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

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D2.1 - *"Cognitive Engine - Experimental Network and System Architecture"*

Project acronym: **ECODE**
Project full title: **Experimental Cognitive Distributed Engine**
Proposal/Contract no.: **223936**

Date of preparation of Deliverable: **October, 2009**

Document properties

Document Number:	ICT-2007-1.6-223936 ECODE-D2.1
Document Title:	Cognitive Engine - Experimental Network and System Architecture
Document responsible:	Benoit Donnet (UCL), Dimitri Papadimitriou (ALB)
Author(s)/editor(s):	Benoit Donnet (UCL), Dimitri Papadimitriou (ALB), Olivier Bonaventure (UCL), Pierre Dupont (UCL), Juan Pablo Narino Mendoza (UCL), Damien Saucez (UCL), Philippe Owezarski (CNRS), Johan Mazel (CNRS), Yann Labit (CNRS), Kavé Salamatian (ULANC), Ing-Jyh Tsang (ALB), Werner Van Leekwijck (ALB), Amir Krifa (INRIA), Chadi Barakat (INRIA), Konstantin Avratchenkov (INRIA), Imed Lassoued (INRIA), Guy Leduc (ULG), Wouter Tavernier (IBBT)
Dissemination level:	PU
Security Type:	PU
Status of the Document:	Final
Version & Revision History	v1.1 – October 2009 v1.0 – February 2009

Table of Content

1. Description of the Document

This document is the ECODE deliverable D2.1. It provides an insight into the experimental network and system architecture we will develop throughout the whole project duration.

In this document, we provide a general overview of machine learning techniques as well as a complete description of the ECODE architecture at two levels: system and network. Note that this architecture represents our current view of the project. Throughout the project duration, we will evaluate how feasible is this architecture and adapt it according to the experimental results. Our architecture is thus not fixed but rather a progressive process. The objective is to have, by the end of the project, a workable (and implemented) architecture.

As ECODE is an experimentally-driven project, this document provides an accurate documentation of each use case, as well as how each use case interacts with the Machine Learning Engine we introduce (initially referred to as Cognitive Engine). In this document, we chose to cluster the various use cases in four groups:

- *Security and monitoring*. This corresponds to use cases a1, a2, and a3, i.e., the development of an autonomous system for network monitoring, traffic management, and anomalies detection.
- *Routing*. This corresponds to use cases c1 and b2, i.e., the development of a solution for speeding up the BGP path exploration process and allowing fast network recovery.
- *Path selection*. This corresponds to use case b1, i.e., the development of a solution for allowing application (of any kind) to rank paths according to particular criteria.
- *Accountability*. This corresponds to use case b3, i.e., the development of a solution for correlating profiles with subscribers' usage and their impact on the network resources.

Finally, we also provide an analysis grid that will be used throughout the whole project for its evaluation. The analysis will be on functionalities and performance.

This deliverable is organized as follows: Sec. 2 provides a list of the various acronyms used in this document; Sec. 3 introduces the document; Sec. 4 provides a large overview of machine learning techniques; Sec. 5 discusses the ECODE architecture; Sec. 6 details the security and monitoring use cases, while Sec. 7 is dedicated to routing use cases, Sec. 8 to path selection, and Sec. 9 to accountability; Sec. 10 provides the analysis grid; finally, Sec. 12 concludes this document and summarizes its main achievements.

2. Acronyms

AAA	Authentication Authorization and Accounting
ADS	Anomaly Detection System
AS	Autonomous System
BGP	Border Gateway Protocol
CPE	Customer Premise Equipment
DAG	Data Acquisition and Generation
DDoS	Distributed Denial-of-Service
DHCP	Dynamic Host Configuration Protocol
DoS	Denial-of-Service
DPI	Deep Packet Inspection
DR	Designated Router
E2E	End-to-End
ECN	Explicit Congestion Notification
EKF	Extended Kalman Filter
EM	Expectation Maximization
ERF	Extensible Record Format
FIB	Forwarding Information Base
FUP	Fair Usage Policies
GPS	Global Positioning System
ICMP	Internet Control Message Protocol
ICMP-SEQ	Number of Sequence ICMP
ICS	Internet Coordinate System
IDIPS	ISP-Driven Informed Path Selection
IDS	Intrusion Detection System
IP	Internet Protocol
ISP	Internet Service Provider
KIB	Knowledge Information Base
KLT	Karhunen–Loève Transform
KP	Knowledge Plane
LISP	Locator/Identifier Separation Protocol
LM	Learning Method
LRD	Long Range Dependence
LS	Link State
MA	Multi-Access
MCMC	Monte Carlo Markov Chain
MP	Measurement Point
MPLS	Multi-Protocol Label Switching
MRAI	Minimum Route Advertisement Interval
MTR	Multi-Topology Routing
OCR	Optical Character Recognition
OSPF	Open-Shortest Path First
P2P	Peer-to-Peer
PBA	Profile-Based Accountability
PCA	Principal Component Analysis
PDU	Protocol Data Unit
PIB	Path Information Base
PIC	Path Information Collector
PLR	Packet Loss Ratio

PPS	Pulse Per Second
QoS	Quality of Service
RIB	Routing Information Base
ROC	Receiver Operation Characteristic
RTT	Round-Trip Time
SRG	Shared Risk Group
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TIV	Triangular Inequality Violation
TLS	Transport Layer Security
TTL	Time to Live
UDP	User Datagram Protocol
WAN	Wide Area Network

3. Introduction

Since the early 90's, the Internet has known an impressive growth and is, nowadays, victim of its own success: its size and scale render the deployment of new network technologies very difficult while it is experiencing increasing demand in terms of connectivity and capacity. Indeed, although the current Internet does work and is still capable of fulfilling its current missions, it suffers from relative "ossification", a condition where technological innovation meets natural resistance, as exemplified by the lack of wide deployment of technologies such as multicast or the new version of the IP protocol (IPv6).

As a result of the Internet growth and the increasing communication requirements, a lot of incremental solutions have been progressively developed and deployed to allow the Internet to cope with the increasing demand in terms of user connectivity and capacity (see for instance [14]). There is, however, a growing consensus among the scientific and technical community that the current methodology of ``patching" the Internet will not be able to sustain its continuing growth at an acceptable cost and performance. Indeed, starting from five base design principles (modularization by layering, connectionless packet forwarding, end-to-end principle, uniform inter-networking principle and simplicity principle), the Internet has progressively become an infrastructure that is architecturally more complex to operate. This is mainly due to various layer violations (e.g., complex cross-layer design) to supposedly optimize network and system resource consumption, the proliferation of various sub-layers (e.g., Multi-Protocol Label Switching [1], or Transport Layer Security [2]) to expectedly compensate for intrinsic shortcoming in terms of forwarding performance and security functionality, IP addressing space overload (including network graph locator, node identity, connection termination), and routing system scalability and quality limitations (e.g., BGP path exploration and oscillations [3]) to name a few. This complexity progressively impacts the Internet robustness and reliability and in turn impacts its scalability (resulting from the violation of the Occam's razor simplicity principle also known as the ``Robustness through simplicity" principle [4]).

Hence, although the design principles of the Internet are still valid, there is growing evidence that the resulting design components, as defined today, face certain objective technical limits. On the other hand, certain objectives of the Internet are no longer adapted to users' new expectations and behaviors. In other terms, the current Internet architecture is progressively reaching a saturation point in meeting increasing users' expectations and behaviors as well as progressively showing its inability to efficiently respond to new technological challenges (in terms of security, mobility, availability, and manageability) and socio-economical challenges. Even worse, misguided attempts to sustain the Internet growth resulted into progressive violation and erosion of the end-to-end principle. Sacrificing the end-to-end principle has in turn resulted in decreasing the Internet availability, negatively impacting its robustness and scalability as well as making its manageability more complex. Over time, the erosion of the end-to-end principle has also resulted in the proliferation of peer-to-peer and application-specific overlay networks that are progressively substituting the end-to-end IP networking layer by an end-to-end applicative communication layer. Indeed, many new applications provide their own path selection to ensure proper connectivity and quality, resulting in an ineffective network level resources use [30, 31, 32, 33].

With an increasing reliance on the Internet infrastructure for economic and social activities, the impact of network-wide terror in the form of worms or viruses is also increasing [34, 35, 36]. Improving the security and accountability of the Internet is thus of the utmost

importance. With the growing penetration of Internet connectivity in terms of geographical size and the number of connected users and the fact that users go from occasionally connected to always connected, the Internet infrastructure is also growing in geographical distribution, number of network elements and heterogeneity of physical connectivity (optical fiber, twisted pair, co-axial cable, wireless, etc.). The Internet infrastructure growth makes its manageability and configurability increasingly complex. It is thus expected that the operating cost of the Internet technology will start to increase more than proportionally to the number of nodes resulting from (*i*) the additional patches that will have to be developed, deployed and operated, (*ii*) the growth of the infrastructure (in terms of number of autonomous systems, routers, and routes), and (*iii*) the increase in both the number of connected users and their activity (in terms of time, location and traffic, and the heterogeneity in application needs). This results in increasing complexity and decreasing maintainability of user's satisfaction while keeping an openly accessible, neutral, and generic Internet infrastructure.

In this context, the main objective of the ECODE research project is *i*) to introduce a new architectural component that allows to transpose the high-level objectives and constraints of what the Internet is supposed to deliver to the end-user into lower-level objectives and constraints than can be enforced by means of the newly proposed component, and given that, *ii*) to determine how the proposed component can ensure the Internet is delivering what it is supposed to deliver and performs as expected to satisfy the end-users. This architectural component will be realized by means a loosely coupled cognitive system. At the node level, the introduction of the cognitive engine implementing machine learning techniques (and thus referred to as machine learning engine) is expected to improve and extend the overall Internet controllability capabilities as well as reducing their resulting cost. Indeed, this step evolution is expected to

- Limit the cost of the Internet infrastructure growth;
- Limit the cost of its operation (compared to the approach that would consist in continuously patching existing routing equipment);
- Provide adequate solutions to the existing and foreseeable upcoming Internet challenges.

The overall objective is thus to ensure the durability of the Internet (and so preserve its design principles underlying its current architecture) by removing complexity from existing routing system components. In other words, adding a machine learning engine to the Internet routing equipment would in turn add functionality to the global infrastructure while maintaining strict bound on complexity. At the same time, this additional component would decrease significantly the equipment and the operational cost as well as the complexity of the Internet compared to an infrastructure that must provide for the same functionality with continuously patched routing equipment.

The remainder of this deliverable is organized as follows: Section 4 provides a broad view of machine learning techniques; Section 5 discusses details of the ECODE architecture on a system and network point of view; Section 6 deepens the security and monitoring use cases while Section 7 details the routing use cases, Section 8 is on the path selection use case, and, finally, Section 9 is about the accountability use case. Each use case is presented in a networking point of view. In addition, we explain how the use case will interact with the machine learning module. We further explain how the use case can be integrated in the ECODE architecture depicted in Section 5; Section 10 provides a grid for evaluating how the various use cases interact with the ECODE architecture. Finally, Section 11 concludes this deliverable.

4. Survey on Machine Learning Techniques

Machine Learning can be considered as a subset of Artificial Intelligence, statistics, and applied mathematics. It aims at building computing machines that learn and improve their responses by learning from experience [21]. Machine Learning and Pattern Recognition are intimately related disciplines [22], since Machine Learning objective is to learn to recognize patterns in order that when new samples are presented to the system in the future, the machine is able to either classify this sample, among a series of classes (supervised learning), or to discover inherent patterns into the data in order to elucidate classes or clusters from the data (unsupervised learning).

It is possible to subdivide the Machine Learning techniques, into the four main types of techniques described below [22]:

1. *Template matching*: In the template matching approach, an incoming sample is compared against 'templates' that describe each one of the classes into which the sample is going to be classified. Some algorithms can be applied to compensate for individual sample differences against the template that can arise. This was the earliest type of Machine Learning/Pattern Recognition approach, which had very low power of *generalization*, that is, depending on the problem, the performance classifying unseen samples was low. Also, usually these techniques are not robust to presence of noise. It is still used in simple cases.
2. *Syntactic approach*: The syntactic approach is concerned to finding the minimum atomic components of a particular problem, called *primitives*, and then finding the rules that govern the pattern formation made of these primitives. These rules are called a *grammar*, since they describe how a pattern is made by smaller components, just like words generate a sentence according to some rules. An interesting appeal of this approach is that it allows to gain insight into how these patterns are generated.
3. *Neural Networks*: This learning paradigm is inspired from a simple model of how the brain works. By using many parallel computing units, it is possible to learn complex, non-linear, input-output relationships. It also has been demonstrated that neural networks can approximate any function with arbitrary precision [23]. This is extremely useful, since neural networks can also be used in regression problems.
4. *Statistical Learning*: This is the most popular approach to machine learning. This short survey focuses on Statistical learning. This approach relies on a statistical description of the process being examined. Usually, the most relevant information or *features* of a process are obtained through a transformation process first (pre-processing). Once the features are obtained, probabilistic and statistical models are employed to infer probabilistic models that, based on the features, will classify an incoming sample. This process of learning is shown on Fig. 4.1.

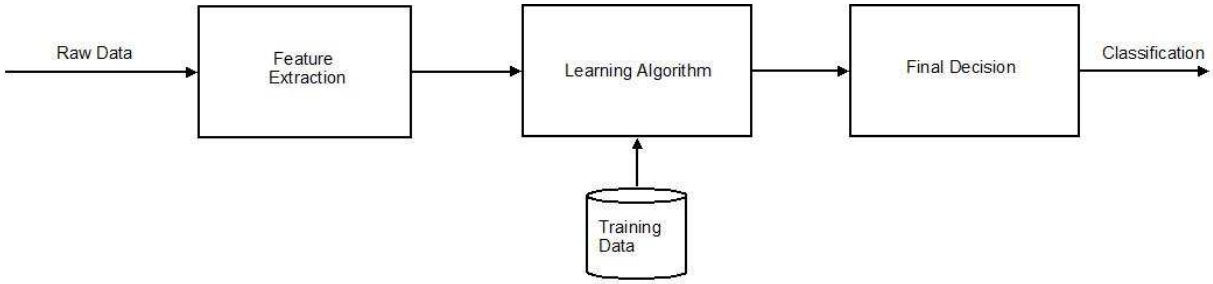

Figure 4.1: Learning process

Statistical learning can be divided into two big types, which are *supervised learning* and *unsupervised learning.* Both have advantages and disadvantages.

4.1. Supervised Learning

The objective of supervised learning or teacher learning is to estimate/predict the output for a novel (never-seen-before) input, after learning on the training data set. This predictive technique consists in creating and predicting value of function that takes as input a training data set that is completely labeled by pre-defined classes (like a teacher) that provides as output a continuous value (i.e., *regression problem*) or, predict a class label of the input object (i.e., *classification problem*) [6, 7, 8]. Typical problems tackled by supervised learning approaches are regression problems and classification problems (predicting a class label of the input). In the network area supervised learning may be used to recognize network mis-configurations, detect intrusions, etc. The main downsides to supervised learning are that the system needs to go trough a training phase requiring a lot of labeled data and it may not work well for predicting values from input data following a different distribution than the training data. Some examples of supervised learning are *naïve Bayes classifier*, *Linear Discriminant*s and *Support Vector Machines* (SVM).

A naïve Bayes classifier assumes that the *d* components of the feature vector are conditionally independent given the class labels and applies Bayes' Rule together with the conditional probability rule, to find the probability that a particular sample belongs to a particular class. Assuming there are *k* classes, the probability that a particular sample belongs to a class *ki* is found by using the Bayes' Rule and applying the conditional probability rule to express this probability as the product of independent conditional probabilities of the features. All the parameters that are required to calculate the probability are obtained from the training set, using the relative frequencies in the set, and finding the maximum likelihood, and sometimes, numerical algorithms such as gradient descent, to maximize the likelihood function. Despite its apparent simplicity, a successful example of using the Bayes Classifier is junk email spam filtering [24].

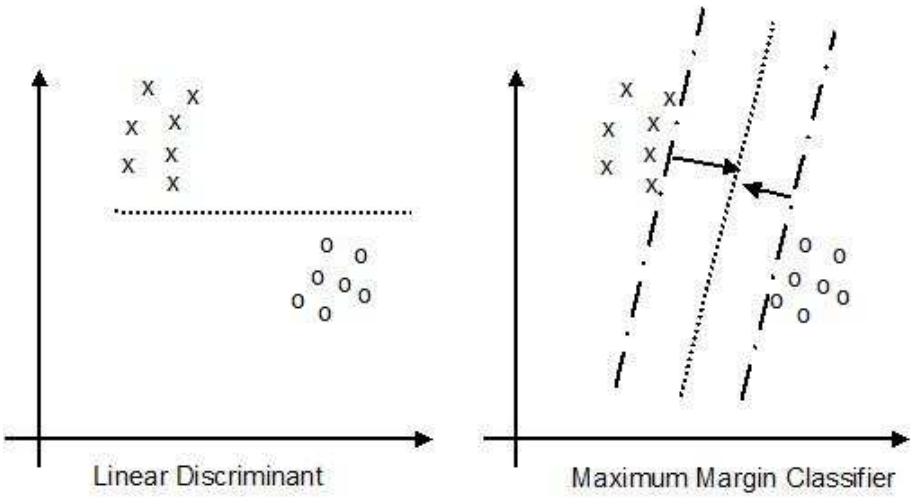

Figure 4.2: Classification

Linear Discriminants attempt to find the hyper planes that best separate different classes, as represented by their features in *d*-dimensional space. This is possible, if the features they represent can be *linearly separated* into classes. There are various methods to find the

parameters of the hyper planes, the most popular being the perceptron algorithm [25]. Successful applications of Linear Discriminants can be found in face recognition.

Support Vector Machines (SVMs) are considered to offer state of the art classification results. SVMs belong to a more general class of linear classifiers, called *maximal-margin* classifiers. SVMs try to find the plane that maximizes the distance between the closest sample in each one of the classes and the separating hyper plane, thus, maximizing the margin. It is also possible, by application of the kernel trick [26], to transform the features via a nonlinear transformation and perform the classification linearly in the transformed space. This transformation, allows for better interclass separability in many cases. A simple example can be seen on Fig. 4.2. Successful examples of application of SVM to real world problems include Facial Expression classification [27], and optical character recognition (OCR) [28].

4.2 Unsupervised Learning

Unsupervised learning, being at the other end of the scale, allows for learning useful structure without labeled training data/classes, optimization criterion, feedback signal, or any other information beyond the raw data and grouping principles. This descriptive technique is typically used for applications requiring clustering, hierarchical clustering (taxonomy creation), novelty detection ("meaningful outliers") or trend detection (extrapolation). Anomaly detection is an example of a network application where unsupervised learning may be used. The objective of such approaches is to develop methods that can perform simultaneous unsupervised learning at all levels of abstraction to hide unimportant variation while exposing important variation

One example of unsupervised learning is k-means clustering. K-means clustering attempts to find an optimal (in some sense) partition of the input space into *k* clusters. To find the clusters, a set of *k*-centers are defined and the distances of the samples to this center are calculated. The most usual distance measure is Euclidean distance. Then, different *k* centroids or centers of the clusters are iteratively tried until the distances between centroids and the samples reach a minimum [29]. Despite its simplicity, this algorithm has worked quite well in many situations. It has been successfully applied to speech recognition, genome data analysis and ecosystem data analysis among others. In Fig. 4.3, there is an idealized illustration of a cluster of different data classes. The actual shape of the spheres or hyper spheres depends on the distance measure used, either if its Euclidean or Mahalanobis distance, among many other possible distances.

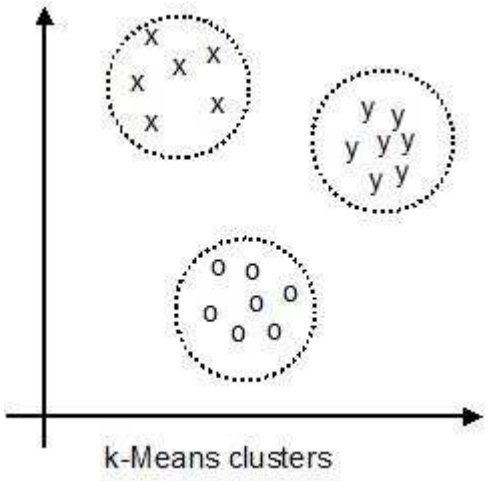

Figure 4.3: Cluster of different classes

There is a clear trade-off beyond these two techniques (to be seen as extremities of the machine learning technique spectrum). Indeed, unlabeled data are usually plentiful whereas labeling of a large set of examples is time-consuming (condition often verified in networking environments). Hence, labeled data are expensive. The trade-off consists in using semi-supervised learning (also called partially-supervised techniques) using both labeled and unlabeled data together [9, 10]. The main idea behind this approach is to use only a small number of labeled examples for bootstrapping and make use of a large number of unlabeled examples for learning.

One objective of the project is to investigate via experiments the applicability of semi-supervised learning approaches to network problems in the area of traffic measurement (*i*) and the area of anomaly detection (network security) at the level of individual packets, traffic flows, protocol messages and their processing (*ii*).

1. Predicting the current state of a path based from past measurements (RTT, bandwidth, etc.) is, from a machine learning point of view, a particular regression problem for time series. Learning methods can be used here to induce a model from past observations and to predict the future state(s). The challenge will be to make these predictions robust from a limited amount of previously analyzed data in a semi-supervised setting, a relatively new but growing framework in machine learning [9].
2. Anomaly detection systems benefit from training on existing data (teacher learning) but also require some level of autonomy in learning, as the network will be subject to new types of attacks/anomalies. The objective is to develop new methods that perform simultaneous semi-supervised learning at different levels of abstraction (e.g., packet vs. flow) to hide unimportant variation while exposing important variation. It is believed that by performing anomaly detection at different levels the global anomaly detection can be made more robust by exploiting regularities at the multiple levels.

4.3. On-Line Learning

In off-line learning data is collected, possibly (manually) labeled, and then provided to the learning algorithm in a batch process. In off-line learning algorithms it is assumed that the time to search through the space of knowledge structures is not strongly limited. This allows for the learning system to learn the structure that minimizes the probability of faulty system outputs. The off-line learning mechanisms may be less applicable in dynamic networking environments where possibly a lot of data is generated at every time instance and where resources and thus also processing capabilities are fairly limited. There is thus a clear need to move to on-line learning mechanisms that are able to process real-time data streams in a real-time fashion (no built-up of unprocessed data).

As opposed to the problem mentioned in the paragraph above with respect to huge amounts of available data that require processing, there is another problem that will need to be addressed. This problem arises from the fact that training data about special events (e.g., anomaly detection) is fairly sparse because their occurrence is not frequent. The system will thus not be able to identify certain traffic patterns the first time they are encountered. A mistake-bounded criterion (how many mistakes does the system make before it learns to recognize the pattern?) will have to be used to evaluate different learning techniques.

Research on on-line learning techniques will be of particular interest for applications that require adapting the model in a possibly changing environment (in particular, for path availability estimation and path performance monitoring). Simple regression models are often estimated via (regularized) least-square methods. Stochastic gradient descent techniques are

available for estimating their parameters on-line. On-line learning is still a difficult task however for more advanced regression methods. Most kernel methods for instance represent the data through a Gram matrix made of pair-wise similarities between individual data points (simple dot products in the linear case). Some methods explicitly deal with well-selected subsets of the data (e.g., [11]) but their objective is to reduce the computational complexity of the estimation algorithm not necessarily to avoid the explicit storage of past data.

4.4. Distributed Learning

While the majority of the machine learning problems is being addressed in a centralized fashion, this no longer holds for massively distributed environments such as networks where the shear amount of data being produces by sensors/detectors/etc. would not lead to scalable solutions. Hence, an additional point to be studied concerns distributed learning: the distribution between several learners, the so-called machine learning engines in the ECODE project. Distribution may concern the data to be analyzed and/or predicted.

Again a networking example can be found in anomaly detection systems where cooperation between nodes is mandatory to be able to detect certain types of attacks and where local anomaly detectors may benefit from relevant traffic models learned elsewhere.

Distribution may concern the data to be analyzed and/or predicted, but also and if needed, the induced regression models or some knowledge deduced from their internal representation. Whereas distribution at the data level does not prevent from using a standard learning algorithm (and can only helps this algorithm to build a model from more relevant information), the availability of general distributed learning algorithms is still an open research issue.

5. ECODE Architecture

5.1. Architectural Outline

The most notable use of learning paradigm applied to networking is the unified *Knowledge Plane* (KP) as proposed by Clark et al. [5]. The driving idea of the KP is to augment the network control system with a higher-level structure that addresses issues of ``knowing what is going on'' in the network. Its main goal is to build a new generation of network that can *i*) drive its own deployment and configuration *ii*) diagnose its own problems *iii*) make defensible decisions about how to resolve them. The KP, by sitting on top of the current control system, is also intended to break the boundaries between the control and management system of the Internet (current Internet management system is driven by weak coupling between its different administrative units).

Architecturally, the KP, that embodies cognitive tools and learning, is a separate structure that creates, reconciles and maintains the many aspects of a high-level view, and then provides services and advices as needed to other network elements. The core foundation of the KP is its ability to integrate behavioral models and reasoning processes into a networked environment. Compared to the KP approach, we believe that a cognitive routing system should be driven by three fundamental principles. First, a cognitive routing system should be structured *modularly* instead of relying on a unified approach for functional but also practical development and deployment reasons. Second, the system should be *segmented* so that it relies on a relative view of the network environment (in particular, from the routing perspective), instead of requiring a global network view to operate, resulting in scaling and deployment issues. Finally, it should be built taking into account the inherent distributed properties and capabilities of the routing system (i.e., *sizeability*) instead of being constructed as a uniform and ubiquitous two-dimensional structure that does not account for the specialization of the routing functionality (e.g., intra-domain vs. inter-domain).

We describe of the ECODE architecture at two-levels: the *System Architecture* level (see Sec. 9.2.) and the *Network Architecture* level (see Sec. 9.3.).

5.2. System Architecture

5.2.1. Overview

The proposed methodology relies on cross-fertilization between the networking and machine based techniques to form a cognitive routing system answering nowadays operational and tomorrow's Internet challenges. Indeed, these networking challenges are similar to the conditions traditionally encountered in classical machine learning problems.

First, the events cannot be well characterized even when examples of such an event are available (the *Nature*). Second, the correlations and trends between events are hidden within large amounts of data that are associated to these events (the *Relationship*). Third, the conditions (*Environment*) are changing over time. This is particularly the case for the routing environment but also the variability of traffic demands, expectations and behaviors. Fourth, the amount of available data is too large for handling by human intervention (the *Quantity*). Finally, new events are constantly detected/discovered (the *Evolution*).

From this analogy, the main concept we develop is to extend existing IP networking equipment, i.e. routers, with a machine learning engine (as shown in Fig. 5.1) and refer to

them as cognitive routers. Based on advances in machine learning techniques including semi-supervision, on-line, and distribution, the *Machine Learning Engine* (MLE) derives a number of observations from the data collected from the routing and/or forwarding engines and possibly from other MLEs. By processing these observations, the MLE learns rules resulting in local decisions (directed towards the local forwarding and routing engines). These decisions can also be distributed to other MLEs in case dissemination of the decision(s) beyond the local router is required. The distribution of the processing as well as the learned rules may depend on the peering relationship between cognitive routers. In particular, the boundary defined by autonomous systems may be a limiting factor to the distribution of such information. Hence, cognitive routers may be operated in one domain independently of the deployment level in adjacent autonomous systems without modifying their peering relationship.

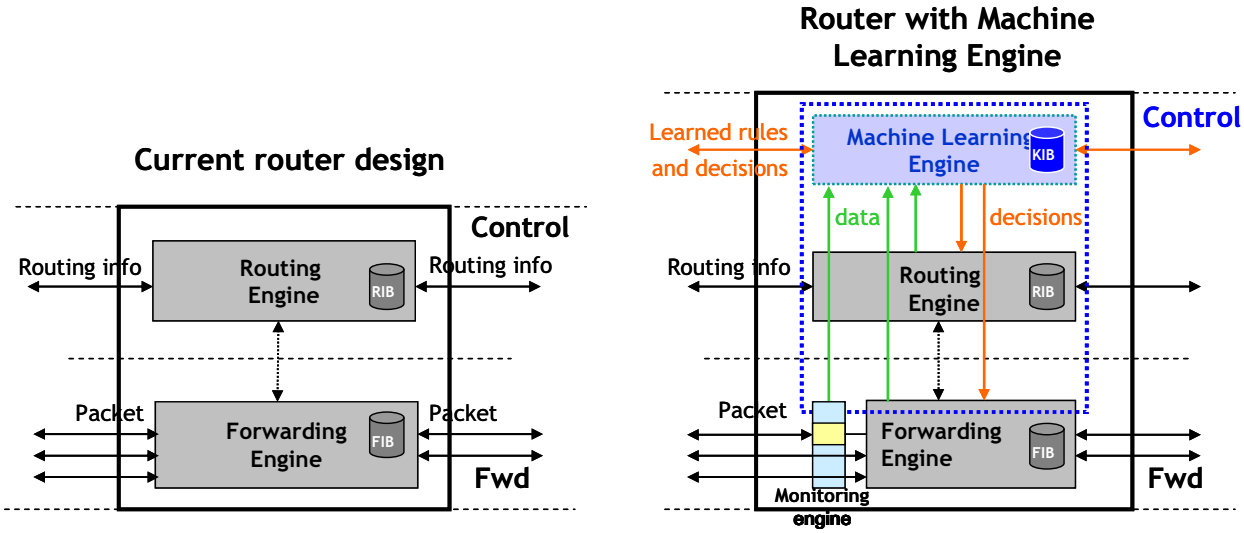

Figure 5.1: ECODE System Architecture Overview

5.2.2. Data structures

As depicted in the left-hand side of Fig.5.1, a standard router comprises a *forwarding engine* (as part of its forwarding plane) and a *routing engine* (as part of its control plane). The forwarding engine includes a packet processor and a *Forwarding Information Base* (FIB). The routing engine includes a routing information processor and *Routing Information Base* (RIB). As depicted in Fig. 5.1, the RIB stores the routes and (in some cases) the metrics associated with those routes to particular network destination prefixes. This information contains the topology of the network immediately around the router. The FIB is used to find the proper interface to which the input interface should send a packet to be transmitted by the router. The FIB is constructed based on the RIB and according to policies defined by the operator. It is optimized for fast lookup of destination addresses.

The ECODE architecture introduces in addition to the forwarding, and routing engines (see right-hand side of Fig.5.1):

- The *Machine Learning Engine* (MLE), part of the control plane, aims at processing by means of learning methods, the input from the network (obtained via forwarding and control components) to subsequently decide on forwarding and routing execution. The MLE provides the means to propagate the corresponding decisions to the routing and forwarding engines.

The MLE comprises four different functional components: the *Translator*, the *Representation*, the *Processing*, and the *Distribution* (see Section 5.2.4).

As part of the MLE, the following data structures are introduced:

- The *Knowledge Information Base* (KIB): stores learned models and (past) decisions. Both constitute the so-called prior knowledge.
- The *Observation Information Base* (OIB) stores bounded sequences of observations that can be accessed by means of the *Register* (R_L) or loaded (on-demand) by the *Processor*.
- The *Learning Methods* (LM) base: stores the learning algorithms

- The *Monitoring Engine* (ME): part of the forwarding plane, it comprises a set of monitoring points that can be either passive of active. When passive, the monitoring point aims at capturing packets (for passive measures), and when active, it can additionally injects probes (for active measures). Once captured, packet data may be classified per source-destination pair, per destination prefix, per traffic class, etc. and metered to measure the bandwidth, delay, packet loss, etc. of classified traffic streams. Alternatively, captured data may be selected (filtered or sampled) before being classified. The ME parameters such as the sampling parameters (e.g. sampling rate), and filtering parameters are controlled by the MLE by means of the ME controller. The functional description of the ME is further detailed in Section 5.3. Resulting data is buffered the monitoring data register part of the reporting module (see also Fig.5.2).

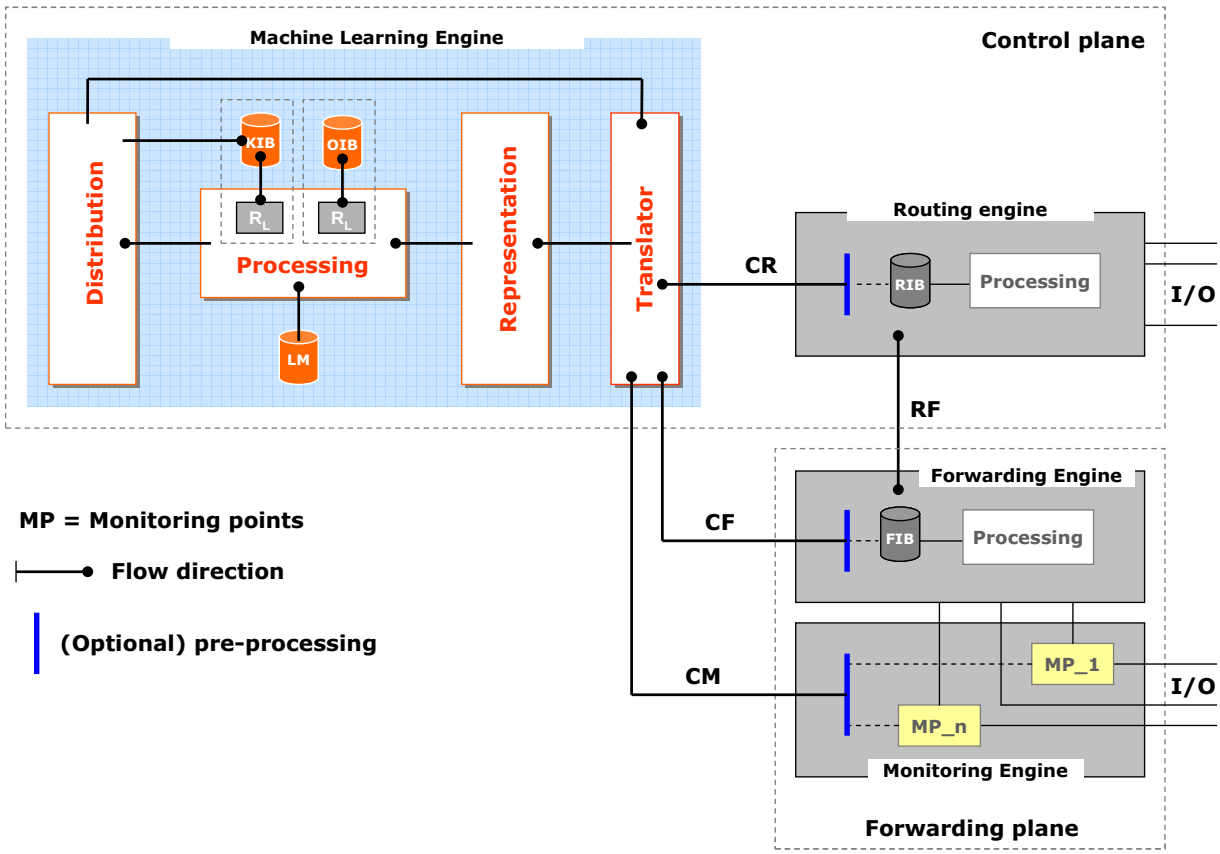

Figure 5.2: Data Structures and Interfaces between Forwarding/Routing/Monitoring and Machine Learning Engine

5.2.3. Interfaces

The interaction between the various engines is performed through dedicated interfaces. We consider four distinct interfaces (see Fig.5.2):

1. RF (for Routing – Forwarding): through this interface, the routing and forwarding engines can communicate with each other and exchange information if requires.
2. CR (for Cognitive – Routing): through this interface, the MLE may retrieve data from the routing engine and communicate to the routing engine the decision it takes.
3. CF (for Cognitive – Forwarding): similarly to the CR, the MLE may retrieve data from the forwarding engine and communicate to the forwarding engine the decision it takes.
4. CM (for Cognitive - Monitoring): through this interface, the MLE may retrieve path performance information from the ME.

Monitoring, routing, and/or forwarding engines provide raw and/or pre-processed data to the Machine Learning Engine (MLE) through the CM, CR, and CF interface, respectively. The reason for optional pre-processing is to prevent potential MLE processing overloading. Based on this data and machine learning methods as well as prior knowledge such as learned rules and/or decisions, the MLE takes decisions and sends them back to the routing, forwarding, and monitoring engines. Note that the learned methods are stored in a particular structure called the *Learning Methods (*LM) base. The so-called prior knowledge and learned models are stored in the *Knowledge Information Base* (KIB) used, in the ECODE architecture, to store prior knowledge such as learned models or decisions. The *Observation Information Base* (OIB) stores bounded sequences of observations that can be accessed by means of the *Register* (R_L) or loaded (on-demand) by the Processing.

Fig. 5.2 provides a representation of the interface between forwarding and routing engines and the Translator function of the MLE. The CM interface is here depicted for a single line card comprising a set of N interfaces/ports, n of them ($n \leq N$) being equipped with a *Monitoring Point* (MP). Thus, a node may comprise multiple instances of the CM interface.

5.2.4. Machine Learning Engine Components

Fig. 5.2 provides a view of the elements composing the Machine Learning Engine (MLE). As illustrated, the MLE comprises four different functional (sub-)components: the *Translator*, the *Representation*, the *Processing*, and the *Distribution*.

- The *Translator* comprises a syntax function that converts the data received from the Monitoring, Routing, and/or Forwarding engines into uniformly formatted data.

- The *Representation* takes the formatted data (received from the Translator) and transforms it into various tagged observations describing states, events, or conditions. A "tag" can for instance include the "originating plane", the "type of information", the "time stamp/interval", etc. In other words, the Representation function provides inputs to the machine learning algorithms. A "tag" is assigned to these observations to selectively call adequate their processing by the processor. Indeed, from the training data set, the processor selects inductively a learning algorithm to derive the target function. In other words, the representation function acts as pre-processor that provides the actual input to the machine learning processing. The reason is to prevent overloading the processor with the large amount data that is received from the routing,

forwarding, and monitoring engines (even if the latter is making use of sampling, filtering, etc).

- The *Processor* includes the Learner and the Performer (functions) and the associated registers. The Learner makes use of observations for training purposes and produces a learned hypothesis h defined as the approximation of the target function representing the prediction rule to be learned. The Performer uses unseen observations and determines if the hypothesis h is a good learned approximation of a target function and the complexity c(h). Taking into account the trade-off between underfit and overfit, once the learned hypothesis is sufficiently accurate over the test data set (test error), the learned rules can then be used to produce decisions. Decisions are taken by applying the learned rules to new incoming observations by combining the obtained decision with previously reached decisions as well as objectives (functional and performance) and constraints (both technical and non-technical).

- Decisions and learned rules are then provided to the *Distribution* component that aims at disseminating them. This dissemination might be local (i.e., inside the router) or external (i.e., between various routers). If the dissemination is local, the decisions are sent to the Translator so that they are correctly formatted for the routing, forwarding, or monitoring engines. The Translator is also in charge of sending this formatted output to the routing, forwarding, monitoring engines through the dedicated interface (`CR`, `CF`, or `CM`). Learned rules are locally stored in the `KIB`. On the other hand, if the dissemination is external, the decisions and learned rules are sent to others Machine Learning Engines through the CC interface, as detailed in Section 5.4.2, and illustrated in Fig.5.6.

5.3. Monitoring Engine

The *Monitoring Engine* (`ME`) comprises set of monitoring points and an (optional) monitoring data pre-processor. Monitoring points (`MP`) can be either i) passive (passive `MP`) or ii) active (active `MP`).

The `ME` could have been complemented with an (external) common monitoring information base. The advantage of the proposed architecture is to benefit from a single interface with the `MLE`. On the other hand, adding a monitoring information base requires an additional data structure and associated processing as well as increases the delay before reaching the Translator. Thus, this option does not seem viable in the context of the proposed architecture. A good compromise consists in distributing pre-processing (per set of `MP`). In this case, delays would be reduced (no intermediate storage outside of the `MLE`) and load on the translator would still be lowered (compared to the situation where all monitoring data reach the translator). The reporting block (depicted in Fig. 5.3) would comprise the monitoring data buffer.

5.3.1. Passive Monitoring Points

Passive monitoring points (passive `MP`) provide the following functions (see Fig.5.3): capturing, selection, classification, metering, and reporting. The selection function may either perform sampling or filtering of the captured packets. The capturing and the selection functions are driven by a controller that is in charge of receiving the decisions from the `MLE` and transpose them into configurations parameters, e.g., sampling rate and filtering patterns.

5.3.2. Active Monitoring Points

Active monitoring points (active MP) are designed as Passive monitoring points with the addition of a prober. The prober injects probes on the "wire" (insertion of "monitoring" packets). The controller commands the setting of the prober parameters.

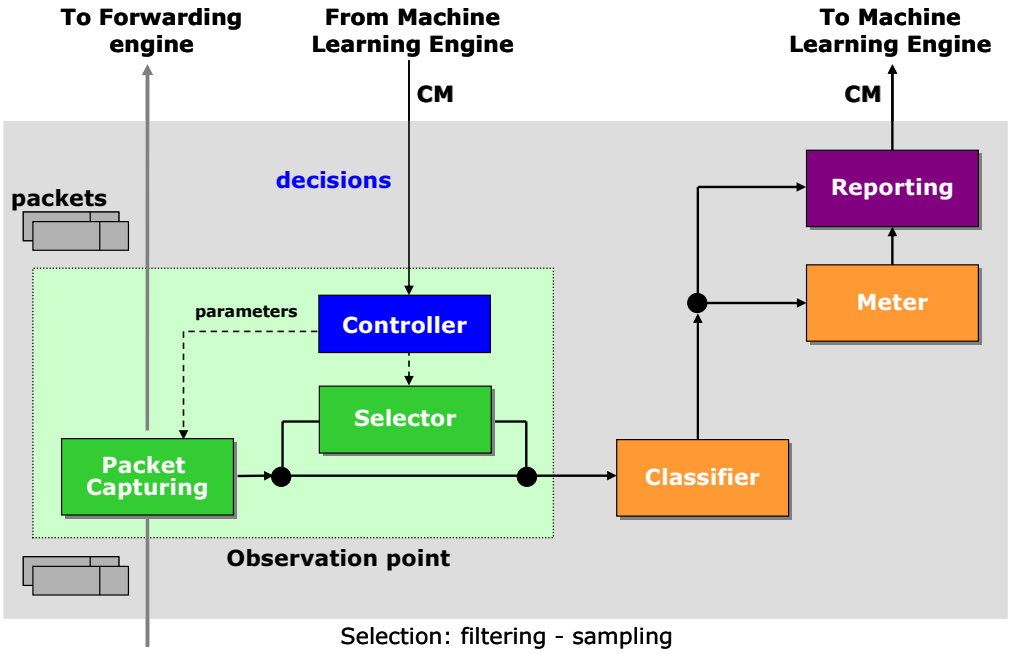

Figure 5.3: Monitoring Point (MP) - Functional architecture

5.4. Network Architecture

Fig. 5.4 outlines a generic architecture of the network so as currently observed in the Internet topology: *Tier structure*.

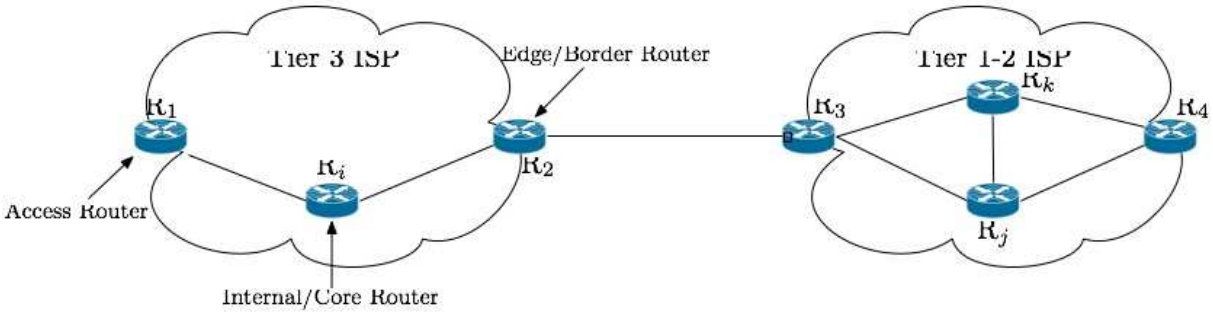

Figure 5.4: General architecture of the network

The network is divided into *Tier* ISP:

- *Tier 1* ISPs refer to backbone providers. There are a dozen of large international and large national ISPs interconnected by multiple private peering points (i.e., shared cost). The Tier 1 ISPs provide transit service (i.e., no “upstream” provider). Examples of Tier 1 ISPs are AT&T, Verizon, Sprint, or Level 3.
- *Tier 2* ISPs are regional or national ISPs. They are typically customer of Tier 1 ISPs (at least one, but often two) and provider of Tier 3 ISPs. They have shared cost links

with other Tier 2 ISPs. Examples of Tiers 2 ISPs are France Telecom, British Telecom, or Belgacom.

- *Tier 3* ISPs refers to stub ASes. They are smaller ISPs, corporate networks, or even content providers. They are only customer of Tier 1 and Tier 2 ISPs. They possibly have shared cost links with other Tier 3 ISPs

5.4.1. Location

Fig. 5.4 also illustrates the different types of routers: *Access*, *Edge* (or *Border*) and *Internal* (or *Core*) routers. Access and Edge routers are located at the border of an ISP (R_1, R_2, R_3, and R_4 on Fig. 5.4), while Internal routers are located within the ISP (R_i, R_j, and R_k on Fig. 5.4). Only Access and Edge routers send/receive information from other ISPs. As already mentioned in Section 5.2., any standard router comprises a forwarding and a routing engine. Any of the three routers types being access, edge or internal routers can be equipped with a `MLE` as discussed in Section 5.2.

5.4.2. Information Exchange and Distribution

This sub-section describes how cognitive routers might cooperate with each other in order to exchange information. Some use cases might take benefit and others simply require exchange between cognitive routers. Distributed traffic anomaly detection is by definition relying on distributed input exchange between detecting routers within the same routing domain whereas profile-based accountability can be executed on a single access router or take benefit of collaboration among a set of access routers belonging to the same domain.

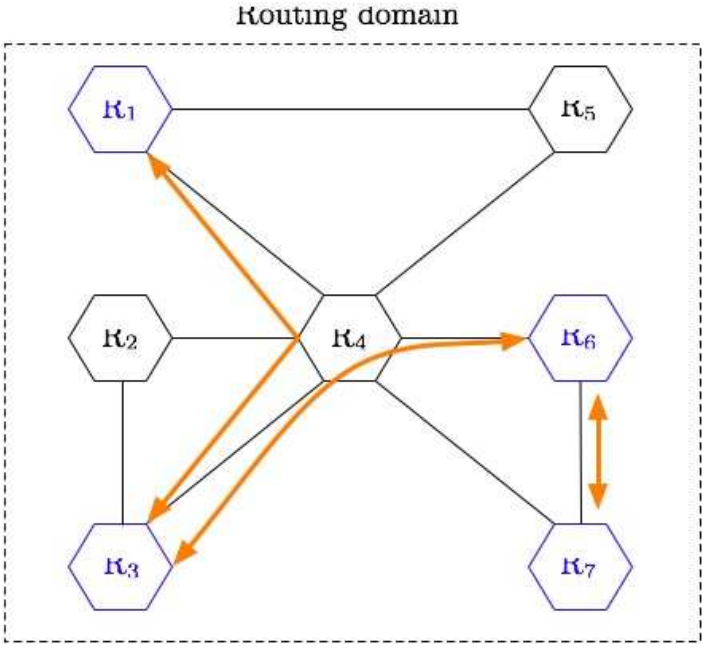

Figure 5.5: Communication channels

Fig. 5.5 represents the different communication channels that can be realized between cognitive routers (orange lines between blue hexagons). The topology of the communication channels can be either congruent (e.g., between R_6 and R_7) or non-congruent (between R_1 and R_3, or between R_3 and R_6) with the routing topology as depicted by the gray solid lines. Note also that the logical topology between cognitive routers does not need to be a full mesh as represented in Fig. 5.5. Also, communication channels between cognitive routers do not need

to be permanent, i.e., they may be setup on demand depending on the distribution of the functionality.

As depicted in Fig. 5.6, exchanges between routers equipped with a MLE are performed by means of a dedicated CC interface that is functionally subdivided into:

1. CC_r: this sub- interface defined between representation modules of peering cognitive routers is used to exchange input to the machine learning processing.
2. CC_p: this sub-interface between processing modules of peering cognitive routers is used to exchange during machine learning execution is used to exchange during processing (for e.g., co-training purposes).
3. CC_d: this sub-interface between distribution modules of peering cognitive routers is used to exchange learned rules and decisions.

As stated before, the distribution of the processing as well as the learned rules and decisions but also input to the machine learning processing may depend on the peering relationship between cognitive routers. In particular, the boundary defined by AS may be a limiting factor to the distribution of such information that is strongly dependent on the peering or client-server relationship between ASes.

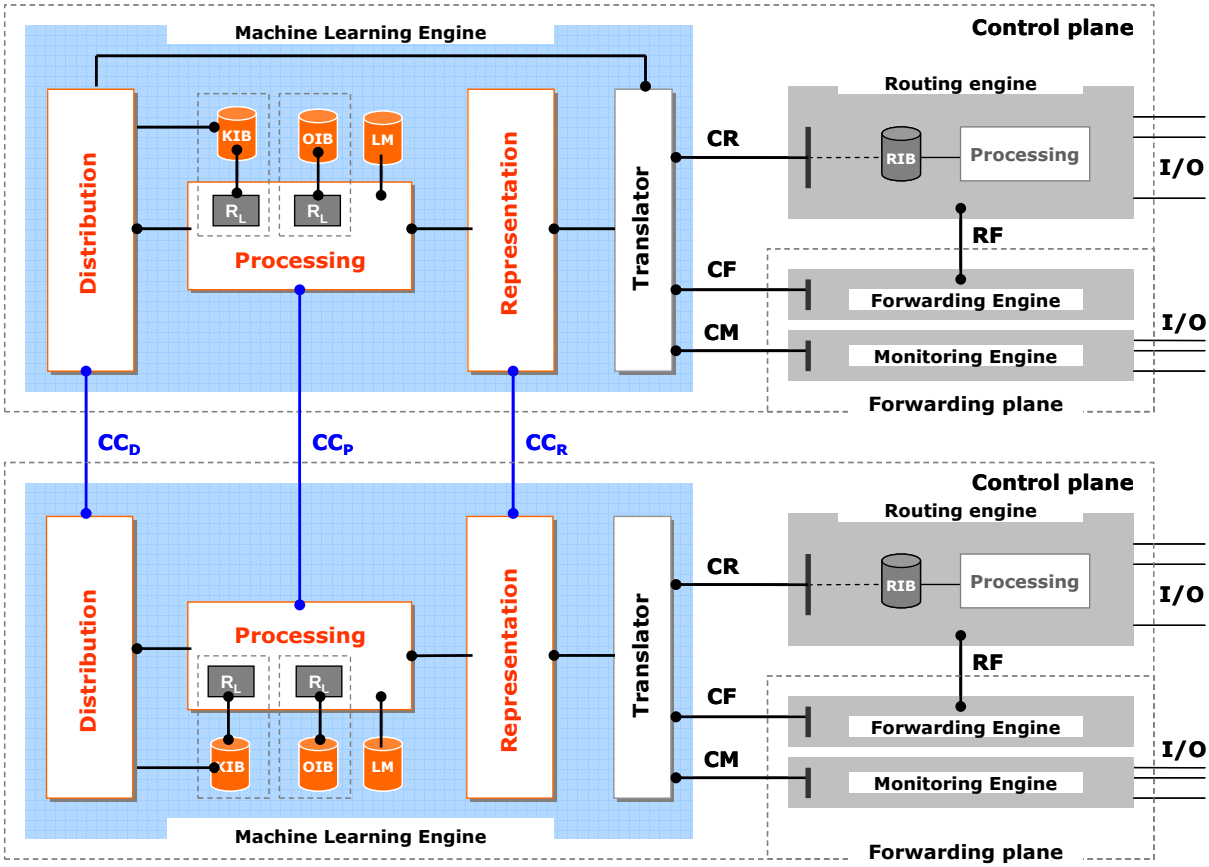

Figure 5.6: Exchange between routers equipped with a Machine Learning Engine (MLE)

6. Security and Monitoring

6.1. Adaptive Traffic Sampling

6.1.1. Use Case Description

Our goal is to develop an autonomous system for network monitoring and traffic management. Starting from a measurement task like for example the calculation of the traffic matrix, the estimation of flow sizes and rates, the prediction of flow rate increase/decrease, or the detection of anomalies, the system will configure the sampling rates in network routers so as to optimize the accuracy while limiting the overhead (volume of collected traffic, packet processing and memory access in routers). The system will include modules to sample the network, collect the sampled data, analyze it, find the optimal sampling rates, and configure routers accordingly.

Using the inferred information on traffic statistics, we will design algorithms for flow management (e.g., flow scheduling, flow blocking) inside routers that improve user and network performances.

6.1.1.1. System's Input Information

At first, the system uses no entry data except measurements carried out by routers, but we may envisage the exchange of information among routers to boost the convergence for the best measurement model and optimal router configuration. The measurements to be collected are for now of the type sampled packet traces and Netflow records coupled with routing and network topology information (e.g., routes followed by flows inside the network). The results of sampling, either at the packet level or the flow level, will serve as an input to traffic management.

6.1.1.2. System's Output Information

In a distributed approach, each node must provide the other nodes with traffic measurements and information on its routing tables. The nature of exchanged data depends on the network-wide measurement task to be carried on. In a centralized approach, nodes send their measurements to a collector, where they are analyzed. The collector takes the decisions about which routers have to be involved in the monitoring and the amount of involvement (to be controlled by the sampling rate and the way sampling is done).

For example, if we want to determine the greediest users in a given network, the collector starts from a configuration of monitors embedded in routers (e.g., a DAG card performing measurements or a Netflow tool), gathers sampled data from these monitors, and depending on these collected data, may increase the sampling rate on some paths of the network and decrease it on others. The objective is always to improve accuracy while limiting the overhead.

6.1.1.3. Interactions between the Decision Engine and the Forwarding/Routing Engines

Our aim is to build a standalone system that infers the status of the network and realizes some monitoring application like for example anomaly detection, flow characteristics estimation and flow management. Routing optimization is not the target but rather leverage routing /

forwarding information during the network state learning phase and for the configuration of monitors and flow management controllers.

Note that the decision engine could be either centralized or decentralized, both situations are possible. In the centralized case, a collector must be deployed to collect information from network monitors deployed in routers. The collector uses then this information to infer the status of the network using machine learning techniques and to optimize the monitoring process.

In the distributed case, network nodes exchange information among themselves to reach the optimal router configuration (from monitoring and flow management perspectives) with or without the help of a central unit.

6.1.2. Interaction with Learning Module

6.1.2.1. Machine Learning Technique(s)

The system will sample the network, collect the sampled data, analyze it, find the optimal sampling rates, and configure routers accordingly for both monitoring and flow management purposes.

During this procedure we use unsupervised machine learning techniques to estimate network and traffic state (e.g., flow size and rate estimations). The focus will be on techniques like the maximum likelihood, the Kalman filtering, and the EM (Expectation Maximization) method. The estimated state of the network and of its traffic will then be used to optimize the sampling rates in network routers so as to increase the measurement accuracy while limiting the overhead (e.g., volume of collected traffic, packet processing and memory access inside routers).

Moreover, to take optimal decisions for the purpose of traffic management (e.g., flow scheduling), one needs to combine flow properties' estimation and control. To perform the joint estimation and control online, one can use reinforcement learning techniques such as Markov Decision Processes, Markov Decision Processes with Partial Observations and Gittins index.

Note that it is important for our system to have an understandable model that explains the nature of the monitoring application to be run and the assumptions on the underlying network (topology, routing, etc). The procedure of learning network status and the optimization of both monitoring and traffic management depend strongly on the nature of the application and the model assumptions.

6.1.2.2. The Learning Stage

The learning stage will take place where the decision function needs to be adapted dynamically to change in the system. We want our system to adapt to any new monitoring application and to any change in network conditions. The system operates then in iterations and during each iteration, there is a collection of sampled information from the network (raw packet traces or NetFlow records), the application of machine learning techniques to estimate network and traffic state, and the optimization of the monitoring configuration. Jointly, we keep evaluating the accuracy of the estimation, and when it is judged acceptable, network management decisions could be safely taken.

6.1.2.3. Inputs Variables to the Machine Learning Module

Our problem is unsupervised. However, we use as input parameters constraints on the volume of sampled data and on the capacity of routers to perform monitoring (e.g., upper limit on the sampling rate).

Then and during the execution of the machine learning procedure, we use additional inputs like the previous estimation of network and traffic states and the corresponding monitoring and control configuration. These additional inputs will serve to find a better configuration of monitors and controllers (e.g., better sampling rate, better handling of delay and jitter). The new configuration will be used to carry out new measurements, which will be used to find a better estimation of network traffic state, and so on.

Note also that the inputs of the machine learning module are time varying. Indeed, our system uses sampled information on network traffic and network topology and routing to optimally tune routers. This information is variable by nature due to network and traffic dynamics, and so the system will adapt by continuously collecting data and changing the monitoring and flow management configuration. Note that the change in the configuration can come from modifying the monitoring application to realize, the targeted accuracy, the system parameters and capacity, or a change in the routing.

6.1.2.4. Outputs of the Prediction Stage

The machine learning module in our system is intended to provide an estimation of network and traffic status. This estimation is afterwards used to find a better configuration of monitors and controllers that reduces measurement errors and improve the management of flows inside routers. For instance, our system can optimize the monitoring to carry out the following measurements:

- The number of packets sent by a given AS (autonomous system).
- The greediest users.
- The number of packets per flow.
- Detection of anomalies.

6.1.2.5. Learning Phase Speed

The duration of the learning phase is a compromise between adaptation to network dynamics, overhead and accuracy. The longer we collect sampled data to learn the more the accuracy and the better the decisions we take, but this requires that the system does not change or at least that our model for the system tracks well its variability. One can boost the learning by collecting more samples; however this will incur more overhead. These tradeoffs are to be carefully studied and analyzed. They strongly depend on the monitoring and management application and the characteristics of the sampled population.

6.1.2.6. Description of Training Samples

The process of learning is unsupervised. So, at first the system uses no entry data (no training phase). The system proceeds in iterations to converge and at the same time to react to any change in network conditions. At each iteration, we sample the network and we gather sampled data from monitors deployed in network nodes. Depending on these collected data we estimate network status, and we configure monitors so as to optimize accuracy and limit resources utilization. Consider the example of finding the greediest users. As a first step, the system uses the same sampling rate in all monitors, and then the information is collected and

analyzed to determine estimates on the traffic of the greediest users. Suppose that these users take the edges [R_1R_2] and [R_3R_6], as illustrated on Fig. 6.1.

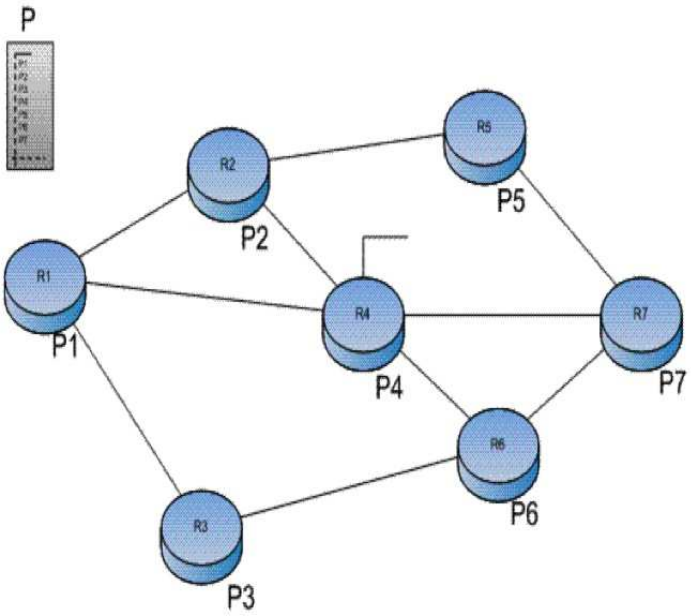

Figure 6.1: Testbed

In a second step, the system increases the sampling rate in routers R_1 and R_3 to increase the accuracy of the flow size estimation of the greediest users while reducing the sampling rate in the other monitors to reduce resource utilization. The optimization should take into account the error introduced by the sampling and the routes taken by flows inside the network (there might be the possibility to monitor a flow at different routers).

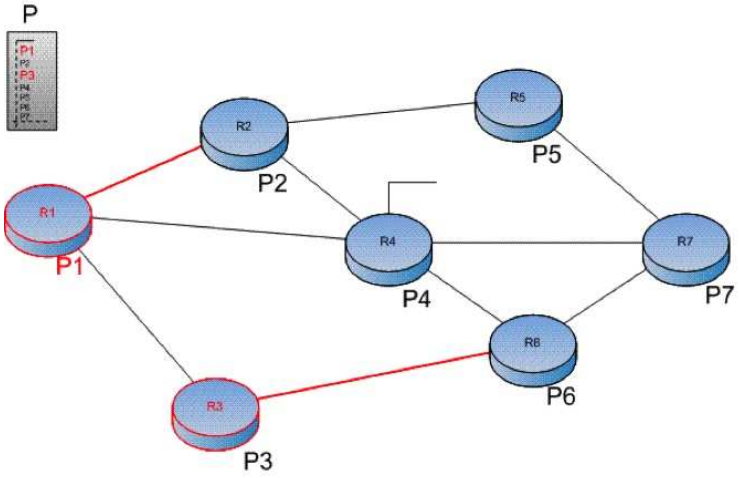

Figure 6.2: Testbed

Note that one can choose to reduce the sampling rate at the other monitors instead of putting them OFF to keep getting an idea on the traffic in the other parts of the network. This way the system can detect any changes in network status such as the emergence of a new greedy user.

6.2. Global/Active Monitoring

6.2.1. Use Case Description

Having measurements or an accurate estimation of performances on network links or on end-to-end paths is of high interest for many functions in networking as admission control, load balancing, (QoS-) routing, congestion control, etc. This research area has been investigated for more than a decade and lead to two different approaches: active and passive monitoring techniques.

6.2.1.1. Passive and active measurements capabilities and issues

Passive monitoring tools are certainly the most appropriate tools for this purpose because of their higher accuracy (active measurement tools most of the time try to estimate performance parameter values based on effectively measured ones). But they are not always available when users need such information. Even carriers or ISPs who manage their own domain or autonomous system and can access any information they need about their own network state, can miss similar information from other carriers or ISP networks they are connected to. As a consequence, tools for estimating performance parameters such as delay, available bandwidth or loss rate on an end-to-end path, are usually based on active measurement techniques, which are said to be user oriented, as opposed to passive measurements, which are carrier or ISP oriented. Active measurement tools can provide a solution for having an easy access to such network feature estimations, and this can be used for any network structure and technology. Many tools for estimating available bandwidth have appeared in the recent years such as Abing, Spruce, Pathload, IGI-PTR, Pathchirp, etc. Ping or Traceroute are quite famous and popular tools for estimating end-to-end delay and loss rate [57, 58, 59]. Tools based on active measurements of the available bandwidth, for example, only allow estimates of this parameter, while passive monitoring tools can measure it in a very accurate way. Despite this limitation, and because of their wider availability, active monitoring techniques for measuring end-to-end path performances are of high interest either for measuring some delay related parameters, either for doing measurements on an end to end basis, even on several ASes.

It is however important to remember that active measurements consist in generating probe traffic in the network, and then observing the impact of network components and protocols on traffic: loss rate, delays, RTT, etc. Therefore, as active measurement tools generate traffic in the network, one of their major drawbacks is related to the disturbance introduced by the probe traffic, which can make the network QoS change, and thus provide erroneous measures. Sometimes, active probing traffic can be interpreted as denial of service attacks, scanning, etc. Probe traffic is then discarded, and its source can be blacklisted. Intrusiveness of probe traffic is a side effect that active measurement tools must take into account. Many studies address this probe traffic intrusiveness issue, by trying to minimize the number of sent packets as well as their impact on network QoS. In addition, if an active measurement tool generates only a few packets, it would certainly provide estimation results in a very short time, which is an important performance parameter in the Internet, whose traffic is very versatile. These two issues, together with the accuracy of estimates for non-directly measurable performance parameters, are the challenging limitations to overcome.

6.2.1.2. Global Monitoring System Components

Global monitoring aims at providing an accurate view on the network performance levels on all links. It relies on both passive and active monitoring tools, but also a reporting mechanism

for broadcasting the measurements results, and possibly their analysis to where they are required. The global monitoring system relies on local measurement points (MP). All MPs are defined in order to integrate several traffic capturing or active measurement tools. They also integrate all needed ECODE modules. Last, they integrate reporting capabilities for exchanging traffic traces, measurement time series or analysis results for online distributed decision making. The basic functional components of the monitoring and measurement system are depicted on figure below.

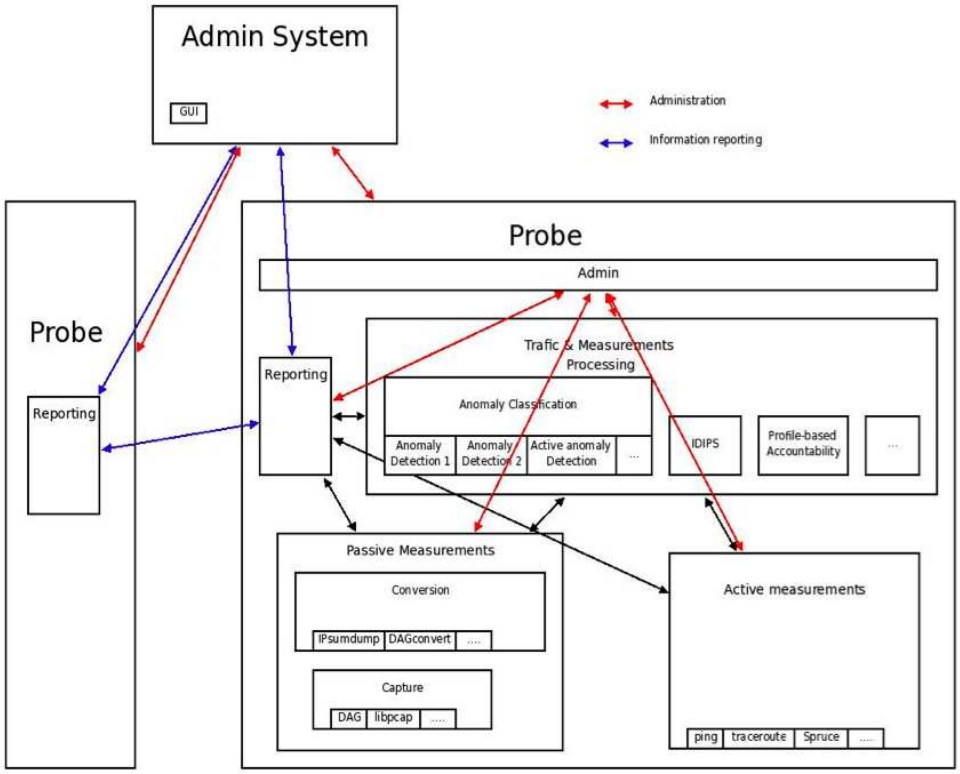

Figure 6.3: Basic functional architecture of the global monitoring system

Note that the functional architecture is quite generic and open. It can integrate many capturing, measurement, and conversion tools, as well as the ECODE modules to be integrating as part of the traffic and measurement processing part.

6.2.1.2.1. Local Passive Monitoring

Local passive monitoring uses a hardware or software tool for capturing traffic. The resulting trace can be a flow or packet trace. In our case, we more specifically consider a packet trace, i.e., provides a trace of each packet (or at least its IP and TCP headers) together with an accurate timestamp. It can also be sampled for limiting the amount of resulting data. The generic format of such packet trace exchange between the passive monitoring and traffic and measurement processing modules is depicted on figure below.

Figure 6.4: Packet Trace Format

6.2.1.2.2. Active Monitoring

Active monitoring uses a hardware or software tool to send packet probes on the network. These packet probes are supposed to get an echo or to be followed on its path as a common flow for all MPs. RTT, end-to-end delays, hop-by-hop delays and PLR are computed on the basis of observation on sent, observed and received packets. UDP, TCP, IP or ICMP packets can be used. Time stamping is a significant issue when we are measuring delays which require considering timestamps provided by different clocks on different machines. Technically, the most accurate solution for having well synchronized clocks is to use GPS antenna and the PPS (Pulse Per Second) signals coming from the reference atomic clocks. As for passive monitoring, the global monitoring system is able to handle several different active monitoring tools which can provide the necessary information for ECODE modules.

6.2.1.2.3. Distributed Decision in Active Monitoring

The active measurement of metrics is impacted by the topology of the AS from which measurements are performed. Indeed, the closer from the target, the more accurate the results. We are also more exposed to the possibility of having asymmetric routes and network changing conditions. Then, depending on the target machine to probe, it is interesting to select the best probe source inside our AS. We can imagine that the selection of such a source probe machine could be a decentralized decision among all the nodes of the AS. For applying the related graph theory algorithm, we don't really think that machine learning procedures would be required as we are supposed to know the topology of the AS we are managing.

6.2.1.2.4. Reporting Protocol

In general, monitoring tools based on passive measurement techniques and running on dedicated routers or any other kind of device in the network are then able to obtain information from a given local point of view. The problem then consists in obtaining a global view of the traffic and the network state. How to advertise the local measurements in the whole network then becomes a question of first importance. Since so far, the number of studies realized in this specific research area is limited. Two generic approaches can be used for obtaining reports from local measurement and monitoring tools:

The polling technique consists in asking periodically local measurement tools about their last measurements. Of course, such requests have to be issued periodically. And this is highly inefficient because many unnecessary queries are issued.

The reactive monitoring is the opposite approach compared to polling. It consists for the local measurement tools to send an advertisement when a change occurs. This can be seen as an event driven approach.

The problem with these two approaches is that they generate excessive extra traffic when used in large networks (and this is especially true for the polling technique). In a large-scale network, a global monitoring system should be lightweight, scalable and fast (near-real-time), i.e., advertising up-to-date information. Nowadays, most proposals rely on the reactive monitoring approach. Looking at the literature, it also appears that most research now addresses the scalability issue. Nevertheless, it does not consider the reactivity level of the reporting system. All improvements of scalability are made at the expense of reactivity. The resulting reporting systems that introduces several seconds of delay between the detection of an event and reporting are not fast enough for solving network issues as congestion control or

intrusion detection. At the opposite, some systems propose a reactive monitoring approach that considers the reactivity level at the expenses of an increase of the advertisement/reporting traffic. To our knowledge, no work has been done to achieve both reactive and scalable monitoring in real time. These are two challenging issues to be solved.

For this purpose, there will be a subscribe/unsubscribe procedure between MPs. If one MP needs the information provided by a second one, then the first will subscribe to the second (similarly as the association between BGP routers) and then will get the information he requests to the MP.

6.2.2. Interaction with Learning Module

No learning stage or module could be of profit to the global monitoring system. However, the global monitoring system, thanks to its capability of providing path performance measurements, is a key component for the routing system which has to select the best path for forwarding packets.

The global monitoring module can be considered in Fig. 5.3, as a part of the Forwarding Engine. In fact, this module will be able to gather raw data from its own hardware but also from other MPs through the distributed reporting system.

6.3. Anomalies Detection

Network traffic anomalies can seriously impact or disrupt the normal operation of networks. It is then vital that network administrators quickly do their identification and mitigation. A specific type, volume anomalies, is responsible for unusual modifications on network traffic volume characteristics (identified by the # of packets, # of bytes or # of new flows) or on its distribution (identified by its application share or its distribution in address or geographical space). These anomalies can be caused by a myriad of events: from physical or technical network problems (e.g., outages, routers misconfiguration), to intentionally malicious behavior (e.g., denial-of-service attacks, worms related traffic), to abrupt changes caused by legitimate traffic (e.g., flash crowds, alpha flows).

Anomalies created by denial-of-service attacks are particularly important because these attacks are extremely common and disruptive to normal traffic characteristics. Even low intensity flooding DoS attacks directly impact the LRD (Long Range Dependence) of network traffic; hence, causing a degradation of the network's QoS [60, 61, 62].

Anomaly detection is achieved by comparing a set of traffic features with a prediction following a normal behavior model. The main issues in anomaly detection in networks are three folds:

- *Issues relative to feature observations*: different anomalies need different features to be observed. Frequently the observation points are distributed over the network. This raises the issue of how to extract features from these distributed measurements and to use them for consistent anomaly detection in the network. An important role of machine intelligence in this context is to decide which metrics should be observed and at what granularity level to enable a reliable anomaly detection and classification. Indeed a simple solution consists to transfer all observation to a central point that will thereafter decide the relevance of each observation to anomaly detection tasks. However this solution is very costly (in term of used bandwidth) and can be

ineffective as sometime local decisions made on more complete observations are more relevant that centralized decision made on aggregated one.

- *Issues relative to model calibration*: The second issue is relative to the choice of structure of the normal behavior model. The model should be expressive enough to describe the complexity of the network traffic behavior and at the same being simple to enable easy calibration. Indeed the main goal of machine learning in the context of anomaly detection is to calibrate this model of normal behavior. Normal behavior model might be defined on a single observation or on a vector of observations. In the latter case, the calibration might occur in a centralized way (when all the observations are available at a central point) or in a distributed way (when observation are only available locally and the monitoring nodes in the network exchange aggregated information to generate a normal behavior model). The distributed nature of networks enforces us to consider the centralized and the decentralized machine learning problem.
- *Issues relative to detecting and recognizing anomalies*: a third challenge is coming from the last step of anomaly detection that consists of deciding if the deviation between a prediction made by the predictive normal model and the observations is incompatible with a normal behavior or not. This last step is difficult as Internet traffic is characterized by self-similarity, (multi-)fractality and long range dependence [60, 61, 62, 63]. Therefore it is hard to relate large variation of observations to anomalies. This makes the identification and mitigation of anomalies a very challenging task.

Despite these difficulties, constant progress has been made in network traffic anomaly detection. Methods have been created to detect anomalies in single-links and network-wide data, and techniques have been used to cope with the high dimensionality of network traffic data. Algorithms for network traffic anomaly detection have evolved from only being able to signal an anomaly in time to providing information about the actual flows that cause the anomaly.

This information is very valuable to network administrators that need to manually verify and mitigate potential anomalies, but is still not enough. Because of the characteristics of network traffic and the frequency of anomalies, it's not feasible for network administrators to manually analyze all anomalies detected by state-of-the-art detection algorithms. Network operators need more information than just the anomalous flows to be able to efficiently prioritize their time between anomalies. Automated classification of anomalies is the next step to give network administrators this information.

Integrating machine intelligence into the routing engine could be very valuable in this context as it will leverage the manual task of administrators and give them an automatic tool for detecting and acting on the source of anomalies with a low operational cost, low false alarm rate and very high detection rates.

We will here describe four use cases that could be applied to the intelligent machine that the project is going to develop.

6.3.1. Use Case Description

6.3.1.1. Anomaly Classification: The Required Next Step in Anomaly Detection

Although there has been some effort to characterize network traffic anomalies, automated classification has not received much attention (a notable exception is the work by Lakhina et al. [64]). Automated classification intends to add meaningful information to the alert of a detected anomaly. Besides the basic information of an anomaly's volume and traffic features, an algorithm for automated classification should be able to make complex derivations over the anomaly's characteristics. Ideally, the computed information can then be used to define the type of the anomaly or to at least help characterize the underlying cause.

6.3.1.2. Active Monitoring Applied to Anomaly Detection

Most of previous work on attack detection has been concentrating on monitoring traffic using passive measurement techniques. This part of this use case is to design an alternative approach based on active measurements, the objective being to make possible the real-time detection of DoS flooding attacks, without intrusive probing. It relies on the assumption that delays will be impacted in case of attack. The advantage of active measurement is its user-oriented nature which allows anybody to cope with DoS attack detection (whereas it is usually devoted to network administrators).

The end-to-end approach of active measurements also makes possible to detect attacks anywhere in the Internet from any source. This represents a great advantage in our situation since it allows us to potentially detect anomalies inside other ASes without an actual exchange of information with these ASes (e.g., if these ASes do not share strategic information). Active measurements would then significantly ease the design of a global attack detection system for the Internet.

Active measurements are very popular in the Internet for measuring delays, loss rates, inferring network topologies, etc. They are also used for estimating available bandwidth (see tools as Abing, Spruce, Pathload, IGI-PTR, Pathchirp, etc [57, 58, 59]). Our proposed method then relies on analyzing time series distributions of measured ICMP request/echo delays. Its originality relies on how these time series distributions are computed in order to exhibit anomalous values corresponding to DoS flooding attacks and classifying them. For this purpose, and taking advantage of previous work on anomaly passive detection which demonstrated the benefits of statistics or entropy analysis, our detection method also computes the entropy from distributions of RTT time series.

However, first results when working on the entropy function or Kullback-Leibler distance only were not convincing in active measurements, because of a high rate of false positives. The methodology will be extended by the use of the Hausdorff distance on distributions of time series, an index of dissimilarity (whereas entropy is a index of similarity) to reduce this level of false positives. Hausdorff distance was used many times in image pattern recognition. The combination of all these distances/indexes of similarity / dissimilarity allows us to reduce the rate of false positives and false negatives. Classification concerning active measurements remains however an open problem. The identification of ICMP-SEQ flood seems to be possible but others types of anomalies still seem to be unidentifiable [65].

The performance of the system is also a great criterion to evaluate its efficiency. This performance can be evaluated in two terms: reactivity and processing resources used. The way time series distributions are built/used allows us to tune these performance indexes. In fact each time series distribution is built with two parameters: the size of the window and the interval between each window or the overlapping between two consecutive windows. In fact, the greater the size of the window is, the shorter the time between each window or the bigger the overlapping are, the more accurate the system is, but at the expense of the processing resources used.

6.3.1.3. Two dimensions ADS (Anomaly Detection System)

6.3.1.3.1. Single Point ADS

This first step aims at designing an Anomaly Detection System (ADS) able to detect and classify traffic anomalies on a single link. Because of traffic variability and the low intensity of anomalies components, it is not easy to detect accurately anomalies, i.e. with a limited amount of false positives and false negatives. In addition, the recent work which relies on strong statistical or signal processing techniques does not provide a satisfactory solution. First, such solutions are not perfectly accurate: Even if they significantly reduce the number of false positive and false negative, they cannot guarantee that some attacks will not remain undetected, whereas some normal variations of the traffic will not get detected as anomalies. Second, if an anomaly is detected in the spectral plane or based on the entropy function, it is very difficult to then identify the anomaly characteristics to give network operator understandable information to cope with the anomaly. We then propose to separate the detection and classification functions. We then recommend to design an as simple as possible detection algorithm based on several volume parameters of traffic – for instance a simple one based on deltoids. The key contribution is then the design of a new algorithm for automated classification of network traffic anomalies. We plan to demonstrate how the information obtained by further analyzing the identified anomalous flows can be abstracted as anomaly attributes. These attributes can then be used in a signature-based classification module to reliably characterize different types of anomalies. By using meaningful and well-explained attributes, network operators can adapt the classification rules to their needs. We will define several of these attributes and show how different types of anomalies could potentially be characterized using them. We will show the expressiveness of this approach to reliably classify different types of anomalies (e.g., DDoS, network scans, and attack responses) and how it gives network operators the flexibility needed so they can use network traffic anomalies detection algorithms in a more efficient manner. The classification algorithm is then also devoted to false alarms reduction, i.e. it can classify the traffic which leads the simple detection algorithm to raise an alarm as normal, and thus stopping the delivery of a false alarm to the operator. In fact, the ultimate goal of the classification algorithm aims at determining the intension behind any anomaly.

6.3.1.3.2. Distributed Collaborative ADS

Network-wide anomaly detection methods developed up to now consisted of transferring a set of metrics to a central point where these metrics where compared to their expected values and divergence were detected. In these settings the operation of an anomaly detector could be divided into two related but different tasks:

- *Model identification*: this step is essentially of machine learning nature. In this step a relevant dynamical mathematical model of the normal behavior of traffic is extracted

by learning from measurements of network parameters during anomaly-less periods. The aim of this model is to predict the behavior of the traffic in either time or frequency domain without going precisely into the details of what is actually happening inside the network. This approximation results in a level of acceptable error in the precision of the model prediction. This error should be an independent and zero mean random variable, as any subsisting bias or dependence in the prediction error would mean that some predictive elements still subsist that could be exploited. The level of acceptable error is characterized by the model identification step. Different approaches have been developed for this step. Indeed the model identification step is subject to a complexity/accuracy trade-off, *i.e.* attaining higher accuracy could be done at the cost of higher complexity. One should also be careful about over-learning that results in the model having poor performance outside the measurement used for calibrating it. For this reason, as a rule of thumb the acceptable prediction error is usually overestimated in a conservative way to enable the use of less complex model that are still able to have good performance outside their learning sets.

- *Filtering*: the normal behavior model developed in the first step is used in this step to remove from observation what could be predicted using the previous model. The residue of this filtering is called the "*innovation process*" and represents the part of the observation that could not be predicted. This step is a classical signal-processing step. When the observed metrics are compatible with a linear Gaussian process assumption the optimal filter for implementing this filtering step is the Kalman filter. Nevertheless, because of its feedback structure the Kalman filter is highly robust to deviation from linearity and gaussianity. However, under more general hypothesis as non-linearity Extended Kalman Filter (EKF) could be used and a particle filter can deal with highly non-Gaussian measurements. This means that whenever the model identification results in a good predictive model, the filtering step become straightforward.
- *Decision step*: the last step of an anomaly detector is the decision step. In this step, the innovation process derived previously is compared with the level of acceptable prediction error obtained in first step through a statistical test. An anomaly is detected if this comparison leads to the decision that the observed innovation process is incompatible with an acceptable level of prediction error. If the modeling and filtering steps are done correctly one can expect to obtain at the decision step an independent and Gaussian process and to have to apply a simple hypothesis test, where we have to decide if an observed value of the innovation process resulting from the filtering step is compatible with the hypothesis of having been generated following a given distribution (resulting from the acceptable level of prediction error coming from the modeling step). Because of the inherent probable error in this hypothesis test, any anomaly detection can be characterized by a Receiver Operation Characteristic Curve (ROC curve) showing the trade-off between the risks of not-detecting an occurring anomaly *vs.* the likelihood that a detected anomaly has not happened in the real world. Because of the approximations done to implement the modeling and filtering steps with an acceptable complexity, very frequently the decision step have to deal with innovation processes that are not precisely independent and Gaussian process. This results in ROC curves that are below the optimal ROC curves expected when everything is perfectly following the theory. Nonetheless, that is *in fine* the ROC curve that could evaluate and compare the quality of different anomaly detector based on different modeling, filtering and decision structures.

Of the three above steps, the first step is essentially of machine learning nature; the two last steps are of statistical signal processing nature.

Classically, the three above steps are done in a central point where all the relevant measurements are transported. Whenever the problem of centralized anomaly detection is a complex enough problem, there have been a good number of researches that have been done on this topic. One can divide these researches in two classes: *parametric* and *non-parametric*.

The parametric approaches use a given model structure to capture the correlation existing in the observed measurements. This correlation is indeed used to predict the future value of observed metrics. This is this correlation captured in the predictive model that is filtered in the filtering step. Maximum Likelihood estimation of Dynamical Linear Systems and Principal Component Analysis (PCA) based modeling are noteworthy examples of parametric approaches that have been applied up to now to centralized anomaly detection.

However, non-parametric approaches follow a different path. In these approaches, we first apply to the data a generic and agnostic un-correlating transform that results in an uncorrelated signal that will retain the properties of the initial signal not anymore in its correlation structure (that has been wiped out by the transform) but rather in its distribution that could be used in the decision step. Because of the un-correlating transform, the predictive model now consists simply of a distribution of transformed observation that should be inferred by machine learning; the filtering step is not anymore needed and the decision step consists of checking if a transformed observed value is compatible with a given distribution. One example of un-correlating function is random linear projection that takes a set of observations and maps it into a random space using a linear projection with random coefficient. Because of the random nature of the projection, thanks to central limit theorem if the number of element used in the linear projection is large enough, the random linear projection results in a jointly Gaussian and independent random vector with concentration around its mean values. These mean values are the distribution that will be characteristic of the normal behavior. Although the non-parametric approaches seem simpler as they do not need a complex model calibration, they have specific complexity related to the choice of the number of observation to mix to obtain convergence of the random projections as well as the robust estimation of the density.

Out of these two approaches, the parametric approach have been the more investigated in the recent years, however recently the non-parametric approaches have been proposed and have resulted in interesting properties.

The more challenging problem we want in particular to investigate in the context of ECODE project is how to perform anomaly detection in a distributed way. Indeed, in a real network source of observation are distributed and one would like to be able to detect anomalies at the observation point rather than delegate the detection to a distant central point. One obvious way consists of assuming the distributed problem as several independent instances of centralized anomaly detection apply only on data observed at this point. However, this simplistic approach misses the correlation between measurement points that could be the only vantage point for distributed anomalies. This means that all the three above steps have to be done in a decentralized way if one wants to benefit from the full correlation structure.

As for centralized approaches, parametric and non-parametric approaches can be applied to the distributed problem. As described earlier, parametric approaches are heavily based on signal processing techniques, and extending them to distributed case becomes an instance of general distributed signal processing problems. The area of distributed signal processing has attracted during the past years a lot of interests. The main problem in distributed signal processing consists of mapping a signal-processing task over a set of distributed processors that will cooperate to achieve the needed task. Some recent researches have targeted implementation of distributed Karhunen-Loeve transform (that is the basis of PCA based approaches), as well as distributed Kalman filtering. Transferring these researches into the context of distributed anomaly detection in networks and implementing them on the ECODE platform is one of the main goals of our research in this project.

Non-Parametric approaches can also be extended to distributed anomaly detection. Recent researches have shown the interest of a generic technique coming from multi-user information theory and named "Slepian-Wolf Coding". This technique consists of exchanging random projections (sketches) between measurement points. These random projections act as side information that are used to build an overall model enabling the local decision about a global anomaly occurring in the whole network. Our aim here is to investigate the relevance of this approach to distributed anomaly detection.

6.3.2. Interaction with Learning Module

6.3.2.1. Distributed Anomaly Detection

- **Learning Stage:** Learning stage takes place whenever a recalibration of the normal behaviour model is needed. This happen normally at the begin of the activity as well as whenever a significant change has happened in the network (for example addition of a new customer that generate a high enough traffic to change its profile) or whenever a significant divergence is observed between actual network behavior an previous model.
- **Machine learning technique:** We expect to use parametric as well as non-parametric techniques during the project. Parametric methods need an unsupervised model calibration phase that can use several techniques as the EM (Expectation Maximization) or the MCMC (Monte Carlo Markov Chain) to estimate a maximum likelihood estimate of model parameters. Another approach is the Karhunen-Loeve transform (or its discrete counterpart the PCA). The filtering phase will use mainly Kalman filtering techniques. Decision phase is not of machine learning nature. Non-parametric techniques are mainly density estimation approaches over random mosaic. The decision phase uses statistical tests based on Kullback-Leibner distance.
- **Inputs variables to the machine learning module:** In the case of the distributed anomaly detection system, we consider that the inputs variables to the machine learning module consist on the local observations and the random projections coming from other nodes. Indeed, these inputs consist of histograms obtained over random projection of observation in other nodes. These inputs are finite length vectors of integers. Note also that inputs are time varying. Inputs are in fact time series. Time is an essential parameter for detection and classification algorithms.
- **Outputs of the prediction stage:** In the most general setting we expect the system to provide as an output the likelihood that an anomaly has been detected and eventually information about the source of this anomaly. However we should add to this

information forwarded to other node in order to help them to build global model of the network (for example random projection of local observation forwarded to neighbors).

- **Learning phase speed:** The learning phase speed depends on the parametric, non-parametric nature of the technique used. One can expect that parametric techniques will have a larger learning phase as they need to calibrate a model. We have carefully to evaluate this speed to ensure that an online operation is feasible.
- **Interaction with the routing/forwarding system:** At the first stage, we are mainly interested about anomaly detection and not about anomaly mitigation. Indeed, this is this last step that will have to interact with the Forwarding/Routing Engine. We will deal with this interaction in later stage.

6.3.2.2. Traffic anomalies detection

- **Learning Stage:** There are two learning processes in the ADS: a process for learning about anomaly signatures, and a process for learning about current traffic characteristics or alarm events produced by single point ADS. Defining anomaly signatures is a long process which requires strong analysis of several examples of the same types of anomalies. As the objective is to determine the human intension behind an anomaly, it seems impossible to make it automatically. Therefore, the definition of the anomalies signatures will be made manually. The anomaly signatures database can then be enriched anytime a new anomaly is discovered and fully analyzed and characterized. For the single point detection algorithm based on deltoids, it is required to learn about a simple traffic characteristic: the standard deviation on deltoids time series (difference between inter-arrival times of consecutive packets). This means that the system will have to compute this statistical value for a given time before being able to detect anomalies.
- **Inputs variables to the machine learning module:** The single point ADS uses as input the output of a passive monitoring system which provides a full packet trace, i.e. a file or stream containing a trace for all packets with at least for each packet full IP and TCP headers. This packet trace can be at the ERF format (the one of the DAG cards [66]) or any other (as the PCAP format of Wireshark for instance). Conversions between different formats should be easy. For avoiding waste of time in conversion, it would be better to use as inputs for the ADS the packet trace format of the used passive monitoring equipment. For the distributed collaborative ADS, inputs are coming from all other ADS involved in the collaboration. At the stage of the design, it is impossible to provide a complete description of such inputs. Anyway, we can imagine that such inputs (ADS outputs) are alarms. Alarms would certainly contain:
    - The type of anomaly detected (based on the classification process results)
    - A confidence level of the local detection and classification algorithms
    - A description of the anomaly characteristics which will depends on the anomaly type. It would however certainly contain source and destination addresses, source and destination ports, anomaly intensity, etc.

  Note that inputs are time varying. Inputs are in fact time series. Time is an essential parameter for detection and classification algorithms.
- **Outputs of the prediction stage:** We do not intend to make any prediction for ADS. ADS, especially the distributed collaborative version, only take advantage of the alarms received. But a new decision can be made anytime a new alarm arrives.
- **Learning phase speed:** It has to be as fast as possible in order to make possible detection as early as possible, as well as launching countermeasures.

- **Interaction with the routing/forwarding system:** The ADS will report any anomaly to the routing/forwarding system. The list of anomalies is not completely set yet. So, it is impossible at this time to completely define the set of alarms to be sent to the routing/forwarding system. However, we can imagine that the same kind of alarms as the ones described as inputs for the distributed collaborative IDS could be used, i.e. including an indication about the type of anomaly and its features. Then, in case of attack, the routing/forwarding system could easily drop attack-constituting packets. In case of legitimate anomaly as flash crowd, it could adapt the routing strategy in order to optimize the performance and QoS levels.

6.3.2.3. Active monitoring applied to anomaly detection

- **Learning Stage:** We need the ML algorithm to be able to give us some reliable threshold for each indices/distances and each parameter of these indices/distances. Example: a threshold for Entropy, Kullback-Leibler distance and Hausdorff distances [65] and the optimum values for the parameters of the distribution's calculation (size of the window and interval between each window or overlapping).
  Note also that, as we need to adapt to the network conditions, the learning should be on line.
- **Inputs variables to the machine learning module:** The inputs are the metrics gathered by the active measurements: real numbers (like times series, packet loss ratio). The values of the parameters in the distances/indices can also be seen as inputs. These parameters can be integer or real numbers. Note that the types of inputs are static but the values of the inputs are constantly changing.
- **Outputs of the prediction stage:** We need a binary classification to know whether we have an anomaly or not. But we also need a classification (if possible) for the type of anomaly detected (port scan, network scan, DoS, DDoS).
- **Learning phase speed:** It has to be as fast as possible in order to make possible detection as early as possible, as well as launching countermeasures.
- **Interaction with the routing/forwarding system:** The same interactions as we described in the Traffic anomalies detection case.

7. Routing

7.1. BGP

7.1.1. Use Case Description

7.1.1.1. Path Exploration Overview

Behavior of *path vector protocols* such as BGP is inherently associated with their *path dependencies*: the path selected by a router depends on paths learned by its neighbors which, in turn, is influenced by the paths selected at the neighbors' peers, and so on. This property of exchanging vectors of ASes (or paths) to prevent routing loops leads also to the so-called *path exploration* [18, 20] phenomenon that delays BGP protocol convergence [20].

As a path vector protocol, BGP exhibits thus path exploration phenomenon. We illustrate the path exploration phenomenon by an example and then describe why, in general, it is impossible to avoid this phenomenon by solely relying on the AS_Paths associated with BGP routes. In this example, we denote an AS_Path as [A_n, A_{n-1}... A_1, A_0], where A_0 is the origin AS to which *d* belongs and A_n the local BGP router.

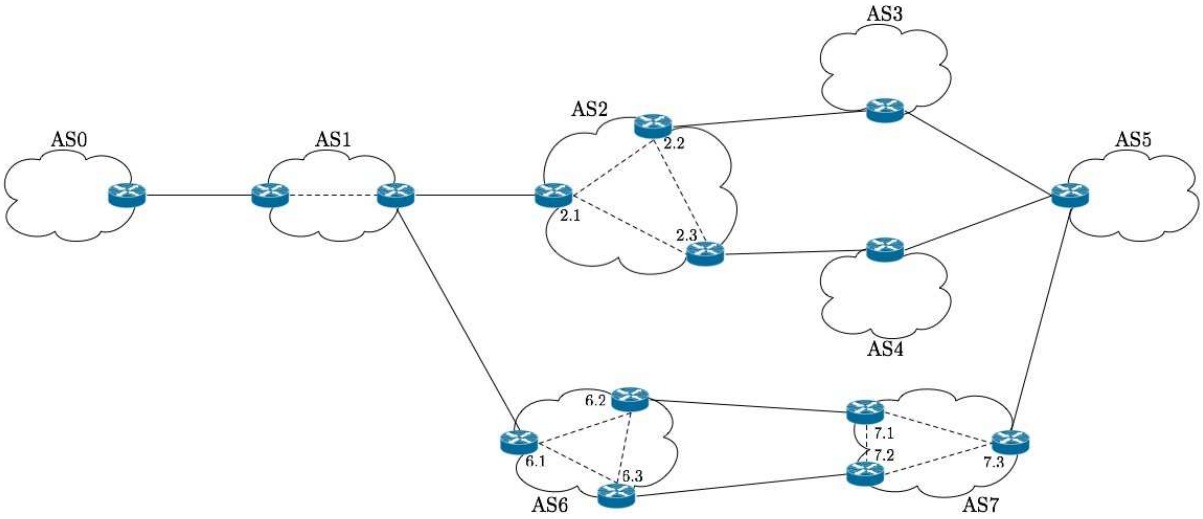

Figure 7.1: BGP and path exploration. Solid/dashed lines represent eBGP/iBGP sessions

Consider the topology in Fig. 7.1. Now suppose AS_0 announces a path to destination *d*, this announcement is received at its neighbors and propagated hop by hop. Finally, when the network converges, AS_5 knows three paths to reach *d*, i.e., [3,2,1,0], [4,2,1,0], and [7,6,1,0] (preferred in that order).

Now consider what happens when the link between AS_0 and AS_1 fails, making *d* unreachable at AS_1. This failure triggers the following sequence of events: AS_1 sends withdrawals to AS_2 and AS_6. In turn, each of them sends withdrawal to their own neighbors. Eventually, AS_5 will receive withdrawals from each of AS_3, AS_4, and AS_7 (in some order). Suppose the first one was from AS_3. AS_5 then removes the path [3,2,1,0], selects [4,2,1,0] as the best path and sends it to its (other) neighbors. However, if the withdrawal from AS_4 arrives next, then this best route is invalidated and AS_5 selects (and announces) [7,6,1,0]. Finally, after AS_5

receives the withdrawal from AS_7, it invalidates the path announced earlier and sends a withdrawal.

This cycle of selecting and propagating (invalid) paths is termed path exploration. Clearly, the cycle stops after all the obsolete routes have been explored and invalidated.

7.1.1.2 BGP Event characterization

The following parameters are possible learning parameters as they characterize events in BGP:

1. Temporal characteristics
    - Arrival time
    - Inter Arrival time (IAT)
        between previous BGP update message
        between previous BGP update message with spatial characteristics
2. Spatial characteristics
    - AS originator
    - Prefix
    - AS_Path (which characterizes the type of event: longer, shorter or equal)
    - Peer (BGP speaker from where the BGP update message is received)

Note the assumption is made here that a pattern detection technique can be used so as to identify path exploration occurrence. Indeed, BGP is a path-vector protocol: localizing topologically where a failure happened in the network is not trivial in BGP as only path-vectors are communicated on updates learning opportunity. The objective is here is thus the detection path exploration in BGP and, if possible, accelerate this process (i.e., avoiding the exchange of useless BGP UPDATE messages between routers) so as to eliminate its detrimental affects on BGP convergence [19]. After its detection, a router could decide to select directly to the actual (and correct) path that will be advertised to its downstream peers. A desired side effect of this acceleration would be to reduce the BGP churn generated during the path exploration. Compared to other damping techniques the process here does only rely on an accelerated route selection process. Other solutions have been proposed [67, 68] but they have the disadvantages of being very difficult to implement and to deploy due to modifications to BGP itself.

7.1.2. Interaction with Learning Module

7.1.2.1. Input to the Machine Learning Module

The input to the machine learning module takes the form of (a sequence of) BGP UPDATE messages. Each BGP UPDATE message takes the form of a byte stream as detailed in [18].

Each BGP UPDATE message contains several attributes such as the *Network Layer Reachability* (NLRI), the *Withdrawn Routes*, and the *Path Attributes* [18]. The Path Attributes are of keen interest for us, particularly, the AS_Path, that identifies the ordered list of AS numbers through which routing information carried in the UPDATE message has passed [18].

7.1.2.2. Output of the Machine Learning Module

The output of the learned model should help in deciding whether the received BGP update message should be forwarded to the router's neighbors or not and which update shall be

generated by the local BGP router to its own neighbors. Depending on the class (see below) in which the BGP message falls, the UPDATE message is propagated or not.

- If the update belongs to the *exploration free* class it can be forwarded per MRAI (Minimum Route Advertisement Interval, the minimum timer between two UPDATE messages) local setting.
- If the update belongs to the *exploration seed* class and that prefix is withdrawn:

The local BGP router should prevent selection of an "exploration cycle" AS_Path (if present). If an "exploration cycle" is received as part of the next update then the router should hold its selection. The local BGP router should prefer selection of another "exploration seed" AS_Path (if present). If none are present for that destination before half of the M_d timer for that destination prefix *d* elapses, then the local BGP router shall select one of the "exploration free" sequence.

7.1.2.3. Classifying BGP Events

A path exploration phase is characterized by a withdrawal of a route towards a destination prefix previously selected and installed in the Local RIB (i.e., the destination prefix is declared unfeasible) followed by the advertisement of that destination prefix with a longer AS_Path, a different AS_Path of the same length, or with the same AS_Path but different attributes. To determine occurrence of such event(s), the task consists in classifying the AS_Path that are inducing path exploration from those that are "safer". As such this classification can be easily extended so as to discover the seed sequences.

7.1.2.3.1. Binary Classification

The classification task consists in determining if a BGP UPDATE message falls within a path exploration phase or not.

- If the BGP UPDATE message does not fall in path exploration phase (path exploration free), then the BGP UPDATE message is propagated as described in the BGP specifications [18].
- If the BGP UPDATE message falls in path exploration phase (path exploration seed), then determine if the sequence to which the BGP UPDATE message belongs will
    - Either stabilize to a newly preferred path for the same prefix
    - Or not. In this case, the path exploration phase ends with no path for the destination prefix.

Base on this classification the purpose is to accelerate the BGP exploration phase either by directly determining which BGP UPDATE message to be propagated to the router's BGP peering neighbors (that avoids the exploration sequence) or by directly determining the absence of a stable preferred route for the destination prefix. A specific message indicating the absence of a stable path for that destination prefix to the BGP downstream neighbors may be considered.

7.1.2.3.2. Classification Based on Pivot AS

In this case, the classification criterion is based on the presence of a pivot AS in the AS_Path. A pivot AS is defined as an AS giving access to a critical link (which is in most cases a link defining a client-server relationship between two ASes [20]). Critical links can be identified using spectral analysis (see below). In the example depicted in Fig. 7.2, AS_2 is a pivot AS.

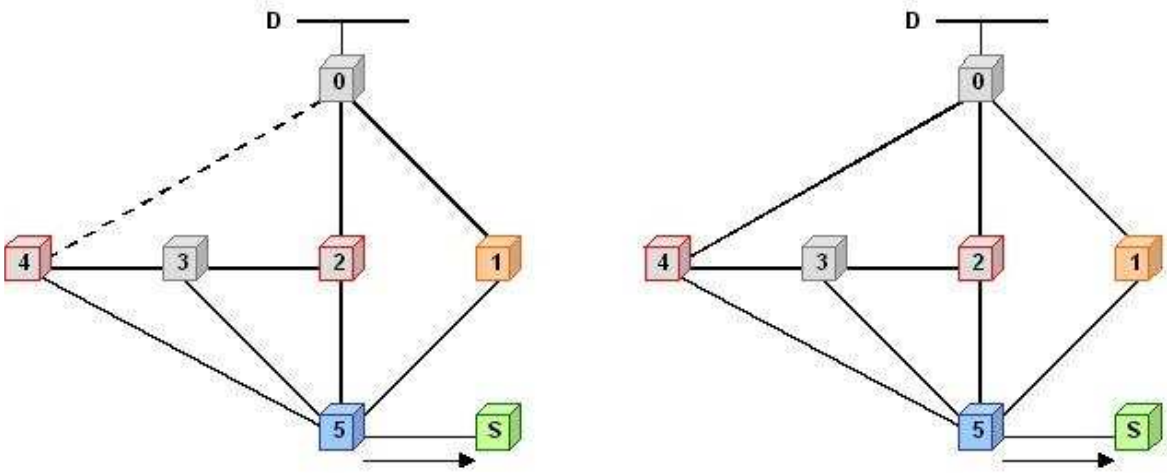

Figure 7.2: BGP Path Exploration Example

The classification task consists then in classifying AS_Paths per destination prefixes *d* as follows:

- *Class 1*: includes the set of AS_Paths that do not cross a pivot AS. Using the example depicted in Fig. 7.2: AS_Path [1,0] belongs to that class. These AS_Paths are expected to be exploration free.
- *Class 2*: includes the set of minimum length AS_Paths that crosses pivot AS (AS_2 and AS_4). AS_Path [2,0] belongs to that class. These AS_Paths are exploration seeds. Indeed, the AS_Path of Class 2 [2,0] defines a seed sequence because after its withdrawal, an update sequence closely coupled in time of longer AS_Path selected from Class 3 would result in exploration phase.
- *Class 3*: includes the set of other AS_Paths that cross a pivot AS (AS_2 and AS_4). Using the above example: [3,2,0], and [4,3,2,0] belong to that class. These AS_Paths are segments part of exploration cycles. When the element under failure is removed from the AS topology, the concatenation of the remaining AS_Paths (e.g. [5,4,3,2] x [2,5]) with the AS_Path between the pivot AS and the local AS shows a cycle.

A timer defined as M_d = *T_Max* length sequence - *T_Min* length sequence is maintained per destination prefix. The latter sequence belongs to the exploration seeds class. One timer is maintained per exploration seed sequence.

Consider now the topology depicted at the right hand side of Fig. 7.2, and let us assume that a link between AS_0 and AS_4 is added, or AS_4 is subject to a policy change. A new set of AS_Paths will reach AS_5: [4,0], [3,4,0], and [2,3,4,0]. The AS_Path [4,0] falls into Class 2, whereas AS_Paths [3,4,0] and [2,3,4,0] falls into Class 3.

To determine critical links, we make use of spectral analysis of the inter-domain routing topology. This technique provides for metrics allowing to extract most important global characteristics of the topology: spectrum-related metrics provides bounds for critical graph characteristics such as distance-related parameters, expansion properties, and values related to graph resilience estimation under node/link removal. The latter provides measure of network robustness under link removal (equals minimum balanced cut size of a graph).

We define the matrix *A* as the $n \times n$ adjacency matrix of a graph constructed by setting the value of its element as $a_{ij} = a_{ji} = 1$ if there is a link between nodes *i* and *j*. All other elements have value 0. Scalar λ are the eigenvalue and vector *v* the eigenvector of A if $A\ v = \lambda\ v$. The

spectrum of a graph is the set of eigenvalues λ of its adjacency matrix *A*. Most networks with high values as eigenvalues have small diameter, expand faster, and are more robust. The graph's with largest eigenvalues provide bounds on network robustness with respect to both link and node removals.

7.1.2.4 Interaction with the Routing System

BGP UPDATE messages constitute the input to the machine learning module. BGP routers (or speakers) advertise network reachability information about destinations by sending to their neighbors UPDATE messages containing set of destination address prefix announcements (feasible routes) or withdrawals (unfeasible routes) together with attributes associated to a path to these destinations.

- An *announcement* informs neighboring BGP routers of a path to a given destination. When a local BGP router propagates a route learned from the UPDATE message sent by one of its peering BGP routers, it modifies the route's AS_Path attribute based on the location of the BGP router to which the UPDATE message containing that route will be sent.
- A *withdrawal* is an update indicating that a previously advertised destination is no longer reachable. Route withdrawals only contain the destination and implicitly tell the receiver to invalidate (or remove) the route previously announced by the sender.

A BGP router receives UPDATE messages from its BGP peering neighbors following a time varying interval bound by a minimum threshold. As mentioned in [18], there is a minimum amount of time (MRAI) between two BGP UPDATE messages sent towards the same BGP router. Thus, a given BGP router receives one BGP UPDATE message per MRAI time interval per neighbor (and sometimes per destination prefixes *d*).

The output (i.e., the class of the BGP UPDATE message) of the learning process is used by the selection process of the local BGP router. This output is not distributed to the router's neighbors or other nodes in the system. However, the router's output (i.e., the BGP update messages that are forwarded) will influence the route selection of the BGP router's neighbor as depicted in Fig.7.3 (the crossed circles represents selectors acting at the input and output of the "BGP route selection" process).

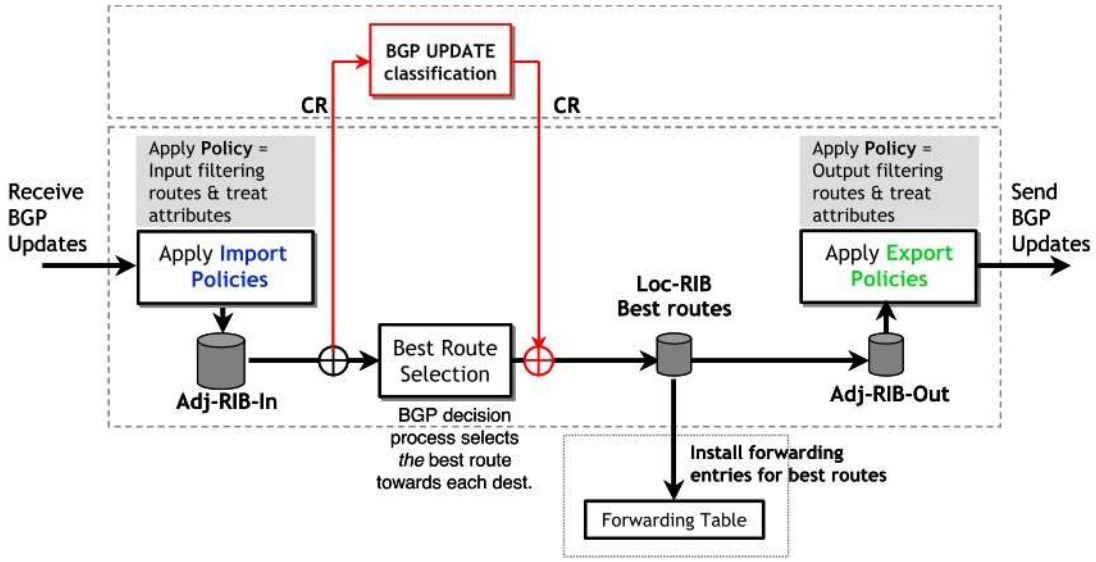

Figure 7.3: Interaction between the routing use case and the Machine Learning Engine

7.1.2.4 Machine Learning Process and Technique

The semi-supervised or unsupervised learning algorithm would be executed online and be distributed on BGP routers. For maintenance and further development reasons, it is important that the learning model is understandable, possibly at the cost of an accuracy loss.

The prediction stage must be executed as fast as possible because a BGP router may receive a huge quantity of BGP messages within a small time window. Consequently, the prediction shall be executed as fast as possible to prevent slowing down the whole BGP route selection process.

7.2. Network Recovery and Resiliency

7.2.1. Use Case Description

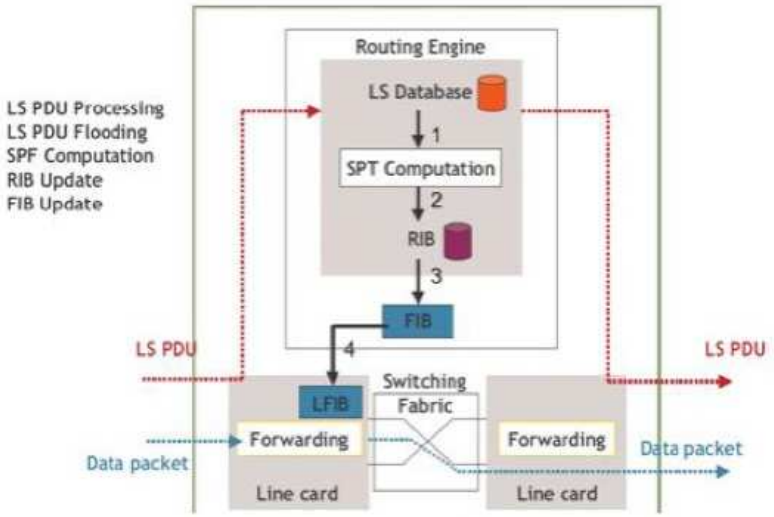

Figure 7.4: Router update process

One can model the recovery process in IP routers as a sequence of four-steps:

1. Computation of the shortest path tree based on an updated link-state (LS) database: taking 30 to 50 microseconds per node in the network.
2. Update of the central routing information base (RIB) based on the shortest path computation.
3. Update of the central forwarding information base (FIB) based on the RIB.
4. Distribution of forwarding information base towards the line cards: taking 30-50ms per set of updates (see Fig. 7.5).

Figure 7.5: Update process time

7.2.1.1. OSPF Event Modeling and Clustering

In an IGP environment, from the moment a Link State Protocol Data Units (LS PDU) is received that is newer than the current database contains, the OSPF protocol [12] will automatically trigger step 1 in the process as shown in Fig. 7.4. However for certain events,

more intelligent behavior is possible based on historical behavior of two types of PDUs: Hello messages which are sent between neighbors to poll the status of the adjacency and LS messages which together – from all routers in the network – build up a view on the entire network topology by communicating the state of their links. Hello messages are sent on a periodic basis, typically every 10s. LS PDUs have both periodic transmissions (typically every 30 min, at half MaxAge time, parameter fixed at 60min), and triggered transmissions upon a network topology event (failure, new link/node, etc.).

Failure detection between OSPF routers can be based either on OSPF Hello PDUs or on a faster Hello protocol such as Bidirectional Forwarding Detection (BFD [42]), where the periodicity is in the order of tens or hundreds of milliseconds. The neighbor adjacency between two OSPF routers typically is declared down from the moment the timer reaches RouterDeadInterval, which is typically a multiplier (e.g., three) of the HelloInterval itself (BFD reacts similarly). However, using the RouterDeadInterval technique does not take into account the historical behavior of sent and received Hello PDUs. This means that it is not improbable that failure detection can happen earlier by several orders of magnitude compared to the usual periodicity.

A possible policy using a probability model for Hello PDUs arrival time can for example assume a failure from the moment the modeled chance that a Hello PDU will still arrive becomes too small (see Figure 7.6). Modeling will become more difficult because of the randomization factor by which the periodicity at the sender will have a delay of 0-15%, meaning that a message will be send in a time frame varying from 15% earlier then the average value till 15% later. Techniques will need to overcome this by taking additional input into account such as the probability of the sequence of several messages, the correlation between Link State PDUs in multi-access segments, etc. In multi-access (MA) networks, for example Ethernet networks, Hello adjacencies are only held between an elected Designated Router (DR) and other routers connected to the MA network. Because the underlying MA network topology is unknown (being a layer 2, while IP is network layer 3, see the possible topologies in the cloud in Fig. 7.6), routers do not know beforehand which adjacencies are correlated. Learning these correlations can help routers to decide faster on how to reroute, and what traffic to reroute earlier. For example, if a failing adjacency between the DR and node A always seems to imply a failure between the DR and node B, one does not have to assume that node B is reachable via the MA network.

Sequences of LS PDUs can be correlated in different ways, for example: node failures resulting into related LS Updates for the adjacent links, Shared Risk Group (SRG) failures resulting into related LS Updates of links being contained in the group or MA network failures resulting into correlated LS Updates of DR adjacencies that failed. Figure 7.6 illustrates that if the node in between node a, b, c and d fails that all of these nodes will flood their LS Updates over the network. This means that all routers receiving these LS Update messages will recalculate all routes (step 1) on every update, taking into account the announced link failure. However more efficient procedures are possible. Goyal et al. make use of hold-off timers to wait for routing database update until it is sure that no node failure has occurred [43], but this has the following issues i) what is the ideal hold-off interval, and ii) how will you synchronize different router databases.

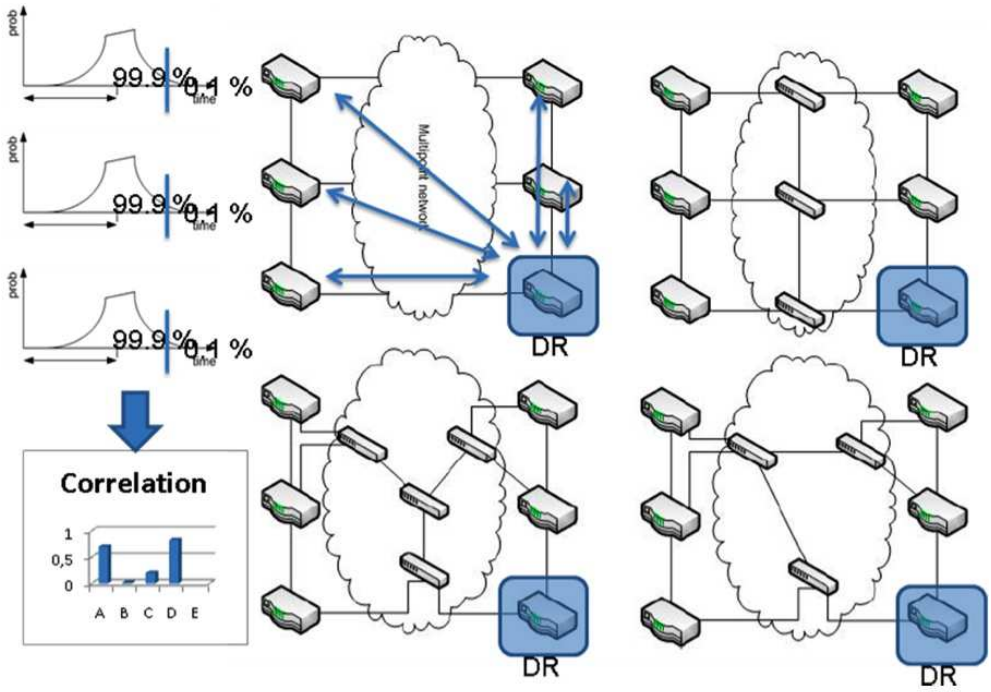

Figure 7.6: Modeling and Correlating Hello PDU Arrival Time

Instead of waiting on possible next LS Updates, another option could be to predict the type of failure based on the sequence of received LS Updates. This can be illustrated for example, using Fig. 7.7. Let us assume that as a result of a failure, node receives LS Updates from A and B. These LS Updates indicate that A and B have lost connectivity to the node in between A, B, C and D. Node E could make the prediction, based on historical behavior, that it is more probable that the node X located in between nodes A, B, C and D failed rather than only the link from node A towards X and/or the link of node B towards X. The difference with Goyal et al.'s solution of [43] is that the switch-over happens as soon as possible, and that no arbitrary waiting time is involved to expectedly reach synchronized router's decision. There is a chance that the wrong prediction is made resulting in a penalty on the network routing. However, as the prediction will act as a worst-case prediction (node failure or not), no additional packet loss can be caused by this prediction. Similar techniques should be developed in order to detect the simultaneous failure of links belonging to SRGs.

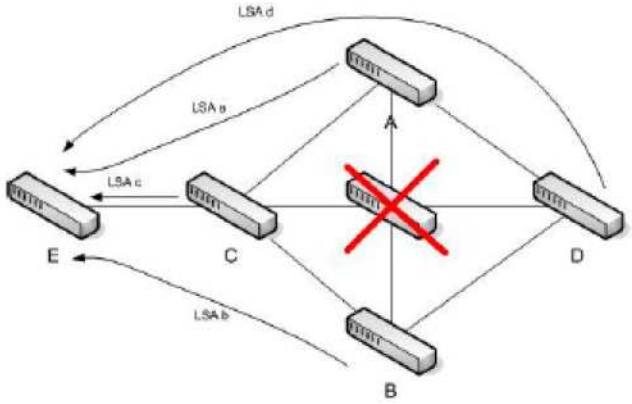

Figure 7.7: Correlated LS Update PDUs on node failure

7.2.1.2. OSPF Cycle Detection and Prediction

During the process of OSPF re-convergence, temporal loops can occur. This is illustrated in Fig. 7.8. Even a simple topology as depicted in this figure a temporal loop can happen upon failure. Let us for example assume that the link between B and C fails, and that C and D have updated their FIB, but E has not updated its own FIB. For packets from C towards B, this can result into C forwarding packets to D, forwarding packets to E, sending them back to C, because E still has a forwarding entry with as next-hop C on its shortest path to B.

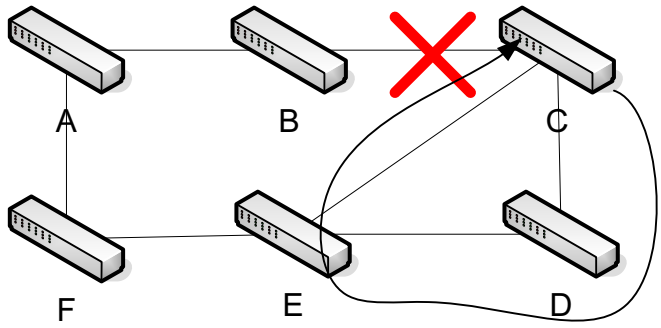

Figure 7.8: Temporal loop on failure

This can lead to temporal packet loss and local network overload. Some techniques such as ordered FIB [44] updates are designed to avoid transient loops. However, this technique can slow down re-convergence and changes the router update process.

Therefore, a learning technique is desirable which is able to quickly detect cycles during the re-convergence process. The desired technique can mark packets that make use of updated forwarding entries (for example, updated FIB entries in C). Based this marking, neighbors e.g. D only make use of update forwarding entries to forward these (re-routed) packets. If such entries are not available, a limited number such packets can be used as probes to detect possible resulting cycles. From the moment a cycle is detected for some prefixes associated to a certain interface, another forwarding interface could be selected to avoid the detected cycle. Besides this reactive component, a predictive component should be made such as to forward prefixes over the interface which has the lowest probability of forming cycles.

7.2.1.3. Minimizing Packet Loss During Routing Table Switch-Over

By default the updates from step 2 towards 3 of the cited update process happen in random order, meaning that a prefix update related to important traffic stream could be delayed much longer than necessary. More optimized behavior should take into account real-time traffic characteristics such that updates can be made more efficient, i.e. packet loss during switch-over should be minimized. However, the goal of introducing additional logic to make achieve this behavior should not result in heavy computational tasks, as this would again slow down the entire process; thus, contradicting the initial goal. Therefore, the following is targeted:

1. Develop a real-time learning traffic monitoring mechanism which is able to detect the ideal interval in between which traffic needs to be monitored and for how long it needs to be monitored. The resulting mechanism maintains statistics (by means of counters) of the number of packets, per destination prefix in the forwarding table. The intensity of this task is linearly proportional to the number of destination prefixes. Thus, the monitoring mechanism should be counter-adaptive: large sampling rate for large flows (elephants) while being able to monitor small flows (mice) by means of

small sampling rate. As this is a well-investigated research domain, further elaboration can be based on for example ANLS techniques [45].

2. Develop a classification mechanism to rank the resulting tuples (traffic stream, volume) in a set of classes. The mechanism should be computationally more efficient than known sorting algorithms (see Sec. 7.2.2.). A first approach of using machine learning for selecting sorting algorithms can be found in [46].
3. Work out a heuristic such as to optimize the quantum interval over time given the input of a set of traffic streams, their volumes and their associated classes (previous step). The ideal quantum time results into minimal packet loss by making an intelligent choice avoiding that more high importance traffic classes (previous step) have to wait on lower important classes in order to be switched-over.

7.2.2. Interaction with Learning Module

The intra-domain routing system is distributed as network nodes communicate by sending each other Hello PDUs (neighbors) and LS PDUs are flooded to all the network nodes (input). Output among the nodes is restricted to LS Acknowledgement PDUs sent to each other to confirm the processing of a received LSU PDU. The decision making in the routing system is decentralized as all the nodes build up their routing table individually (however being based on information synced with other nodes).

7.2.2.1. Inputs to Machine Learning Module

The following input is available at nodes:

- Routing protocol related (all input is locally available): Hello PDUs from OSPF or BFD, LS PDUs from OSPF, arrival time and associated header fields, and link load information.
- Traffic related: during measurement at the forwarding plane and from a given traffic stream.

7.2.2.2. Output of the Machine Learning Module

Inputs are indeed time-varying as PDUs and their related attributes change over time. The desired outputs are depending on the desired tasks:

- OSPF modeling and clustering
- Traffic dependent router update processing (based on traffic-related input)
- Predictive module for estimating effect of routing configuration

7.2.2.3. Interaction with the Routing System

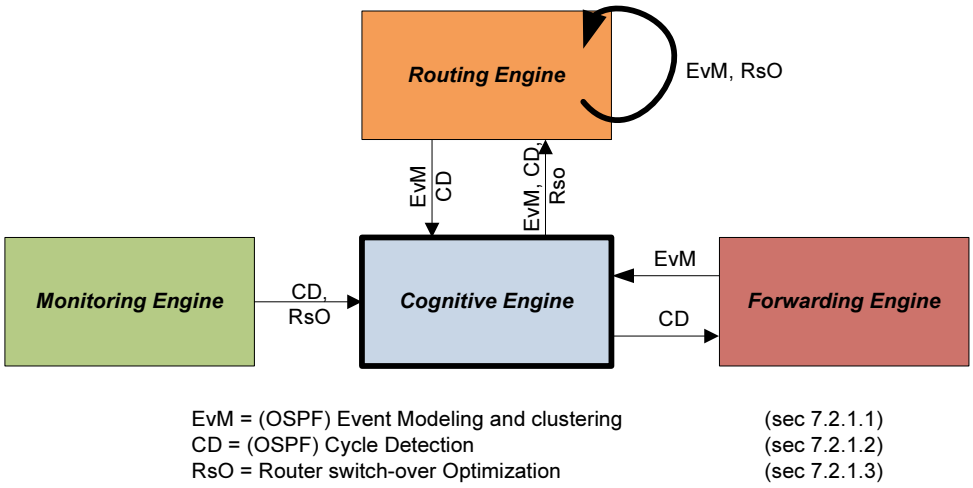

Figure 7.9: Interactions of the resilience use case within the ECODE architecture

For the case described in Sec. 7.2.1.1, the Machine Learning Engine (or Cognitive Engine) receives its input from either the Routing Engine (via `CR` interface) or the Forwarding Engine (via `CF` interface). The collected information comes from incoming PDUs from OSPF or BFD. This input is used to learn correlations between OSPF events (clusters). These will be stored in the `KIB`. When a certain pattern is detected in the Machine Learning Engine (for example early failure detection, or failure correlation) a decision is sent to the Routing Engine (via `CR` interface) as shown in Fig. 7.4. This decision should subsequently trigger SPF re-computation. The entire learning process of this case is local.

For the case described in Sec. 7.2.1.2, the cycle detection functionality in the Machine Learning Engine receives its input mainly from the Monitoring Engine (runtime traffic) and the Routing Engine (the used topology deduced from the Link-State database), and again stores learned information in the `KIB`. Upon cycle detection, the Machine Learning Engine signals to the Routing Engine and Forwarding Engine to directly send cycling traffic on another interface. The entire learning process of this case is local.

For case discussed in Sec. 7.2.1.3, the Machine Learning Engine needs path- and traffic-related information from the Monitoring Engine in order to model the traffic classes and to deduce the ideal monitoring and inter-monitoring interval. These last two parameters will be fed back to the Monitoring Engine (both directions via the `CM` interface). Upon the reception of a LS Update, the routing engine triggers the Machine Learning Engine in order to optimize its routing-forwarding table update process in function of minimizing packet loss (requesting for traffic classes, and ideal quantum interval sequence). The entire learning process of this case is local.

7.2.2.4. Machine Learning Process and Technique

For case described in Sec. 7.2.1.1, learning will happen at two stages: one before and one after failure occurrence. Before the failure, one can only model typical protocol dynamics such that the given model can be used to estimate how far the current situation falls regarding the modeled protocol dynamics. Once a failure has occurred, this information can be used as feedback and be taken into account such as to model correlated failures in point-to-point networks (SRGs) or MA networks (correlated adjacencies, Sec. 7.2.1.1).

All learning involved happens online, except for Sec. 7.2.1.2. For the case discussed in Sec. 7.2.1.2, learning happens during traffic measurements, to deduce traffic behavior such that monitoring intervals can be estimated continuously to classify the given traffic streams. All learning involved happens online except for classification for which it is possible to introduce upfront learning.

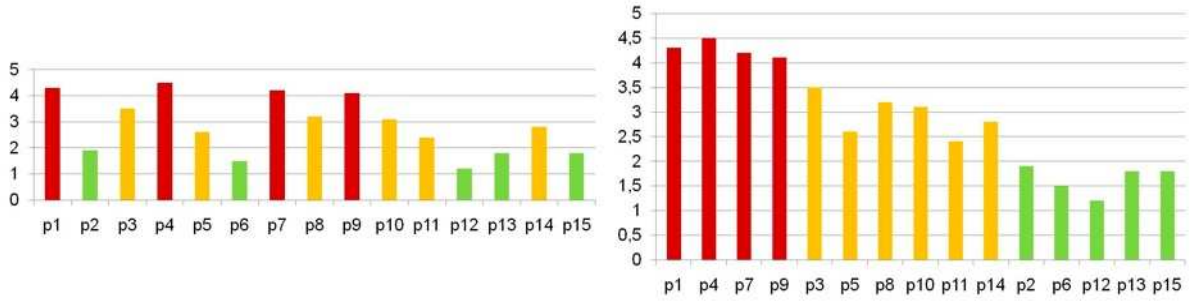

Figure 7.10: Classification of traffic streams in terms of volume

8. Path Selection

8.1. Informed Path Selection

8.1.1. Use Case Description

While, previously, a single path between two machines was assumed, we are now seeing with the evolution of the Internet topology expansion the rising of multiple paths with different performance. For example, end systems can be dual stack, or their network can be multi-homed. Another example is when LISP (*Locator/Identifier Separation Protocol*) [33] is used, thus allowing a given identifier (i.e., a client or server) to be reached via multiple locators (i.e., edge routers). In such a context, it is crucial for both applications and operators to easily select the path that better suits their needs. This is clearly the issue we tackle by proposing a generic path selection service that works in any context requiring a path selection.

8.1.1.1. Path Selection Requirements

We are heading towards an Internet that makes available a set of paths to the host (in terms of source and/or destination addresses) for reaching, for instance, any content. In such a context, it is thus crucial for applications to select the "path" (i.e., the <source, destination> address pair) that better suits their requirements. These requirements might be expressed in terms of network performance (e.g., delay, bandwidth). In addition, it would be interesting for applications to increase their reliability. Indeed, by taking into account multiple paths, whenever the current path fails, it should be possible to quickly and easily switch to another path. Further investigation should reveal the real benefits or drawbacks of such a switch. Another requirement for applications refers to the cost associated with network usage. Applications might want to favor the cheapest path and, consequently, decrease their networking bills. In addition, battery-based devices would like to avoid consuming too many resources when selecting a path. This means that, ideally, the path characterization should be done elsewhere. Finally, applications should be able to decide by themselves which path to use, meaning that the path selection cannot be forced on them by a third party but rather suggested.

Allowing each application to make this selection by itself is not a sustainable solution (for scaling and triggered control/responsiveness and consistency/performance reasons). Instead, letting applications to cooperate with operators might lead to a win-win situation. Indeed, by helping applications to select a path, operators can meet their own requirements. The first requirement of an operator concerns the details it should reveal to allow the path selection. It is obvious that operators do not wish to disclose details on their topologies as well as their policies. Operators might also want to influence both incoming and outgoing traffic, so that they could control their own network usage in terms of performance but also in terms of cost and traffic engineering (i.e., some links could be defined as primary links while others are backup links).

As a global requirement (i.e., it is common to applications and operators), any path selection service deployed should be scalable, should not imply a change in routers, nor add state information in them. Moreover, it should also be generic, i.e. addressing future scenarios.

It is well recognized that traffic fluctuates with time. It is thus mandatory to build traffic engineering systems that are traffic independent. This is clearly the approach we follow in this section when discussing the high-level behavior of a path selection mechanism.

8.1.1.2. IDIPS Server

Our path selection mechanism is called IDIPS "ISP-Driven Informed Path Selection" [37, 38].

Our assumption is that specialized boxes are installed in the network. These boxes, called IDIPS servers, are in charge of running a path selection algorithm reflecting the operator requirements in terms of traffic engineering. Every time an application or service needs to select one path among others or to rank a list of paths, it contacts the box that replies with ranked paths. In our terminology, the specialized box is called a "server" and anything querying the server is called a "client". It is worth noticing that more than one server can be deployed in the network and that clients do not have to deal with that (e.g., servers can be deployed in anycast).

With respect to the ECODE architecture (see Sec. 5), the IDIPS functionality can reside in routers or in dedicated servers. When it is activated in a router, the local interfaces `CR`, `CF` and `CM` (see Fig. 5.2) will be directly available to access the Routing, Forwarding and Monitoring Engines present in the router. When the IDIPS functionality is deployed in separate servers, the principle remains the same, but the local interfaces `CR`, `CF` and possibly `CM` will be accessed remotely.

Any request sent by a client contains the following information: a list of sources, a list of destinations and an optional performance criterion (e.g., route stability). The server processes the request and builds a list of all possible paths based on those two lists. This paths list is then ranked using information on the network state owned by the server such that the higher the rank the more promising the path.

Sources and destinations sent by the clients are typically IPv4 or IPv6 addresses. Instead of replying with complete addresses, the server can work with prefixes. In other words, if two paths with different ends have the same rank and if it is possible to aggregate the sources within a single prefix and to aggregate the destinations within a single prefix, the returned ranked list will only contain one path where the source is a prefix encompassing the two source addresses and the same for the destination. Aggregation offers several advantages. It

allows one to reduce the size of the replies (e.g., a single prefix can include several addresses indicated in the request) as well as the amount of potential paths to process. In addition, it avoids revealing topology details and network policies to clients. Nevertheless, a drawback in such an approach is the loss of precision (e.g., the reachability of a prefix is not the same as the reachability of a host).

In addition to a ranked prefix pair list, the server reply contains a time-to-live (TTL) information indicating how long the path ranking remains valid. This TTL is configured by the network operators and depends on the performance criterion provided by the client. When the TTL expires, it is up to the client to contact the path selection service to obtain a new path ranking.

Considering ranked prefix pair lists allows one to reduce the risk of attacks or the disclosure of sensitive information to competitors, which is often required by ISPs. Further, it allows the operators to modify the ranking algorithms according to their needs without involving clients. It thus separates the clients and operators while enabling cooperation.

8.1.1.3. Cooperation between IDIPS servers

Better scalability can be achieved if an IDIPS server can rely on other IDIPS servers (typically in other ASes) to reduce its measurement load. For example, IDIPS servers could measure path characteristics between themselves, rather than to (all) remote locators/prefixes.

If IDIPS servers are all associated with a large set of locators/prefixes, and if every IDIPS server can measure locally the characteristics of paths towards its own locators/prefixes, we may approximate the characteristics of a path by combining the characteristics of the following 3 paths:

- from source to source IDIPS (monitored by source IDIPS),
- from source IDIPS to destination IDIPS (monitored by either or both of them),
- from destination IDIPS to destination (monitored by destination IDIPS).

If there are N IDIPS servers and K locators/prefixes per IDIPS server, the number of paths to be monitored per IDIPS server boils down from $O(N*K)$ to $O(N+K)$.

This is a win-win situation, because all IDIPS servers win if all co-operate. To this end, IDIPS servers in ASes could join a P2P (peer-to-peer) system. Similarly some edge networks could run an IDIPS server on their edge router, and join this P2P system.

For delay measurements for example, the measurement load can be further reduced. If N servers ping all the other ones, the measurement load is $O(N^2)$. Now if IDIPS servers run an Internet Coordinate System (ICS), e.g., Vivaldi [39], every IDIPS server pings a fixed number (say 32 or 64) of other IDIPS servers only, and infer their coordinates in a suitable metric space, allowing them to estimate other delays that are not measured. The load boils down to $O(N)$ instead of $O(N^2)$.

In this scenario, every IDIPS server would also accept to disclose its coordinates to others. This can be seen as an extra level of co-operation, but it pays off, reducing once again the measurement load by a factor N.

8.1.2. Interaction with Learning Module

An important engine of the IDIPS server, called "Path Information Collector" (PIC), collects path information. Information is of two types:

- Administrative information (i.e., network policies and billing, but also routing information such as BGP or IGP);
- Measurements information (i.e., active and passive measurements).

This PIC element can be seen as a particular Monitoring Engine (see Fig. 5.2). Therefore, it will communicate with the Machine Learning Engine to provide its measurements. The Machine Learning Engine will learn from these data and communicate in return with the Monitoring Engine, e.g., to adjust the frequency of active measurements. The nature of these exchanges will be further elaborated below.

8.1.2.1. Measurement prediction and adaptation

The problem of predicting future QoS quantities from past observations can be stated, in machine learning terms, as a *time series regression* problem. The data *D* consists of measurements from the previous time step *t-1* back to some time index *t-l* in the past: $D = \{y_{t-1}, y_{t-2}, \ldots, y_{t-l}\}$. A model of this time series aims at predicting the next value of the metric y_t and, possibly, additional values in the future y_{t+1}, y_{t+2}, … This problem thus reduces to a *time series modeling and prediction* problem.

A time series is *stationary* if its statistical properties, such as its moments, do not change over time. Various studies [15, 16, 17] have found that the time series that model the evolution of metrics considered by the IDIPS server are *piecewise stationary*. As a result, we can partition a time series into *k* intervals and consider the time series to be stationary inside any interval. We identify two issues:

1. How to detect a change of regime or, in other words, how to determine the stationary intervals.
2. How to model time series in a given stationary interval.

We will first address the second issue by considering various modeling alternatives (auto-regressive, moving average, and auto-regressive moving average models. As a second step, we will consider Switching Markov models that generalize the previous models by allowing for various regime changes.

The application of those time series modeling to Internet QoS measurements raises several issues:

- Among the possible time series models considered, one needs to investigate which is the most appropriate choice for QoS measurements. For a given model family, the best meta-parameters need to be chosen. Selection criteria include the amount of training data required to come up with reliable estimates, the actual degree of on-stationarity of the observed time series, space and time requirements during the learning phase and the prediction phase.
- The first objective of QoS measurements modeling is the prediction of the next measurement(s) based don past observations. Past observations may, however, not always be fully available since one would like to reduce as much as possible the amount of active probing. The main issue is to maximize the quality of the predicted measurements while minimizing active probing. In other words, one has to optimize a performance/cost trade-off of some sub-sampling of the actual measurements.

- Parameter estimation is the central task of a learning algorithm with a usual trade-off between the complexity of the model to be estimated and the robustness of the estimates. In this context, maximum likelihood estimation is not necessarily the best choice. Modifying the maximum likelihood can be done either explicitly by adding a regularization term to favor simpler models or, implicitly, by smoothing the maximum likelihood estimates. One open issue is to determine which of those options work best in terms of quality of the estimated models and computing time.
- The presentation focuses so far on the modeling and the prediction of uni-dimensional time series, for instance the delay measure. In our IDIPS context, several related measurements are of keen interest (delay, bandwidth, jitter, etc.). Extension to multivariate time series modeling can be considered. The objective is to explicitly consider the dependence between various measurements rather than modeling them independently.

These time series prediction techniques are complementary to current development in measurement scalability [40].

8.1.2.2. Finding low delay paths

A first example of a co-operation between IDIPS servers is to assume that they all take part in an ICS (Internet Coordinate System) and will therefore infer their coordinates in a suitable metric space. By measuring RTTs with a limited set of neighbor IDIPS servers (say 32 or 64), every IDIPS server will nonetheless have good RTT estimations to all others, at no extra measurement cost.

Moreover, by observing the ICS (e.g., its inaccuracies, its oscillations), we expect to infer other interesting path characteristics. For example, if there is an indirect path between two IDIPS servers, via a third one, that provides lower latency (what is referred to as a TIV, i.e., *Triangular Inequality Violation*), it is known that the ICS will be inaccurate and some coordinate may not stabilize. By observing suitable variables that capture these phenomena, we plan to infer when there are shortcut paths for a given pair of source-destination IDIPS server. This again is useful to trigger a search for these paths and therefore propose a larger set of interesting paths to rank.

We have already identified two sub-problems to be addressed by Machine Learning:

8.1.2.2.1. Shortcut detector

This detector should either predict whether the best path between two IDIPS servers is the direct path between them or an indirect path via some intermediate IDIPS servers (in which case, this is a binary classification problem) or predicts the expected gain of the shortcut (in which case, this is a regression problem) [41].

Input of the shortcut detector: source and destination IDIPS servers, ICS variables.

Output of the shortcut detector: a criterion (or set of combined criteria) telling with high confidence whether or not there exists an indirect path (via other servers) that has a lower RTT than the direct path.

The input can also include the level of (relative/absolute) gain the shortcut path should provide, in which case the output of the learning algorithm predicts whether or not there exists

an indirect path of this level between the source and the destination. Another possibility is to ask the learning algorithm to predict as an output the gain of the shortcut path.

8.1.2.2.2. Relay detector

This detector should predict the intermediate IDIPS servers that are more likely to offer interesting indirect paths to some destination IDIPS server. In this case, the output of the machine learning algorithm is thus a ranking of IDIPS servers according to their probability to offer a shortcut path between the source and destination servers. From a machine learning point of view, this could be considered as a ranking problem.

Input of the relay detector: source and destination IDIPS servers (e.g., for which we know there is an interesting shortcut with high likelihood, see above).

Output of the relay detector: a criterion to narrow the search of intermediate IDIPS servers that are likely to provide interesting shortcuts paths between the source and destination servers.

For both shortcut and relay detectors the learning phase can be centralized and off-line. When models (e.g., the two above-mentioned criteria) are learnt this way, they can be plugged in all IDIPS servers afterwards.

Models can also be tuned on-line in a decentralized way. For example, one could first plug the bootstrap model in all IDIPS servers and then tune their parameters locally by exploiting some online learning strategy.

9. Accountability

9.1. Profiling and Accountability

Accountability has always been a requirement for the architecture of the Internet but at the same time, it has never been satisfactory addressed [52]. In fact, some researchers [13, 50] have observed that a solution has not been achieved due to the elusive definition of Internet accountability itself. Thus, plainly defining *fairness* in terms of *flow rates* and how these flows impact the network has been an impractical way of addressing the issue. The profile-based accountability system proposes to go beyond simple flow rate control, by correlating profiles with subscribers' usage and their impact on the network resources. The definition of profile and accountability is part of this system work. While a profile can be created depending on different measurements, such as flows, packets or time transition/correlation, accountability can be related to different entities such as a user, a host or even an application. [55, 56] offer a both a good list of possible flow measurements and a comprehensive review of machine-learning techniques used in traffic classification. This is an attempt to tackle a complex issue, thus as we further advance in the project and with feedback from discussion and experimentation, adaptation and evolution of the use case and system work will probably be required.

9.1.1. Use Case Description

To tackle the accountability problem there are three basic issues to be addressed that requires proper definition and clarity, i.e. who is accountable, for what and how to measure, and the consequences of failing. Moreover, profiles are defined by both traffic patterns and how the behavior of these patterns affects the network. In turn, traffic patterns are characterized by the subscribers' network packet flows. Information of the network traffic is embedded in the

protocol headers and in the dynamicity of the packet flows, thus traffic characterization can be derived by closely examining and monitoring these two groups of parameters.

9.1.1.1. Who is Accountable, for what and how to Measure

There are several possible answers to the question on who is accountable. Looking at the issue from a microscopic to a macroscopic level one can define responsibility to different entities such as a *socket*, an *application*, a *host* or a *subscriber*. We will focus on the subscriber's definition, which is characterized by all the traffic in a line card to/from a user connected to an access network. The advantage of this definition is that it has a legal or business aspect, since the relationship between the ISP and the subscriber can be framed in a contract through which accountability could be imposed. From a technical perspective and in a practical sense, subscribers or more generically the customer premise equipment (CPE) of a subscriber is assigned an IP address (usually by an ISP PPP- or DHCP-server after authentication from an AAA-server), which is used by the ISP WAN to identify traffic to/from a subscriber. All the traffic from the different devices in the home network goes through the CPE NAT. Thus, seen from the network, a subscriber traffic flow can be identified by the IP address given to the CPE.

The second question, what a subscriber is accountable for and how to measure this accountability is a more complex issue. Here, we expect machine-learning techniques to contribute with a fresh approach to an important problem. There are two approaches for defining profiles. An *expert* can be used to provide a priori discrimination to traffic patterns, which could be used to feed a (semi)-supervised machine-learning algorithm, depending on how *expensive* the a priori discrimination procedure is to producing the labeled data. Alternatively, unsupervised methods could be used to infer profiles from the subscribers' traffic pattern itself. Moreover, profiles should not be limited to the task of discerning subscribers' network behavior, but it should also relate to the impact on the network. In addition to defining profiles, machine-learning techniques shall be used to monitor subscribers by classifying them to a particular profile. It should also indicate the deviation of subscribers from a specific profile.

9.1.1.2. Network Traffic Characterization

As far as profile-based accountability is of concern, traffic flows will be mainly based on network and transport layer protocol information, with the possibility of using link layer header (WAN information, i.e., CPE MAC address). The use of application layer information, through Deep Packet Inspection (DPI) technology is out of scope, due to the required computational resource. Moreover, it is expected that some of this information will be inferred from the machine-learning technology. In addition to the network information at the protocol level (independently of the layer), traffic patterns are characterized by the behavior and dynamics of packets flows in the time domain. Some measurements that might be used to characterize the traffic pattern are number of flows, average packet size, number of packets-in/out, and average inter-packet gap per flow. Moreover, results of the subtask a1 (see Sec. 6), specially referring to adaptive traffic sampling and management will directly contribute to network characterization for profile-based accountability.

9.1.1.3. Profile Modeling

Profiles can be designed and devised in different ways. For example, profiles can categorize *subscribers* or profiles can be modeled to characterize *actions.* Modeling profiles in either way demands different specifications and requirements from the system.

Subscriber profiling assumes that there are different types of users, which demands or uses the Internet in discernable ways, with respect to the resources they require from the network. As a consequence their requirement and impact on the network are distinct. The advantage of this model is that it enables operators to determine new ways to define Fair Usage Policies (FUP), which are in fact the binding contract a user has with the operator.

Action profiling is another way of modeling profile-based accountability system. The assumption is that these profiles will define actions subscribers are undertaking in the network. More specifically, the term *action* means network usage that has distinct properties and traffic load demand. For example if we translate this action to application level, users might be performing different tasks that have distinct network requirements, such as normal web browsing, use of P2P applications or file download. One-way of looking at action profiles (in respect to the subscriber profiles) is that they define the behavior on a restricted set of activity a subscriber undertakes within a certain period of time.

In summary, subscriber profiles try to characterize the *general* subscribers' activity in terms of traffic pattern and network demand. While action profiles characterize *specific* actions that a user undertake for a restrict time period (see Fig. 9.1). Profile-based accountability ultimately refers to the subscribers' profile, while *action* profiles have been introduced as a means to achieve this end goal. A challenge then is to infer a subscribers profile from a sequence of action profiles, which will characterize a user and his/her demand on the network.

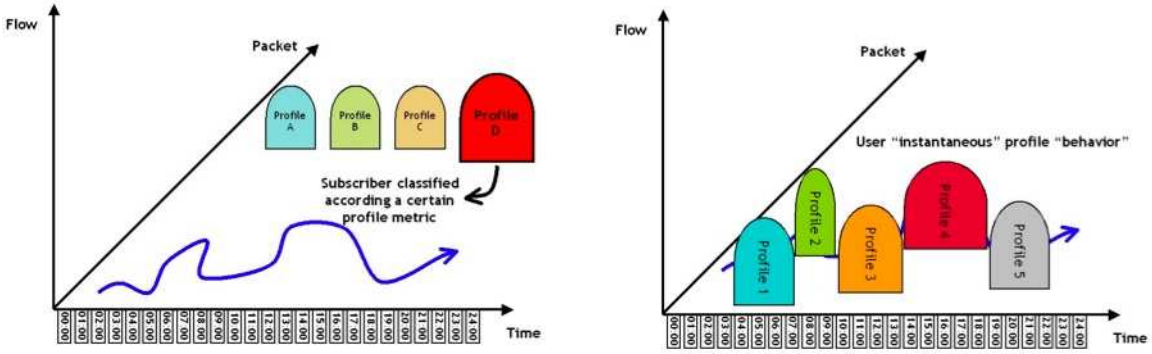

Figure 9.1: Schematic view of *subscribers* (left) and *action* (right) profiles.

9.2. ECODE Architecture Cross-Reference

9.2.1. System Architecture

The general functional diagram of the ECODE framework shows the distributed Machine Learning Engine and its relation towards the traditional elements of an IP network, i.e., the routing and forwarding engines (see Fig. 5.1). While the routing engine is a fundamental element, profile-based accountability is not expected to relate to this function. It will interact with the Forwarding Engine, both at the data collection and drive decision back to this engine. The profile-based accountability functional modules are divided in four blocks, where the machine-learning part is shown as the "classify" block in Fig. 10.1 (see Sec. 10). The system will operate in two parts, one subsystem, i.e., learning, defining the profiles. At the same time, a second subsystem will be responsible for classification. The two first functional blocks composed the learning domain, while the classification domain is represented by the complete loop depicted in the functional diagram. The classification subsystem includes decision on how the system should react upon classification. If a subscriber is found to be breaching his/her profile, the system might decide to execute measures to bring the user to his/her pre-

defined profile. For example, it can decrease the available bandwidth or block flows to that subscriber.

9.2.2. Network Architecture

As part of the networking domain, technical objective (b3)-profile-based accountability is closely related to the objective (a1)-adaptive traffic sampling and management. The first step for profile-based accountability requires data sampling and pre-processing, which consists in getting the raw data and perform feature selection and extraction. This is part of the objectives of adaptive traffic sampling and management. Fig. 9.2 shows the architectural diagram relating case a1 and case b3. Results of the adaptive traffic sampling and management task will be directly used as part of the solution for the profile-based accountability challengers.

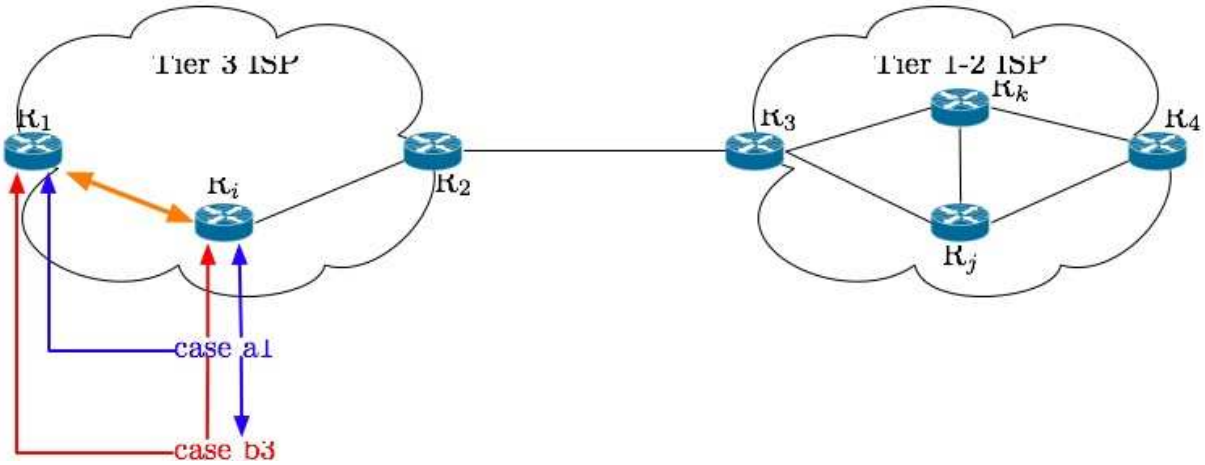

Figure 9.2: Architectural diagram showing case (b3 – in red) profile-based accountability and its relation to case (a1 – in blue) adaptive traffic sampling and management

Fig. 9.3 shows an ideal network configuration including the profile-based accountability (PBA)-engine. While the PBA-engine resides on the access network, the engine would be capable of receiving network information from other elements. In this way, a more encompassing inference of the subscribers' profiles in respect to the network resource allocation can be achieved (more detail in deliverable D3.1 - *Detail Experimental Plan and Scenarios*). Optimally, the PBA-engine will operate in a distributed, real-time mode, and learning is distributed over the access nodes where the engine resides.

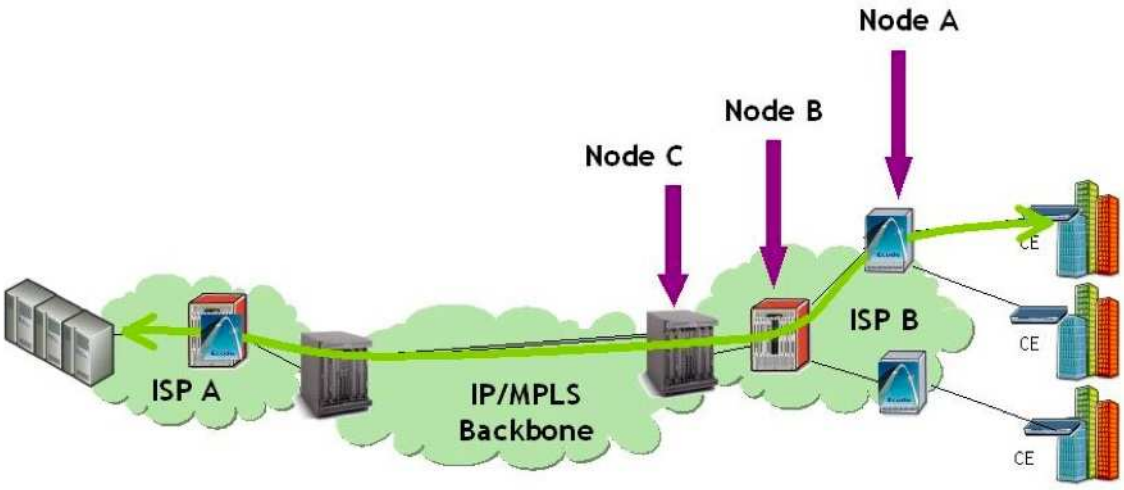

Figure 9.3: Diagram showing the profile-based accountability module distributed over the network elements

9.3. Interaction with Learning Module

Many studies have already track the use of machine learning for traffic flow identification [51, 56]. Some of these work focus on application identification [49, 54, 55], while others on discovering underlines rules that governs the communication over a network [53]. The aim of profile-based accountability goes beyond traffic classification. It should infer the demand subscribers are requesting from the network, so that the network resources can be *fairly* allocated and accountability properly imposed respecting the contract subscribers have with their operator. Machine learning is expected to strongly contribute to this task. The architectural specification of the desired system, in terms of interaction with the routing/forwarding system and inputs and outputs to the machine-learning algorithms are described in more detail in the next subsections.

9.3.1 Type of Routing/Forwarding Engine

The task of the machine-learning algorithm is two-folded, first to define the profiles and secondly to classify (with a certain metric) traffic usage according to the profiles. The first problem requires that the system be *distributed*, such that the data set encompasses all possible subscribers' traffic. The broader the dataset is implies in a richer, i.e. more general, input for the learning systems. The PBA-engine would require exchanging information to achieve the distributed scheme, thus each system would receive *inputs* and provide *outputs* to other nodes. The second problem, i.e. classification can be achieved in a decentralized manner, since the system is already distributed for the first problem. Classification decision can take advantage of the resources and capabilities of all the distributed PBA-Engines.

9.3.2. Inputs to the Machine Learning Module

The inputs for the machine-learning module are deduced from measurements that characterize the subscribers network traffic (see Sec. 9.1.1.2). Those measurements describe the information over connectivity and dynamicity of the traffic, capturing the behavior and activities of packets flows. However, more importantly the inputs should capture information of the subscribers network demand, and the resources and availability of the network as a whole. So qualitatively, the system has to be able to yield profiles, which encompass information on both the load, the subscribers is requesting from the network and at the same time indicate the present network load. Quantitatively, it is clear that inputs can assume different types such as *bit, integer, real or words from some alphabet*. Moreover, time is a fundamental element. These parameters varies in time, and so does the fundamental aspect that we try to model, i.e. the profiles that reflects the different subscribers' traffic pattern behavior.

9.3.3 Outputs to the Machine Learning Module

There are two major tasks expected from the learning system, which will define two types of output, one for the *profile learning* stage and the second for the *profile prediction* stage, or in our case the *profile classification* stage. Profile learning refers to the process of defining or categorizing profiles. In this stage the output from the input variables can determine the partition of the dataset into different clusters, with each cluster characterizing a profile. Each profile indicates specific network traffic behavior associated with the usage and demand on the network resources. Ideally, the learning algorithm would be an online, distributed semi-supervised or unsupervised system, where PBA-engine would reside in access nodes within

an ISP. Profile prediction refers to the classification process of users according to the learned profiles. The output of this stage should be a classification decision to one of the possible classes. In addition, a proximity metric, indicated by a numerical value, should reflect the distance subscribers are in respect to the possible profiles. This information could be used to quantify to what level subscribers are breaching their network accountability.

9.3.4 Interaction with the Routing/Forwarding Engine

At this stage, the results of the process described for the PBA-engine is expected to have at most passive influence on the routing system, for example triggering protocol flags such as in Explicit Congestion Notification (ECN). Most likely, it will act on the forwarding system, by either trying to bring the subscriber to the assigned profile or by punishing the subscriber, for example by delaying or just dropping packets. Moreover, what is required from the system is a continuous monitoring of subscribers' profile information.

10. Analysis Grid

This section introduces an analysis grid of the network and system architecture (as detailed in Sec. 5) against the use cases described in Sec. 6, 7, 8, and 9. The objective of this analysis grid is to determine the architectural items that will be further investigated throughout the project lifetime and reach a common/reference architectural framework. This analysis will consist in a *functional analysis* and a *performance analysis*.

10.1. Functional Analysis

To determine the characteristics of the proposed experimental architecture against the closed control loop (detection, processing, decision, and execution) functional analysis will be performed using a pre-defined set of functional criteria. Assessing the degree of conformance of the experimental architecture and its capabilities with respect to the closed control loop depicted in Fig. 10.1 is the main objective of this analysis. Note that the functional blocks depicted in this figure may be the object of further decomposition.

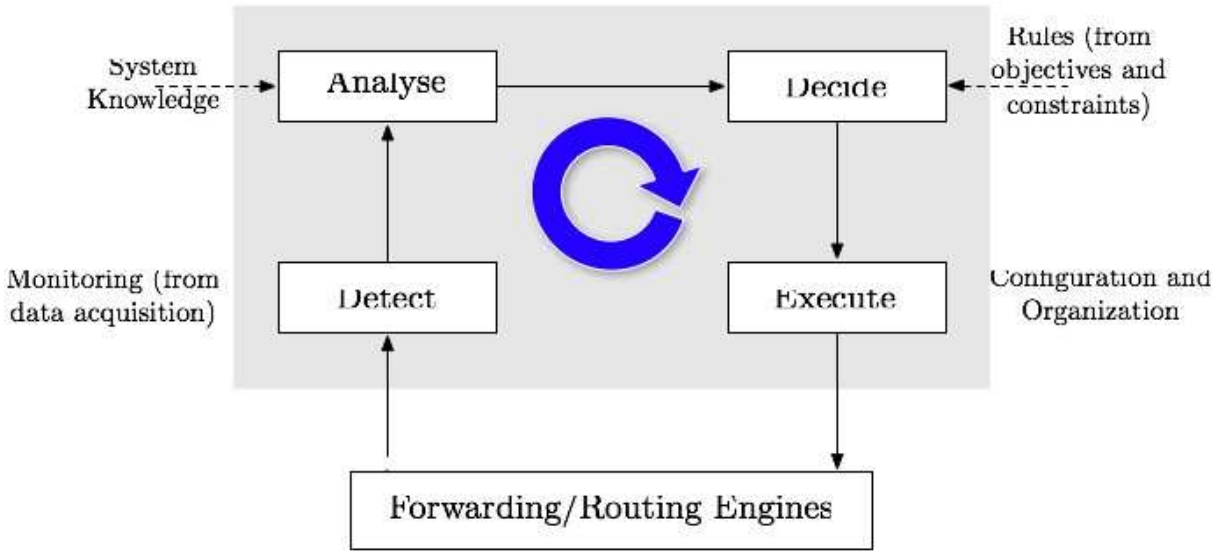

Figure 10.1: Closed control loop

Indeed, as more than one design may be proposed, functional analysis is a powerful tool to determine the support level of each functionality by the experimental architecture depicted in

Section 5. Our functional analysis study will also provide the exact implications in terms of correlation/dependency and constraints the supported feature imply on other features.

10.1.1 Criteria

Functional analysis criteria are provided to evaluate and assess the level of conformance of a given design. They translate the fundamental networking rules (also referred to as network design principles) that allow the identification of the forwarding, routing and machine learning needs when assessed against the needs resulting from the design of a closed control loop.

10.1.2. Methodology

Different attempts to define software quality as a complex concept that can be decomposed in more detailed characteristics have been presented since 1970s, e.g. McCall et al, 1977. The key idea was to enable evaluation of quality through the evaluation of more detailed characteristics that are supposed to be easy to measure or assess. Standardized quality models based on this idea, e.g. ISO 9126 were subsequently developed based on this concept.

McCall's Quality Factors (McCall, Richards, Walters; 1977) provides a direction towards measuring software quality. This model that aims at system developers during the software development process identifies 3 areas of software work:

- *Product Operation*: refers to the product's ability to be quickly understood, efficiently operated and capable of providing the results required by the user
- *Production Revision*: relates to error correction and system adaptation
- *Product Transition*: distributed processing, rapid change in hardware

McCall expressed software quality in terms of ***11 measurable quality factors***. These 11 quality factors focus on three important aspects of a software product. Some of the quality factors are responsible for successful product operation; some of the quality factors are responsible for successful product revision and some are responsible for successful product transition. The 11 quality factors are grouped with respect to these ***3 areas of work***:

Product operation:

- *Correctness*: the extent to which a program satisfies its specification and fulfills customer's mission objective.
- *Reliability*: the extent to which a program can be expected to perform its intended function with required precision.
- *Usability*: the effort required to learn, operate/use, prepare input, and interpret output of a program.
- *Integrity*: the extent to which access to software or data by unauthorized persons can be controlled.
- *Efficiency*: the amount of computing resources and code required by a program to perform its functions.

Product revision:

- *Flexibility*: the effort required to modify an operational program.
- *Testability*: the effort required to test a program to ensure that it performs it's intended functions.
- *Maintainability*: the effort required to locate and fix an error in a program.

Product transition:

- *Portability*: the effort required to transfer the program from one hardware and/or software system environment to another.
- *Reusability*: the extent to which a program or parts of a program can be reused in another application.
- *Interoperability*: the effort required to couple one system to another.

All the quality factors mentioned above depends upon set of 22 metrics; and the dependency can be given by the following formula:

$$F_q = c_1 m_1 + c_2 m_2 + \ldots + c_n m_n$$

where,

F_q is the software Quality Factor;
c_n are the regression coefficient such $c_1 + c_2 + \ldots + c_n = 1.0$
m_n are the metrics

Most of these metrics (listed here below) can be measured subjectively. These metrics can be used in the form of a check list. The grading scheme for each of the 22 metrics is between 0 (low) and 10 (high). The value of the regression co-efficient is dependent upon the products and the weight given for that particular metrics.

The 22 metrics (or criteria) are defined as follows:

- *Auditability*: the ease with which conformance to standards can be checked.
- *Accuracy*: the precision of computation and control.
- *Communication Commonality*: the degree to which standard interfaces, protocols and bandwidth are used.
- *Completeness*: the degree to which full implementation of required function has been achieved.
- *Complexity*: the degree to which the program is complex (complexity metric)
- *Conciseness*: the compactness of the program in terms of lines of codes.
- *Consistency*: the use of uniform design and documentation technique throughout the software development project.
- *Data Commonality*: the use of standard data structures and types throughout the program.
- *Error Tolerance*: the damage that occurs when the program encounters an error
- *Execution Efficiency*: the run time performance of a program.
- *Expandability*: the degree to which architectural, data or procedural data can be extended.
- *Generality*: the breadth of potential application of program components.
- *Hardware Independence*: the degree to which the software is decoupled from the hardware on which it operates.
- *Instrumentation*: the degree to which the program monitors it's own operation and identifies errors that do occur.
- *Modularity*: the functional independence of program components.
- *Operability*: the ease of operation of a program.
- *Security*: the availability of mechanisms that control or protect programs and data.
- *Self-Documentation*: the degree to which the source code provides meaningful documentation.

- *Simplicity*: the degree to which a program can be understood without difficulty.
- *Software System Independence*: the degree to which the program is independent of nonstandard programming language features, operating system characteristics, and other environmental constraints.
- *Traceability*: the ability to trace a design representation or actual program component back to requirements.
- *Training*: the degree to which the software assists in enabling new users to apply the system.

Each of these metrics contributes to one or more quality factors, as shown in Table 1 (Source: Roger S. Pressman, "Software Engineering: A Practitioner's Approach" (European Adaptation), Ch. 19, Fifth Edition, 2000). Note that the weight given to each metric depends on individual products and concerns.

Table 1: Relationships between McCall's quality factors and metrics

Quality factor / Software quality metric	Correctness	Reliability	Efficiency	Integrity	Maintainability	Flexibility	Testability	Portability	Reusability	Interoperability	Usability
Auditability				X			X				
Accuracy		X									
Communication commonality										X	
Completeness	X										
Conciseness			X		X	X					
Consistency	X	X			X	X					
Data commonality										X	
Error tolerance		X									
Execution efficiency			X								
Expandability						X					
Generality						X		X	X	X	
Hardware independence								X	X		
Instrumentation				X	X		X				
Modularity		X			X	X	X	X	X	X	
Operability			X								X
Security				X							
Self-documentation					X	X	X	X	X		
Simplicity		X			X	X	X				
Software system independence								X	X		
Traceability	X										
Training											X

10.2. Performance Analysis

10.2.1. Performance Analysis

Performance analysis will be conducted by considering the following events classes:

- *External events* resulting in deviation(s) from initial performance objectives (resource-oriented, traffic-oriented)
- *Internal events* resulting in deviation(s) from initial performance objectives (resource-oriented, traffic-oriented)

These events will be used to experiment of the Machine Learning Engine and determine the suitability and sustainability of the proposed architecture under various running conditions and constraints.

The definition of *i*) performance objectives and *ii*) conditions and constraints are use case dependent and they will be detailed in the next revision of this document.

10.2. Sensitivity Analysis

10.2.1 Overview

Large-scale experimentation models (simulations, emulations, etc.) often involve a large number of parameters, making it prohibitive to run more than a small fraction of all potentially relevant cases. In this context, sensitivity analysis attempts to identify how responsive the results of an experimental model are to changes in its parameters: this is an important tool for achieving confidence in experimentation and making its results credible. The general goal of Sensitivity Analysis is to characterize, qualitatively or quantitatively, what impact on a system a particular variable will have if it differs from what was previously assumed. In other words, by using Sensitivity Analysis, the analyst can determine how changes in one or several parameters will impact the target variable.

Sensitivity analysis quantifies the dependence of system behavior on the parameters that affect the modeled process and in particular its dynamics. It is used to determine how sensitive a model is to i) changes in the numerical value of the model parameters: parameter sensitivity analysis aims at determining the uncertainty associated with the numerical values of model parameters (resulting thus in parameter estimation but also prediction). In this case, sensitivity analysis is used to increase the confidence in the model and its predictions, by providing an understanding of how the model responds to changes in its parameters, and ii) changes in the structure of the model.

In the present case, we will perform sensitivity analysis on

- Detection/identification time, and rate:
    - on traffic variations (for forwarding dependent cases)
    - on routing information/ topology variations (for routing dependent cases)
    - etc.
- Execution (re-configuration/re-organization) time, and scope: on decision sequence(s) so as to determine/identify conditions for
    - Oscillations effects leading to action/reaction chains
    - Coupling effects leading to amplification/annealing chains

10.2.2 Sensitivity Analysis Methods

Perturbation Theory based methods: study a set of models which are different from a nominal model by some small terms. Sensitivity Analysis is closely linked with Perturbation Theory. Perturbation Theory comprises mathematical methods that are used to find an approximate solution to a problem which cannot be solved exactly, by starting from the exact solution of a related problem. Perturbation Theory can be applied if the problem under study can be formulated by adding a "small" term to the mathematical description of the exactly solvable problem. Thus, Perturbation theory can be viewed as a tool for Sensitivity Analysis. Furthermore, it can be classified as an analytic tool for the Sensitivity Analysis. The main types of mathematical models for perturbation methods are.

- Linear Algebraic Systems
- Non-linear Algebraic Systems
- Mathematical programming

The advantage of analytic Perturbation Theory based methods is that these methods are based on a solid theoretical ground. The disadvantage of the analytic methods is that typically the deviations of parameters need to be small and a good knowledge of the system's structure/dynamics is required.

The other class of tools for Sensitivity Analysis is ***Sampling based methods***. Analytical methods require a good knowledge of the system and might require tedious calculations. The sampling based methods are designed to overcome these disadvantages. Sampling methods are particularly well suited to withstand the changing one-factor-at-a-time (OAT) paradox.

- *FAST (Fourier Amplitude Sensitivity Test)*: method which deals with static models. 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. FAST is considered to be one of the most efficient methods in sensitivity analysis [ref]. Among its advantages are: fast implementation, deals with non-monotonic models, allows arbitrary large variations in input parameters, and does not require the knowledge of the mathematical model. The latter two features are in particular positively distinguishing FAST from analytical methods. However, FAST suffers from computational complexity for a large number of inputs. Moreover, the basic FAST method can only be applied to static models with independent parameters. As, in many cases the parameters are correlated with one another, extended FAST (EFAST) has been proposed for models with correlated parameters i.e. EFAST can address higher order interactions (see Saltelli98).
- *Path based Sensitivity Analysis (of Markov Chains)* for dynamical systems: the key idea in path-based sensitivity analysis of Markov chains is the observation that a sufficiently long sample path contains enough random deviations to test the system sensitivity.

Sampling based methods do not require access to model equations or even the model code. These methods require running a series of experiments. Experiments can be either real-life or numerical. The disadvantage of the sampling based methods is that the number of experiments required can be very large.

11. Conclusion

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

Our first goal, in this deliverable, is to deeply discuss the ECODE architecture. We provide a two levels architecture: the system and the network architecture. On one hand, the system architecture deals with how the ECODE project improves the current router architecture. Mainly, the improvement resides in the addition of the Machine Learning Engine. This engine will be in relation with the already existing routing and forwarding engines. We also introduce a fourth engine, the Monitoring Engine that aims at collecting path performance information. On the other hand, the network architecture aims at explaining where the ECODE contributions are located in the network. We explain that there exist various types of routers, i.e., internal, edge, and access routers. Depending on the use case, the contributions will be located in one (or several) of these routers.

Next, we provide a general insight in machine learning techniques. We discuss the learning techniques: supervised (output prediction for a novel input after learning on a training dataset), unsupervised (learning useful structure without any kind of information beyond the raw data and grouping principles), on-line (the training data is provided to the learning algorithm as a batch process), and distributed learning (distribution of the prediction or the data to analyze).

ECODE is an experimentally driven research project built on use cases. In this deliverable, we deeply discuss all use cases considered in the project. We also explain how the use case will make use of machine learning techniques and how they will be integrated in the ECODE architecture, at the system and network levels. Subsequent implementation of these use cases is expected to provide precious input on suitability of the proposed architecture as well as sufficient results to assess added value of machine learning.

Thus, in the current state of the project, we cannot claim that the network and system architecture provided in this document is the final one. Instead, during the whole project duration, the architecture feasibility will be evaluated and, necessarily, adapted so that, by the end of the project, we will have converged towards a workable architecture.

References

[1] E. Rose, A. Viswanathan, R. Callon. *Multiprotocol Label Switching Architecture*. Internet Engineering Task Force. RFC 3031. January 2001

[2] T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. Internet Engineering Task Force. RFC 2246. January 1999.

[3] T. Li, G. Huston. *BGP Stability Improvements*. Internet Engineering Task Force. Internet Draft (Work in Progress) draft-li-bgp-stability-01. June 2007

[4] W. Willlinger, J. Doyle. *Robustness and the Internet: Design and Evolution*. Technical Report Caltech. March 2002. See http://netlab.caltech.edu/pub/papers/part1_vers4.pdf

[5] D. Clark, C. Partridge, J.C. Raming, J.T. Wroclawski. *A Knowlegde Plane for the Internet*. In Proc. ACM SIGCOMM. August 2003

[6] L. Breinam, J. Friedman, R. A. Olshen, P. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[7] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer Series in Statistics, 2001.

[8] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] O. Chapelle, B. Schölkopf, A. Zien (Eds.). *Semi-Supervised Learning*. MIT Press, 2006.

[10] A. Blum, T. Mitchell. *Combining Labeled and Unlabeled Data With Co-Training*. In Proc. 11th Annual Conference on Computer Learning Theory. 1998.

[11] N. Lawrence, M. Seeger, R.Herbrich. *Fast Sparse Gaussian Process Methods: The Informative Vector Machine, Advances in Neural Information Processing Systems*. 15, MIT Press, 2003.

[12] J Moy. *OSPF Version 2*. Internet Engineering Task Force. RFC 2328. April 1998

[13] B. Briscoe. Flow Rate Fairness: Dismantling a Religion. In ACM SIGCOMM Computer Communication Review, 37(2), pp 63-74. 2007

[14] M. Allman, V. Paxson, W. Stevens. *TCP Congestion Control*. Internet Engineering Task Force. RFC 2581. April 1999.

[15] Y. Zhang, N. Duffield. *On The Constancy of Internet Path Properties*. In Proc. ACM SIGCOMM Workshop on Internet Measurement. 2001.

[16] M. Yang, J. Ru, H. Chen, A. Bashi, N.S.V. Rao. *Predicting Internet End-to-End Delay: A Statistical Study*. In Annual Review of Communications, Vol. 58. 2005

[17] V. Bui, W. Zhu, A. Pescape, A. Botta. *Long Horizon End-to-End Delay Forecasts: A Multi-Step-Ahead Hybrid Approach*. In Proc. IEE Symposium on Computers and Communications (ISCC). 2007.

[18] Y. Rekhter, T. Li, S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force. RFC 4271. January 2006.

[19] T. G. Griffin, B. J. Premore. *An Experimental Analysis of BGP Convergence Time*. In Proc. 9th International Conference on Network Protocols (ICNP). November 2001.

[20] R. Oliveira, B. Zhang, R. Izhak-Ratzin. *Quantifying Path Exploration in the Internet*. In Proc. ACM Internet Measurement Conference (IMC). October 2006.

[21] T.Mitchell. *Machine Learning*. McGraw-Hill, 1997

[22] A.K Jain,R.P.W Duin, J. Mao. *Statistical Pattern Recognition: A Review* In IEEE Transactions On Pattern

Analysis and Machine Intelligence, 22(1). January 2000

[23] E.J Hartman, J.D Keeler, J.M Kowalski. *Layered Neural Networks with Gaussian Hidden Units as Universal Approximations.* In Neural Computation, 2(2). April 1990.

[24] M. Sahami, S. Dumais, D. Heckerman, E. Horvitz. *A Bayesian Approach to Filtering Junk E-mail*. In Proc. AAAI'98 Workshop on Learning for Text Categorization. July 1998

[25] R.O. Duda, P.E. Hart, D.H. Stork. *Pattern Classification* (2nd ed.), Wiley Interscience, 2000

[26] B. E. Boser, I. M. Guyon, and V. N. Vapnik. *A Training Algorithm for Optimal Margin Classifiers*. In Proc. 5th Annual ACM Workshop on Computational Learning Theory (COLT). July 1992.

[27] J. Ghent and J. McDonald. *Facial Expression Classification Using a One-Against-All Support Vector Machine*. In Proc. Irish Machine Vision and Image Processing Conference. August 2005.

[28] B. Schölkopf, C. J. C. Burges, V. N. Vapnik.. *Extracting Support data for a Given Task.* In Proc. 1st International Conference on Knowledge Discovery Data Mining (KDD). August 1995

[29] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, A. Y. Wu. *An Efficient K-means Clustering Algorithm: Analysis and Implementation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7). July 2002.

[30] M. Bagnulo. *Default Locator-Pair Selection Algorithm for the Shim6 Protocol*. Internet Engineering Task Force, Internet Draft (Work in Progress) draft-ietf-shim6-locator-pair-selection-02. July 2007

[31] Akamai. *Web Application Acceleration and Performance Management, Streaming Media Services, and Content Delivery*. See http://www.akamai.com

[32] A. Matsumoto, T. Fuijisaki, R. Hiromi, K. Kanayama. *Problem Statement of Default Address Selection in Multi-Prefix Environment: Operational Issues of RFC 3484 Default Rules*. Internet Engineering Task Force, Internet Draft (Work In Progress), draft-ietf-v6ops-addr-select-ps-05. April 2008

[33] D. Farinacci. *Locator/ID Separation Protocol (LISP)*. Internet Engineering Task Force, Internet Draft (Work in Progress) draft-farinacci-lisp-05. November 2007

[34] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver. *The Spread of the Sapphire/Slammer Worm*. CAIDA Technical Report. 2003

[35] S. Staniford, D. Moore, V. Paxson, N. Weaver. *The Top Speed of Flash Worms*. In Proc. ACM Workshop on Rapid Malcode (WORM). October 2004.

[36] D. Moore, C. Shannon. *The Spread of the Witty Worm*. In IEEE Security and Privacy, 2(4). July/August 2004.

[37] O. Bonaventure, D. Saucez, B. Donnet. *The Case for an Informed Path Selection Service*. Internet Engineering Task Force, Internet Draft (Work in Progress) draft-bonaventure-informed-path-selection-00. February 2008

[38] D. Saucez, B. Donnet, O. Bonaventure. *IDIPS: ISP-Driven Informed Path Selection*. Internet Engineering Task Force, Internet Draft (Work in Progress) draft-saucez-idips-01. November 2008

[39] F. Dabek, R. Cox, K. Kasshoek, R. Morris. *Vivaldi, a Decentralized Network Coordinated System*. In Proc. ACM SIGCOMM. August 2004.

[40] D. Saucez, B. Donnet, O. Bonaventure. *On the Impact of Clustering on Measurement Reduction.* Under Submission. 2008

[41] Y. Liao, M. A. Kaafar, B. Gueye, F. Cantin, P. Geurts, G. Leduc. *Detecting Triangle Inequality Violations in Internet Coordinate Systems by Supervised Learning*. Under Submission. 2008.

[42] D. Katz, D. Ward. *Bidirectional Forwarding Detection*. Internet Engineering Task Force. Internet Draft (Work in Progress) draft-ietf-bfd-base-08. March 2008

[43] M. Goyal, G. Choudhury, A. Shaikh, K. Trivedi, H. Hosseini. *LSA Correlation to Schedule Routing Table Calculations*. Internet Engineering Task Force. Internet Draft (Work in Progress) draft-goyal-ospf-lsacorr-00. October 2008

[44] P. Francois, O. Bonaventure, M. Shand, S. Bryant, S. Previdi. *Loop-free convergence using oFIB*. Internet Engineering Task Force. Internet Draft (Work in Progress) draft-ietf-rtgwg-ordered-fib-02. February 2008.

[45] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, Y. Chen. *Accurate and Efficient Traffic Monitoring Using Adaptive Non-linear Sampling Method*. In Proc. IEEE INFOCOM. April 2008.

[46] H. Guo. *Algorithm Selection for Sorting and Probabilistic Inference: A Machine Learning-Based Approach*. Ph. D. Thesis. Kansas State University. 2003.

[47] A. Atlas, A. Zinin. *Basic Specification for IP Fast Reroute: Loop-Free Alternates*. Internet Engineering Task Force. RFC 5286. September 2008.

[48] S. Bae, T. R. Henderson. *Traffic Engineering with OSPF Multi-Topology Routing*. In Proc. IEEE Military Communications Conference (MILCOM). October 2007.

[49] L. Bernaille, R. Teixeira, K. Salamatian. *Early Application Identification*. In Proc. ACM CoNEXT. December 2006

[50] B. Briscoe. *A Fairer, Faster Internet Protocol*. In IEEE Spectrum, 45(12), pp. 42-47. December 2008

[51] A. Hijazi, H. Inoue, A. Matrawy, P. C. Van Oorschot, A. Somayaji. *Discovering Packet Structure Through Lightweight Hierarchical Clustering*. In. Proc. IEEE International Conference on Communications (ICC). May 2008.

[52] D. Clark. *The Design Philosophy of the DARPA Internet Protocols*. In Proc. ACM SIGCOMM. August 1988.

[53] S. Kandula, R. Chandra, D. Katabi. *What's Going On? Learning Communication Rules in Edge Networks*. In Proc. ACM SIGCOMM. August 2008.

[54] W. Li, A. W. Moore. *A Machine Learning Approach for Efficient Traffic Classification*. In Proc. IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). October 2007

[55] A. McGregor, M. Hall, P. Lorier, J. Brunskill. *Flow Clustering Using Machine Learning Techniques*. Proc. Passive and Active Measurement Workshop (PAM). April 2004.

[56] T. T. T. Nguyen, G. Armitage. *A Survey of Techniques for Internet Traffic Classification using Machine Learning*. In IEEE Communications Surveys and Tutorials, 10(4), pp. 56-76. December 2008.

[57] Y. Labit, P.Owezarski, N. Larrieu. *Evaluation of Active Measurement Tools for Bandwidth Estimation in Real Environment*. In Proc. IEEE/IFIP Workshop on End-to-End Monitoring (E2EMON). May 2005

[58] M. Jain, C. Dovrolis. *End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput*. In IEEE/ACM Transactions on Networking, 11(4), pp. 537-549. August 2003.

[59] R. Prasad, C. Dovrolis, M. Murray, K. Claffy. *Bandwidth Estimation: Metrics, Measurement Techniques, and Tools*. In IEEE Network, 17(6). November/December 2003.

[60] P. Borgnat, N. Larrieu, P. Owezarski, P. Abry, J. Aussibal, L. Gallon, G. Dewaele, K. Boudaoud, L. Bernaille, A. Scherrer, Y. Zhang, and Y. Labit. *Détection d'Attaques de Déni de Service par un Modèle Non Gaussien Multirésolution*. In Proc. Colloque Francophone sur l'Ingénierie des Protocoles (CFIP). October 2006.

[61] A. Scherrer, N. Larrieu, P. Owezarski, P. Borgnat, P. Abry. *Non-Gaussian and Long Memory Statistical Characterizations for Internet Traffic with Anomalies*. In IEEE Transactions on Dependable and Secure

Computing, 4(1), pp. 56-70. January 2007.

[62] P. Borgnat, P. Abry, G. Dewaele, A. Scherrer, N. Larrieu, P. Owezarski, Y. Labit, L. Gallon, J. Aussibal. *Une Caractérisation Non Gaussienne et à Longue Mémoire du Trafic Internet et de ses Anomalies: Validation Expérimentale et Application à la Détection d'Attaque de DDoS*. In Annales des Télécommunications, 62(11/12), pp. 1401-1428. November/December 2007.

[63] M. Taqqu, V. Teverosky, W. Willinger. *Is network traffic self-similar or multifractal?* In Fractals, 5 (1), pp. 63–73. 1997

[64] A. Lakhina, M. Crovella, C. Diot. *Mining Anomalies Using Traffic Feature Distributions*. In ACM SIGCOM Computer Communication Review, 35(4), pp. 217-228. October 2005.

[65] Y. Labit, P. Owezarski. *An Entropy Based Analysis Method of Network Delays for a Discriminating DoS Detection*. In Proc. Workshop on Monitoring, Attack Detection and Mitigation (MonAM). November 2007.

[66] J. Cleary, S. Donnely, I. Graham, A. McGregor, M. Pearson. *Design Principles for Accurate Passive Measurement*. In Proc. Passive and Active Measurement Workshop (PAM). April 2000.

[67] J. Chandrashekar, Z. Duan, Z.-L. Zhang, J. Krasky. *Limiting Path Exploration in BGP*. In Proc. IEEE INFOCOM. March. 2005

[68] D. Pei, M. Azuma, D. Massey, L. Zhang. *BGP-RCN: Improving Convergence Through Root Cause Analysis*. In Computer Networks, 48(2), pp. 175-194. June 2005

[69] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, S. Tarantola. *Global Sensitivity Analysis: The Primer*. John Wiley and Sons. January 2008.