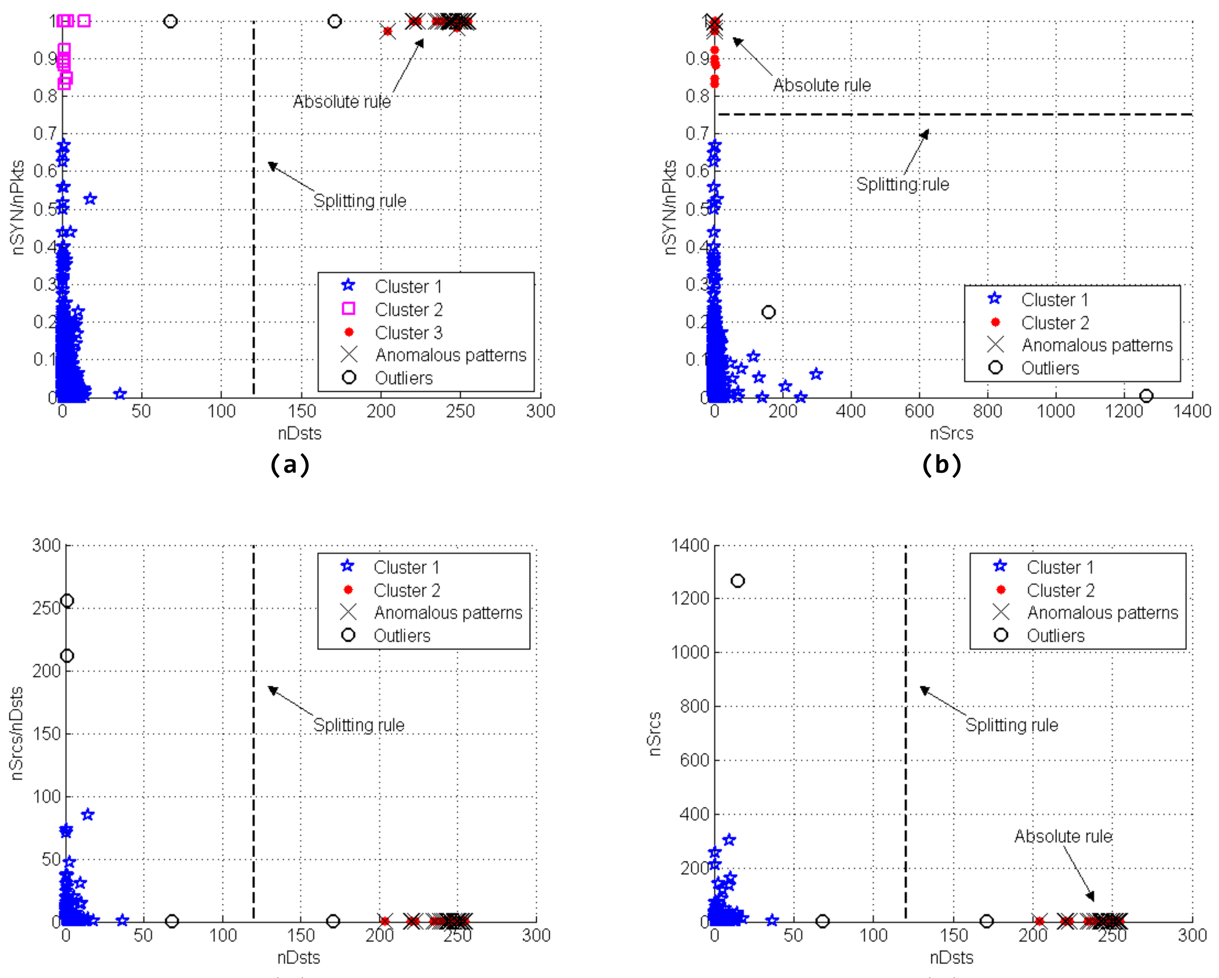


Figure 26: Absolute and splitting rules for anomaly signature.

Adding this rule to both previously generated rules produces a new anomaly signature that can be expressed as:

$$<(nDsts > \lambda_1) \ \& \ (nSYN/nPkts > \lambda_2) \ \& \ (nSrcs == 1)>$$

The two other partitions depicted in Figure 26.c and Figure 26.d produce rules that are already contained in this signature, which further verifies its relevance. The most interesting observation of this step of the algorithm is that the generated anomaly signature permits to effectively isolate all of the patterns that conform to the network SYN scan, correctly classifying all the corresponding IP packets that are aggregated into each pattern.

Let us now present the evaluation of the second traffic scenario, consisting of a TCP SYN DDoS attack. Table 2 presents the set of attributes that are use to describe each of the traffic patterns in the feature space.

Id	Attribute	Description
1	***nDsts***	Number of different destination IP addresses
2	***nPkts/nDst***	Ratio of number of packets to number of different destinations
3	***nDstPorts***	Number of destination ports
4	***nSYN/nPkts***	Proportion of SYN packets
5	***nPkts/sec***	Number of sent packets per second

Table 6: Set of Attributes for the Feature Space.

Given this set of attributes, the sub-space clustering algorithm produces a clustering ensemble $\boldsymbol{P} = \{P_1, P_2 \ldots P_N\}$ that contains $\boldsymbol{N = (5*4)/2 = 10}$ partitions.

The detection of the attack is done in exactly the same way as for the SYN network scan attack; hence we shall focus the attention on the last step of the Unsupervised Anomaly Detection algorithm, consisting in building the filtering rules and the corresponding attack signature.

Figure 27: **Absolute and splitting rules for a TCP SYN DDoS attack.** depicts some of the partitions P_i where both absolute and splitting rules where found (those ranked with highest Fisher Score). The partition depicted in Figure 27.a presents two clusters, one of them exclusively composed of anomalous patterns. This partition builds three different rules; the former is an absolute rule, in which every pattern of the anomalous cluster has the SYN flag activated for all of its IP packets. As before, this rule makes perfect sense, because the DDoS attack is a TCP SYN DDoS attack, which uses exclusively SYN packets. The second rule is a splitting rule also relative to the proportion of SYN packets; as in the case of the network SYN scan, this splitting rule imposes that not necessarily every packet of the DDoS attack must be a SYN packet, but that it is sufficient to have a fraction of SYN packets larger than a certain threshold λ_1. The last filtering rule is an absolute rule regarding the number of destination hosts, which must be equal to 1. Therefore, the predominant filtering rule obtained from Figure 27.a can be expressed as $<(nSYN/nPkts > \lambda_1) \ \& \ (nDsts == 1)>$.

The partition depicted in Figure 27.b shows three clusters. If we consider the proportion of SYN packets, it seems quite clear that both clusters number 2 and number 3 are composed of anomalous patterns. However, we can appreciate that only those patterns belonging to cluster number 3 have been detected as anomalous by our algorithm. If we now consider the number of packets per second sent by each host, we can see that those patterns detected as anomalous have a packet rate remarkably high w.r.t. those patterns in cluster 2, which explains the distinction produced by the clustering algorithm. Indeed, the algorithm has detected those anomalous patterns that were more aggressive during the DDoS attack, sending more than λ_2 packets per second to the victim host. Therefore, the additional filtering rule obtained from this partition is ***<(nPkts/sec > λ_2)>***.

The last filtering rule comes from the partition depicted in Figure 27.c, where almost every anomalous pattern has the same single destination port. This translates into an absolute rule of the form ***<(nDstPorts == 1)>***.

Combining the filtering rules produces a new anomaly signature for the TCP SYN DDoS attack, which can be simply expresses as:

<(nDsts == 1) & (nDstPorts == 1) & (nSYN/nPkts > λ_1) & (nPkts/sec > λ_2)>

This signature is able to correctly isolate those patterns that represent the most aggressive hosts of the botnet where the DDoS attack comes from.

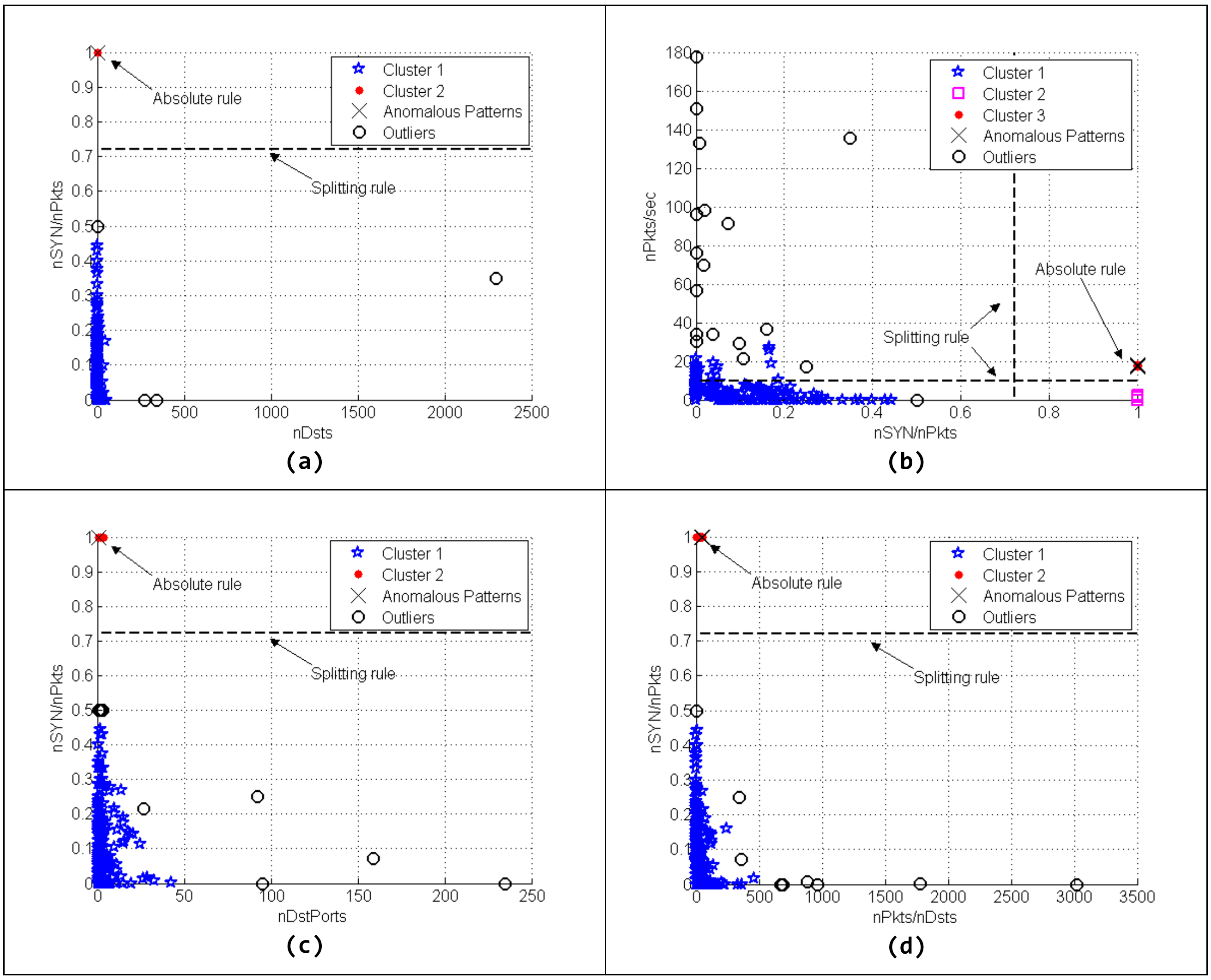


Figure 27: **Absolute and splitting rules for a TCP SYN DDoS attack.**

To conclude with the experimental section, we shall present the evaluation of the iLAB implementation, previously described. The implementation of the NewNADA algorithm and the Unsupervised Anomaly Detection module is done in Objective-CAML, a general-purpose programming language developed by INRIA, which is specially designed with program safety and reliability in mind. For high performance, a native-code compiler permits to directly integrate these implementations in numerous architectures.

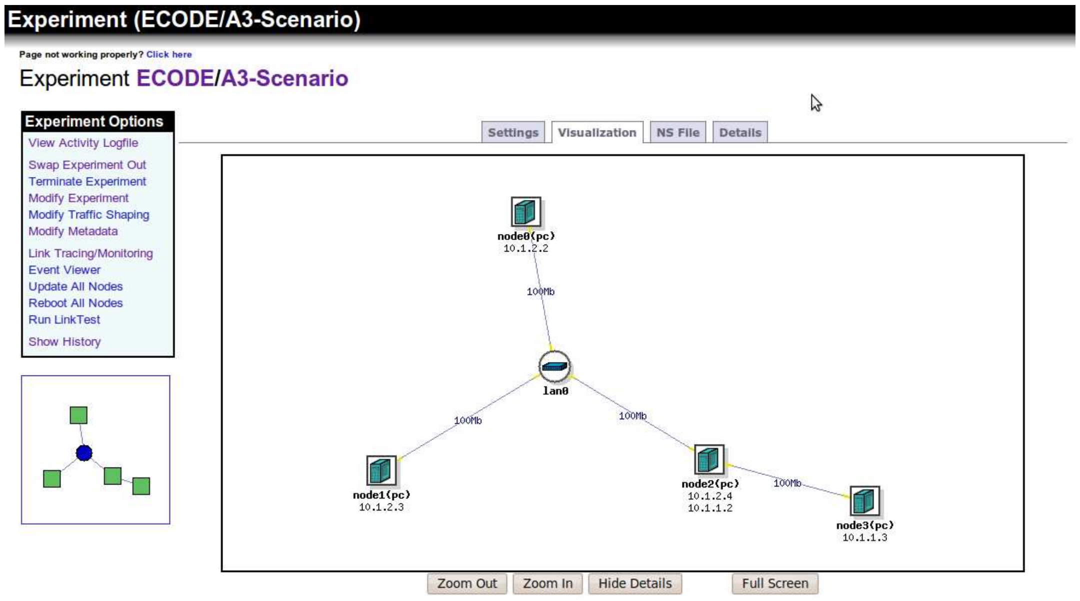
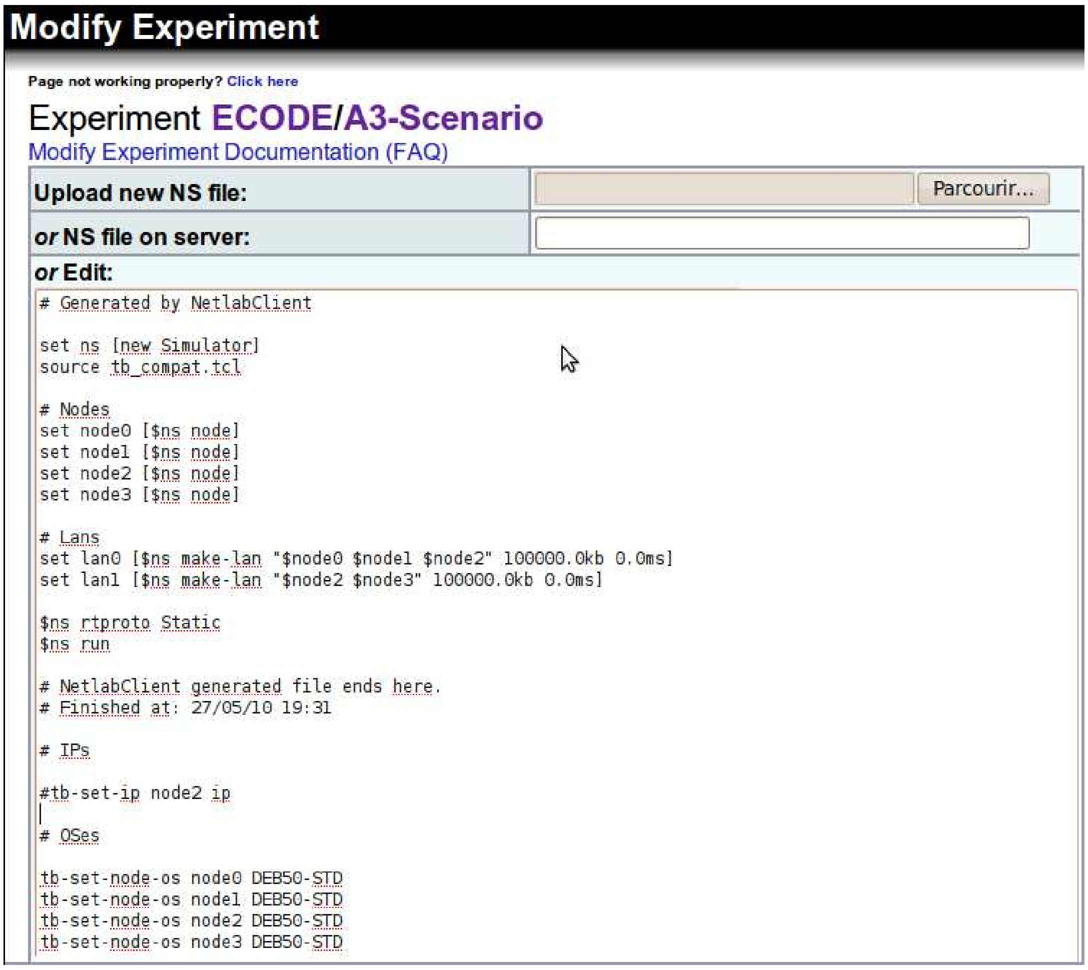


Figure 28: iLAB Topology for Experimentation.

The iLAB framework provides a graphical interface to setup the experimentation, using a Virtual Wall application. Figure 28 shows a screenshot of the topology used to reproduce and detect the distributed network SYN scan attack, using real traffic traces from MAWI. In this topology, hosts with labels “node0” and “node1” replay a MAWI trace free of anomalies as background traffic, sending packets to host labeled as “node3” with the original IP addresses. The “node3” host simply acts as a gateway for all the traffic directed towards the original IP addresses of the MAWI trace. The anomaly detection algorithms run on host labeled as “node2”. In order to simulate the network SYN scan attack, “node0” replays a trace composed of only those traffic packets from the real network scan, previously extracted from the MAWI trace.

The topology is implemented through a Network Simulator tcl file, specifying the number of hosts, their interconnections, the routing tables as well as the operating systems that will run on each host. In our case, we have run our algorithms in a Linux Debian-based OS distribution. Figure 29 shows a screenshot of the NS configuration file.

As we claimed before, the results of the detection of the network scan attack as well as the automatic generation of a signature that were obtained in the iLAB experimentation are exactly the same as those already presented for the prototype in Matlab, simply because the traffic traces that were used are exactly the same. We will therefore focus the attention on a more representative performance indicator for an on-line anomaly detection framework as the one we have implemented: execution time.


Modify Experiment
Page not working properly? Click here
Experiment ECODE/A3-Scenario
Modify Experiment Documentation (FAQ)
Upload new NS file:
Parcourir...
or NS file on server:
or Edit:

```
# Generated by NetlabClient

set ns [new Simulator]
source tb_compat.tcl

# Nodes
set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]

# Lans
set lan0 [$ns make-lan "$node0 $node1 $node2" 100000.0kb 0.0ms]
set lan1 [$ns make-lan "$node2 $node3" 100000.0kb 0.0ms]

$ns rtproto Static
$ns run

# NetlabClient generated file ends here.
# Finished at: 27/05/10 19:31

# IPs

#tb-set-ip node2 ip

# OSes

tb-set-node-os node0 DEB50-STD
tb-set-node-os node1 DEB50-STD
tb-set-node-os node2 DEB50-STD
tb-set-node-os node3 DEB50-STD
```

Figure 29: iLAB Topology Generation via NS File.

Table 7 presents the execution time involved in the processing and detection of the network scan in a temporal sliding-window of 20 seconds long. The execution time corresponds to the time involved in processing all the traffic patterns that belong to the 21-st sliding-window. Execution time is measured in different platforms, including a host provided by iLAB (Id Test 1) and a host of our LaasNetExp platform (Id Test 2). In order to compare the performance of the algorithms implemented in native-code with the prototype implementation, we also include the execution time measured in Matlab to process exactly the same temporal sliding-window (Id Test 3).

Id Test	Platform	Execution Time (s)
1	AMD Opteron Dual-Core, 2.0 GHz, 4Go RAM, Debian Lenny 5.0	90.8
2	Intel Dual-Core, 3.2 GHz, 4Go RAM, Ubuntu Karmic Koala 9.10	40.5
3	Intel Quad-Core, 2.4 GHz, 2Go RAM, Fedora Constantine 12	24.3

Table 7: Execution time for a 20s sliding window and different platforms.

In the three cases, the execution time is higher than the length of the temporal sliding-window under analysis. From our tests, we can see that the prototype Matlab implementation outperforms the native-code implementation, subject to the particular platform where the algorithms run.

Our native-code implementation was originally designed for detection performance, paying less attention to code-optimization regarding execution time. In one hand, these results clearly suggest that the native-code implementation should be improved; on the other hand, we can expect an execution time that will be lower than the prototype Matlab implementation, which is already good enough for a non-optimized prototype in a platform-code-based application like Matlab.

6. Update on Section 6 of D3.2: Recommendation for integration into common ECODE architecture

In this section, we describe more in details the new recommendations for the integration of the different use cases related solutions within the common ECODE architecture. This section updates the one numbered 6 in D3.2.

6.1 Use case a1: Adaptive traffic sampling

Regarding the Adaptive traffic sampling architecture, we mainly provide new recommendations related to the following interfaces of the common ECODE architecture:

- The CM interface which is responsible for communication between the TCI (Translation and Communication Interface) and the Monitoring Points (MP).
- The TR Interface which represents the feedback from the TCI to the Machine Learning Engine (MLE).
- The DP+TD interfaces which provide a way for passing the information from the MLE to the TCI.

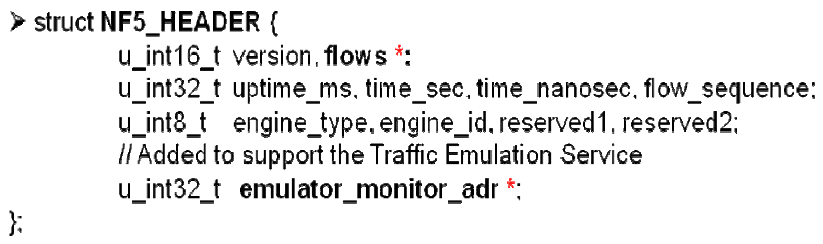
6.1.1 CM Interface

- *Types of monitoring information used as input to the Machine Learning engine:*

  Our traffic monitoring and sampling service will be incorporated within the ME (Monitoring Engine) component of the global ECODE architecture, and will rely on the CM (Cognitive Monitoring) interface towards exporting the collected statistics to the MLE. The latter statistics consists of NetFlow Reports and more specifically of the Cisco Netflow(tm) version 5 packet format (more details on http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/nfc/nfc_3_0/nfc_ug/nfcform.htm). Note that our ME uses by default NetFlow version 5 reports; however, it also supports exporting reports according to the NetFlow version 9 format.

- *The structure and syntax of the monitoring information received from MP:*

  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 *.

- *Frequency at which information are receive from the various monitors:*

  The adaptive traffic sampling and monitoring architecture that we are proposing relies mainly on a passive monitoring solution. Hence, once our monitoring and sampling service is started, it sends back NetFlow reports via the **CM** interface as soon as one of the initially configured conditions is satisfied. These conditions are: the size of the buffer holding the NetFlow records and the expiration timers associated with the different types of flow records (TCP, UDP...).

- *Special configuration issues:*

  Our passive monitoring engine relies on a set of text configuration files, which should be setup before starting the daemon.

6.1.2 TR Interface

- *Attributes/Objects (output from the syntax translator) passed to the representation module:*

  Once the NetFlow reports are received by the TCI, the later 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 *observation 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.

- *Structure of the messages (combination of attributes):*

  The structure of a NetFlow report is as follows: [**NF5_HEADER**, **NF5_FLOW**, **NF5_FLOW**, **NF5_FLOW** …]

- *Message sequencing and ordering:* Apply the FIFO policy for message ordering.

- *Message rate (pacing/flow control and/or rate limitation expected):*NetFlow Reports are received in a sporadic way.

- *Errors need to be recuperated:* All the errors are managed at the ME. None is reported to the MLE.

6.1.3 PD+DT Interfaces

- *Information passed to the TCI over the PD and DT interfaces:*

  Our cognitive system will need to configure the sampling rates in our passive monitoring points 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 cognitive algorithm will need to change the sampling rate of a given ME at a given network interface. The latter communication will be ensured via the DP+TD interfaces to the TCI and then to the ME via CM interface. The generic 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)]

  We managed to run a control server within our ME which will listen for incoming packets at a given control port. The same control port should be used by the TCI in order to forward control messages coming from the ML engine towards the ME via the CM interface. Once the control message is received by the ME, it is parsed and the sampling rate of the corresponding interface is changed according to the new received value.

  We describe next the current hard representation of the control message that we are using. This control message consists of the following string:

  P1#P2#Network_Interface

  where:
  - P1, P2 and Network_Interface are also string values.
  - The new sampling rate value is equal to P1/P2 knowing that P2 > 0.
  - And Network_Interface follows the following format: a.b.c.d which represents the monitoring interface for which we will change the sampling rate.

6.2 Use case a3: Cooperative traffic anomalies and attacks detection

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 of D3.2 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 (as in Tables 5 and 6 for instance), 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.

7. Conclusion

This deliverable outlines the methods proposed for use cases a1 and a3 and details the validation results that were obtained through extensive simulations and network emulation campaigns. Through the use case a1, we presented a network-wide monitoring system that adopts an adaptive centralized approach to coordinate responsibilities across monitors by adjusting their sampling rates. Our system extends the existing NetFlow-like monitoring tools with a cognitive engine that correlates collected measurements from all routers to infer a better global view of the network traffic. Moreover, it automatically reconfigures the sampling rates in the different monitors according to the monitoring application requirements and resource consumption constraints. By using our system, the network operator just has to select a measurement task and a monitoring resource constraint (TO: Threshold of Overhead). Our self configuring system will then iterate measurements and adjust sampling rates in small steps in order to address the tradeoff between monitoring accuracy and overhead.

We validated the performance of our system over a real platform we developed for the purpose of the study. Our platform replays TcpDump traffic traces and deploys real flow monitoring tools. For the case of estimating the full traffic matrix, experimental results proved the ability of our system to continuously improve the monitoring accuracy while limiting the overhead to its target value. Moreover, the system provides a fair allocation of sampling rates over monitors so that measurement errors are homogenously distributed among flows independently of their volumes. Compared to static edge configuration, our network-wide adaptive system has shown its advantages in better capturing network flows especially for small flows.

Our future research will focus on the validation of our approach with more applications. The counting of flows, the tracking of some user-specific flows, the detection of anomalies, and the identification of greedy users are among the applications we want to cover. The distribution of the control and adaptive overhead tuning are other interesting objectives to realize.

As regards the Unsupervised Anomaly Detection algorithm that we have proposed within use case a3, this deliverable presents many interesting advantages w.r.t. previous proposals in the field of anomaly Detection.

Through the use of NewNADA, the algorithm can use different levels of traffic aggregation to construct traffic patterns in ***Y***, improving detection results, particularly as regards detection sensitivity to low intensity anomalies. For example, in [Fernandes 2009] we have used aggregation at the victim’s network level, using destination IP address and various network masks (e.g., /0, /8, /16, and /24) to aggregate traffic packets. This flexibility provided by NewNADA increases the performance of the unsupervised detection.

The unsupervised learning module uses exclusively unlabelled data to detect traffic anomalies, without assuming any particular model or any canonical data distribution. This allows detecting new previously unseen network attacks. Despite using ordinary clustering techniques to identify traffic anomalies, the algorithm avoids the lack of robustness of general clustering approaches, by combining the notions of sub-space clustering and multiple evidence accumulation.

Using sub-space clustering, we are able to perform clustering in low-dimensional feature spaces, avoiding the “curse of dimensionality” problem. Low-dimensionality clustering can be efficiently performed by density-based clustering algorithms, which are well known to be very powerful to discover clusters of arbitrary shapes [Jain10].

The sub-space clustering approach also permits to obtain easy-to-interpret and tractable results, providing insight and explanations about the detected anomalies to a human network manager. Additionally, clustering in low-dimensional feature spaces provides results that can be visualized by standard techniques, which improves the assimilation of results.

We have verified the effectiveness of this unsupervised detection approach to detect and isolate real distributed network attacks in a completely blind fashion, without assuming any particular traffic model, detection threshold, significant clustering parameters, or even clusters structure beyond a basic definition of what an anomaly is. Additionally, we have tested the performance of the algorithm both in a prototype Matlab implementation as well as in a real network topology represented by iLAB, obtaining the same detection accuracy in both cases. However, the method still requires validation as regards the detection of a larger variety of network attacks, and the native-code implementation should be optimized in order to apply the algorithms for on-line unsupervised anomaly detection. A comprehensive evaluation of the algorithm is part of our current on-going work.

References

[Abilene] Abilene or internet2: The us research and academic backbone. http://www.internet2.edu/network/.

[Agrawal98] R.Agrawal, J.Gehrke, D.Gunopulos and P.Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications", Proc. ACM SIGMOD: Management of Data Conference, 1998.

[Cukier73] R.Cukier, C.Fortuin, K.Shuler, A.Petshek, and J.Schaibly. "Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients", The Journal of Chemical Physics, 59:3873–3878, 1973.

[Cukier75] R.Cukier, J.Schaibly, and K.Shuler, "Study of the sensitivity of coupled reaction systems to **uncertainties** in rate coefficients. iii analysis of the approximations", The Journal of Chemical Physics, 63, 1975.

[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.

[Duffield05] N.Duffield, C.Lund, and M.Thorup, "Optimal combination of sampled network measurements". In Proc. IMC 2005 2005.

[Duffield02] N.Duffield, C.Lund, and M.Thorup, "Properties and prediction of flow statistics from sampled packet streams", In Proc. of IMC 2002.

[Denning87] E.Denning, "An Intrusion-Detection Model", IEEE Transactions on Software Engineering, 13(2):222-232, 1987.

[Dubes76] R.Dubes and A.K.Jain, "Clustering Techniques: The User's Dilemma", Pattern Recognition, 8:247-260, 1976.

[Duda01] R.O.Duda, P.E.Hart, and D.G.Stork, "Pattern Classification, second edition", Wiley Publisher, 2001.

[Eskin02] E.Eskin, A.Arnold, M.Prerau, L.Portnoy, and S.Stolfo, "A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabelled Data", Applications of Data Mining in Computer Security, Kluwer Publisher, 2002.

[Ester96] M.Ester, H.P.Kriegel, J.Sander, and X.Xu, "A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", Proc. ACM SIGKDD, 1996.

[Everitt93] B.Everitt, "Cluster Analysis", John Wiley & Sons, 1993.

[Fernandes 2009] G.Fernandes and P.Owezarski, "Automated classification of network traffic anomalies", SecureComm'09, Athens, Greece, 15-17 September 2009.

[Fraley98] C.Fraley and A.E.Raftery, "How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis", The Computer Journal, 41(8):578-588, 1998.

[Fred02] A.Fred and A.K.Jain, "Data Clustering Using Evidence Accumulation", Proc. of 16th Int. Conf. Pattern Recognition, 2002.

[Fred05] A.Fred and A.K.Jain, "Combining Multiple Clusterings Using Evidence Accumulation", IEEE Trans. Pattern Analysis and Machine Intelligence, 27(6):835-850, 2005.

[Jain99] A.K.Jain, M.N.Murty, and P.J.Flynn, "Data Clustering: A Review", ACM Computing Surveys, 31(3):264-323, 1999.

[Jain10] A.K.Jain, "Data Clustering: 50 Years Beyond K-Means", Pattern Recognition Letters, 31(8):651-666, 2010.

[Geant] Geant: The european research and academic backbone. http://www.geant.net/.

[Krifa10] A.Krifa, I.Lassoued, and C.Barakat. "Emulation platform for network wide traffic sampling and monitoring". TRAC, 2010.

[Kaufman90] L.Kaufman and P.J.Rosseeuw, "Finding Groups in Data: An Introduction to Cluster Analysis", John Wiley & Sons, 1990.

[Liu98] H.Liu and H.Motoda, "Feature Selection for Knowledge Discovery & Data Mining", Kluwer Academic Publishers, 1998.

[Lu01] Y.Lu and S.Mohanty. Sensitivity analysis of a complex, proposed geologic waste disposal system using the Fourier amplitude sensitivity test method. Reliab. Eng. syst. Safe., 72, 2001.

[Leung05] K.Leung and C.Leckie, "Unsupervised Anomaly Detection in Network Intrusion Detection Using Clustering", Proc. of Australasian Computer Science Conference, ACSC05, 2005.

[Mawi] Mawi working group traffic archive: http://tracer.csl.sony.co.jp/mawi/.

[Portnoy01] L.Portnoy, E.Eskin, and S.Stolfo, "Intrusion Detection with Unlabelled Data Using Clustering", Proc. of ACM DMSA Workshop, 2001.

[Parsons04] L.Parsons, E.Haque, and H.Liu, "Subspace Clustering for High Dimensional Data: a Review", ACM SIGKDD Explorations Newsletter, 2004.

[Softflowd] Softflowd flow-based network traffic analyser: http://www.mindrot.org/projects/softflowd/.

[Saltelli00] A.Saltelli, K.Chan, and M.Scott, "Sensitivity analysis", Wiley, 2000.

[Saltelli04] A.Saltelli, S.Tarantola, F.Campolongo, and M.Ratto, "Sensitivity analysis in practice: A guide to assessing scientific models". Wiley, 2004.

[Saltelli99] A.Saltelli, S.Tarantola, and K.P.-S.Chan, "A quantitative model-independent method for global sensitivity analysis of model output", Technometrics, 41:39–56, 1999.

[Strehl02] A.Strehl and J.Ghosh, "Cluster Ensembles – A Knowledge Reuse Framework For Combining Multiple Partitions", Journal on Machine Learning Research, 3:583-617, 2002.

[Theodoridis99] S.Theodoridis and K.Koutroumbas, "Pattern Recognition", Academic Press, 1999.

[Tcpdpriv] Tcpdpriv: A program for eliminating confidential information from packets. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html/.

[Wide00] K.Cho, K.Mitsuya and A.Kato, "Traffic Data Repository at the WIDE Project", USENIX Annual Technical Conference, 2000.