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

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D3.4

"Design and Implementation of Technical Objective 2 (Path availability estimation, network recovery and resiliency techniques, and profile-based accountability)"

Project description
Project acronym: **ECODE** Project full title: **Experimental Cognitive Distributed Engine** Grant Agreement no.: **223936**
Document Properties
Number: **FP7-ICT-2007-2-1.6-223936-D3.4** Title: **Design and Implementation of Technical Objective 2** Responsible: **Philippe Owezarski** Editor(s): **Guy Leduc** Contributor(s): **François Cantin, Didier Colle, Goutam Das, Benoit Donnet, Pierre Geurts, Guy Leduc, Yongjun Liao, Juan Narino, Dimitri Papadimitriou, Damien Saucez, Wouter Tavernier, Ing-Jyh Tsang, Werner Van Leekwijck** Dissemination level: **Public (PU)** Date of preparation: **29th Oct. 2009** **Version: 1.0**

D3.4 - Executive Summary

This deliverable is part of WP3 (Cognitive network and system experimentation). The feasibility, benefits and applicability of introducing a cognitive engine in the ECODE architecture are experimented using a number of use cases covering different problem areas identified as Internet architectural and design challenges.

In particular we address techniques for path availability estimation, for improving network recovery and resiliency, and profile-based accountability. The following three use cases are studied in-depth:

- Path availability: the goal is to design and experiment techniques, embedded in our so-called IDIPS server, to rank internet paths based on their characteristics, such as delays, and available throughput. To this end, on-line machine learning techniques are used to estimate future path characteristics based on past measurements, in order to reduce the future measurement load. Machine learning techniques are also used to Improve Internet Coordinate Systems to better estimate delays between nodes with scalable active delay measurements.

- Network recovery and resiliency: the goal is to reduce the recovery time and increase the recoverability, when routes are updated in a network. To this end, we design and experiment machine learning techniques to minimize packet loss during rerouting and infer SRLGs (Shared Risk Link Groups, i.e. sets of links that can fail all together) in networks, and anticipate their occurrence very early.

- Profile-based accountability: the goal is to 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. To this end, we design and experiment machine learning techniques to cluster customer profiles based on their network resource demands, and to classify customers according to the learned profiles.

Based on this experience, we describe how we currently view the mapping of these use cases onto the anticipated ECODE architecture designed earlier. In particular we show how the functionalities of all the use cases are split among the Routing Engine (RE), Forwarding Engine (FE) and Machine Learning Engine (MLE) of the architecture, and which message exchanges are necessary among these components.

From a research perspective, the main results of this deliverable can be summarized as follows:

- Measuring a path performance according to one or several metrics, such as delay or bandwidth, is becoming more and more popular for applications. However, constantly probing the network is not suitable. To make measurements more scalable, the notion of clustering has emerged. We demonstrate in [2] that clustering can limit the measurement overhead in such a context without loosing too much accuracy. The paper shows that measurement reduction can be observed when vantage points collaborate and use clustering to estimate path performance. The paper also shows how effective is the overhead reduction and what is the impact in

term of measurement accuracy. In addition, in [1], we demonstrate how to reduce the path performance metric measurements by applying machine learning techniques. We express the problem as a time series regression problem and propose several adaptative models for predicting those metrics. Based on a large dataset collected through the PlanetLab testbed, we evaluate our models and demonstrate their efficiency.

- When network nodes run an Internet Coordinate System (ICS), the measured distances (typically delays) between some of pairs of nodes are embedded into a metric space (or coordinate system). When the coordinates of the nodes are known, the prediction of the distances (delays) between two nodes is a straightforward application of a distance function where no explicit communication between them is required. This significantly reduces the overhead of active probing and largely improves the efficiency of the network. We have used Machine Learning techniques to improve the accuracy of an ICS [3, 4, 5]. We have derived automatically a criterion that can be used by nodes to better select their neighbours in the ICS and thereby reduce the impact of Triangular Inequality Violations (TIVs), which are detrimental to an ICS. The knowledge of estimated delays between nodes can also be useful to select better paths for real-time applications. We have also proposed some methods that rely on the nodes running an ICS to detect useful routing shortcuts in networks [6].

- Upon failures the IP router must update its routing and forwarding tables, which may take some time and lead to packet losses. We have evaluated traffic-informed router update models using strategies with either fixed or optimized variable sizes for the update-distribution batches. The resulting models were implemented in a simulation environment and were quantitatively characterized. Depending on the context, we showed that the formulated strategies can result into a decrease of packet loss of 10 to 80 percent using small router process quantums [7]. Because a traffic-informed router update models can only be effective if the traffic statistics that are being used are accurate, the work is being extended such as to predict the short-term trend of aggregated network traffic flows. Currently, experiments are being carried out using Self-Organizing Maps (SOMs), Feed-Forward Neural Networks (FFNN) and Support Vector Regression (SVR) techniques.

- IP routers exchange link state advertisements (LSAs) to know about network failures and to initiate recalculation of routing paths in case of network failures. Path computation and routing table updates takes time and induce packet loss in the network. On the other hand, by design, IP networks have Shared Risk Link Groups (SRLGs) that might give several failure indications. Current OSPF protocol cannot identify SRLGs and separately responds to each failure notification, which leads to higher packet losses. Within the ECODE project framework, a process is being developed to identify the SRLGs from the time sequences of LSAs that arrive in the process of failures. Here a router identifies an SRLG locally and reduces protection switching time by simultaneously updating the forwarding table for all the links that fail under the same SRLG. A simple Bayesian network based approach to model the SRLG identification have been developed. The Bayesian network based model includes a state transition approach that can per-

form online learning and infers probabilistic determination procedure for SRLGs. Further the study of the covariance among the LSA inter arrival times are being considered to enhance the probability based state space Bayesian network model and to include temporal data to infer the state transition probabilities. A realistic simulation scenario using GTNetS is being carried out.

- The aim of profile-based accountability is to 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. There are two major tasks expected from the machine 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. Each profile indicates specific network traffic behavior associated with the usage and demand on the network resources. 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. Two distinct machine learning techniques are considered. A clustering technique based on the Fuzzy K-Mean algorithm is used to detect and identify commonalities in users/subscribers *actions* (unsupervised learning). An Hidden Markov Model (HMM) is learned for the supervised classification of subscribers/users (supervised learning).

List of Authors

UCL	Benoit Donnet, Juan Narino, Damien Saucez
ULg	François Cantin, Pierre Geurts, Guy Leduc, Yongjun Liao
IBBT	Didier Colle, Goutam Das, Wouter Tavernier
ALB	Dimitri Papadimitriou, Ing-Jyh Tsang, Werner Van Leekwijck

List of Figures

Table of contents

Chapter 1

Introduction

1.1 Scope of Deliverable

This deliverable is part of WP3 (Cognitive network and system experimentation), which is an experimental work package that started at M03. The feasibility, benefits and applicability of introducing a cognitive engine in the network elements are experimented using a number of use cases covering different problem areas identified as Internet architectural and design challenges (manageability, security, availability, routing scalability and quality). WP3 comprises three tasks (T3.1, T3.2, and T3.3) that are dedicated to the experimental phase 1. Each of these tasks is associated with the networking scientific and technical objectives, including prototype development, setting up the test environment and performing the actual testing. Three types of hands-on experimental tasks are planned in this work package:

- T3.1: Experimentation on Technical Objective 1 (TO1) addressing adaptive traffic sampling and management, path performance monitoring, and intrusion and attack/anomaly detection techniques;
- T3.2: Experimentation on Technical Objective 2 (TO2) addressing techniques for path availability estimation, for improving network recovery and resiliency, and profile-based accountability;
- T3.3: Experimentation on Technical Objective 3 (TO3) addressing techniques to improve routing system scalability and quality (convergence, stability/robustness, and stretch).

For each Technical Objective (and in particular for TO2, which is the topic of this deliverable), use-case driven experimentation are performed on the cognitive engine (network and system architecture) elaborated in WP2. The experimentation of the different networking scientific and technical objectives will use different test settings and running conditions. Each experiment will be structured around the following approach:

- Formalisation of the technical problem addressed by the use case;

- Description of the machine learning techniques and/or algorithms used;
- Description of the implementation of the use case so far;
- Description of the experimental setup: Performance objectives, Evaluation criteria, Methodology;
- Recommendations for the integration of the use case into the common ECODE architecture ;

This deliverable D3.4 focuses on Technical Objective 2 (TO2), which is composed of the following three use cases:

b1) Path availability (UCL and ULg)

- Design and experiment techniques, embedded in our so-called IDIPS server, to rank internet paths based on their characteristics, such as delays, and available throughput;
- Find and experiment on-line machine learning techniques to estimate future path characteristics based on past measurements, in order to reduce the future measurement load;
- Use machine learning techniques to Improve Internet Coordinate Systems to better estimate delays between nodes with scalable active delay measurements;
- Design and experiment criteria, based on node coordinates and some measured delays, to discover appropriate routing shortcuts

b2) Network recovery and resiliency (IBBT and ALB)

- Reduce the recovery time and increase the recoverability, when routes are updated in a network;
- Design and experiment machine learning techniques to minimize packet loss during rerouting;
- Design and experiment machine learning techniques to infer SRLGs (Shared Risk Link Groups) in networks, i.e. sets of links that can fail all together;
- Design and experiment machine learning techniques to anticipate the occurrence of SRLGs very early.

b3) Profile-based accountability (ALB)

- 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

- Design and experiment machine learning techniques to cluster customer profiles based on their network resource demands
- Design and experiment machine learning techniques to classify customers according to the learned profiles

1.2 Structure of Document

The path availabity use case (b1) is addressed in chapters 2 and 3. The former presents the IDIPS architecture proposed for intelligent ranking of Internet paths. In this context, machine learning techniques are designed to estimate future path characteristics based on past measurements, in order to reduce the future measurement load. Chapter 3 complements this approach by adding an Internet Coordinate System in the routing architecture, with the goal of finding low delay paths in a scalable manner.

Chapter 4 addresses the network recovery and resiliency use case (b2). Two distinct problems are solved: firstly, packet loss during re-routing is minimized by accelerating the updates of the router FIBs (Forwarding Information Base). This is done by updating the major flows first and by grouping the updates into batches of optimal sizes. Secondly, routing update messages are used to identify shared risk link groups (SRLGs) and thereby to anticipate their occurrence very early, e.g. by deciding whether a link failure is likely to be isolated or followed by all the other link failures in a SRLG.

Chapter 5 addresses profile-based accountability, whose aim is to 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. For this purpose, two distinct machine learning techniques are considered. A clustering technique based on the Fuzzy K-Mean algorithm is used to detect and identify commonalities in users/subscribers *actions* (unsupervised learning). A Hidden Markov Model (HMM) is learned for the supervised classification of subscribers/users (supervised learning).

In chapter 6 we describe how we currently view the mapping of all the use cases described so far onto this general architecture that was described in deliverable D2.1. In particular we show how their functionality is split among the Routing Engine (RE), Forwarding Engine (FE) and Machine Learning Engine (MLE) and which message exchanges are necessary among them.

Chapter 7 concludes this deliverable. It summarizes the main contributions and describes future work.

Chapter 2

Intelligent Path Ranking Using IDIPS

ISP-Driven Informed Path Selection (IDIPS) is a service aiming at ranking paths according to their performance. Fig. 2.1 presents an IDIPS service overview. To be able to make path ranking, IDIPS needs, from one hand, to collect path performance information (i.e., top of Fig. 2.1), store it, and, on the other hand, collect path ranking requests and process them with a ranking algorithm.

The collection part of IDIPS works with path information. A path is defined by a $(source, destination)$-pair and information are divided into two categories: (i) administrative and (ii) measurement. Administrative information is related to the network management like network policies and billing, but also routing information such as BGP feeds or IGP topologies. On the other hand, measurement information dynamically evolves with the traffic and refers, for instance, to the delay or bandwidth.

Collected information is abstracted and stored in a unified format within IDIPS for later use.

When a ranking request arrives, IDIPS computes a cost for each feasible path in the request. It then groups paths of similar costs within the same rank and informs the requester of the path ranks.

The machine learning aspect related to IDIPS concerns the path information collection. This collect might be done in two ways: actively (i.e., probes are injected into the network) or passively (i.e., information is silently collected). As active probing is intrusive and resource greedy, we propose to consider machine learning techniques to infer active probing results without injecting traffic (or at least reducing the amount of traffic) in the network.

Sec. 2.1 formalizes the problem of path performance prediction as a time series problem; Sec. 2.2 explains our machine learning techniques for predicting path performance; Sec. 2.3 explains how we could implement the IDIPS service; Sec. 2.4 evaluates the accuracy of our machine learning techniques; finally, Sec. 2.5 concludes this chapter by discussing the future directions.

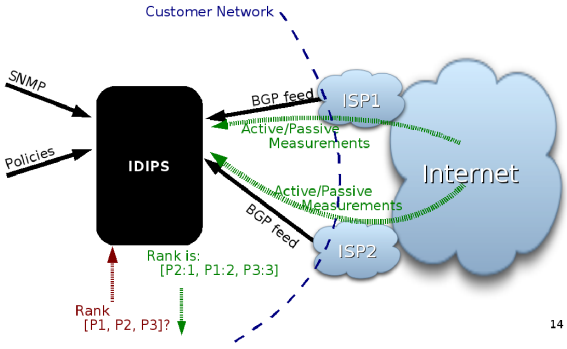


Figure 2.1: Overview of the IDIPS service

2.1 Problem Formalization

Prediction of Path Performance Metrics (PPM) is a topic that has been explored in the past [8, 9, 10]. However, how to adapt to the network dynamic conditions such that models reflect this is still an open question. Furthermore, how to update the models and the effects of using partial updates, that is, how to handle predictions when new data for model update is not periodically available are topics that have not been throughly explored in PPM prediction literature.

The main idea is to *infer* or learn different statistical properties from the time series data. After model parameters have been found, predictions of the PPMs can be made. Statistically, predicting future PPMs from past measurements can be stated, in machine learning terms, as a *time series* regression problem. The *training* dataset D consists of measurements from the previous time step $t-1$ back to some index $t-k$ in the past:

$$D = \{y_{t-1}, y_{t-2}, \ldots, y_{t-k}\} \tag{2.1}$$

A model of this time series aims at predicting the next PPM value y_t and, possibly, additional values in the future $y_{t+1}, y_{t+2}, \ldots$.

2.2 Machine Learning Techniques

Machine Learning consists of algorithms and techniques that aim at recognizing and learning automatically patterns from the data. In the context of PPM prediction, the main goal is to estimate models for accurate prediction of PPMs. Statistical models are fit from the data, and its different parameters are estimated. Furthermore, adaptivity was included in the models, in order to accommodate changing network conditions.

2.2.1 Standard Models

2.2.1.1 ARMA Model

An *AutoRegressive Moving Average* (ARMA) model [11] describes a time series as a linear combination of previous samples plus Gaussian noise, thus constructing a model for forecasting. The amount of p previous samples used is called the *autoregression order*. The q noise samples, used in the model are called the *moving average order*. A general $ARMA(p, q)$ model is described by the following equation:

$$y_t = \alpha_1 y_{t-1} + \cdots + \alpha_p y_{t-p} + \phi_1 \epsilon_{t-1} + \cdots + \phi_1 \epsilon_{t-q} + e_t \tag{2.2}$$

where $\epsilon_i \sim N(0, \sigma^2)$, $e_t \sim N(0, \sigma^2)$, p indicates the order of the auto-regression, and q the order of the moving average. For simplicity, $q = 0$ is assumed for the rest of this paper.

The auto-regressive order is chosen by doing rolling crossvalidation for each of the involved series for a particular loss function.

2.2.1.2 State Space Model

Linear Dynamical systems (LDS) can be described as a system of joint linear equations, describing their dynamical evolution and how these observations are made [12]. A state space representation of an LDS is modeled as the evolution of nonobservable variables $\vec{x}$ and observable variables $\vec{y}$. The LDS is expressed as:

$$\vec{y}_t = \mathbf{A}\vec{x}_t + \vec{v}_t \tag{2.3}$$

$$\vec{x}_t = \mathbf{B}\vec{x}_{t-1} + \vec{w}_t \tag{2.4}$$

where $\vec{v}_t \sim N(0, R)$ and $\vec{w}_t \sim N(0, Q)$ model noise either in the transition or in the actual measurement. The link between observed variables $\vec{y}_t$ and hidden variables $\vec{x}_t$ is given by the matrix **A**. The matrix **B** describes the dynamical evolution of the system in the hidden or dynamical space. The parameters $\mathbf{A}, \mathbf{B}, R, Q$ are estimated from the collected dataset D, using an *expectation maximization* (EM) algorithm for LDS [13].

Once the model parameters have been found, there are two possible ways to predict data. The first one consists in initializing the LDS linear system equations by using as initial vector the last observation available in the training set. The second way, allowing inclusion of new data in a natural way, is by use of Kalman filtering [14].

2.2.1.3 Support Vector Regression (SVR) Model

Support Vector Machines (SVM) have been used for classification, regression, and time series analysis [15]. SVR intends to find a function $f(\vec{y})$, by solving an optimization problem, that calculates the best function fitting a regression hyperplane that passes within a ϵ distance from certain training samples (support vectors). An epsilon-loss

function is used to generate an "ϵ tube". The regression function $f(\vec{y})$ passes through the center of the tube. After defining a Lagrangian and solving for the dual variables α's, the regression function is defined as:

$$\hat{\vec{y}}_t = f(\vec{y}) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) k(\vec{y}_i, \vec{y}) + b \tag{2.5}$$

$$\hat{\vec{y}}_t = \{\vec{y}_{t-1}, \vec{y}_{t-2}, \cdots, \vec{y}_{t-k}\} \tag{2.6}$$

For dynamical systems, we can use the previous samples as the input to the regression function [15]. The input data $\vec{y}$ used for the prediction at a given time t is a vector representing the current data context: $y = [y_{t-1} \dots y_{t-p}]'$ the p previous measurements. By analogy with an AR model, we refer to p as the SVM model order. The input vector $\vec{y}$ is projected to a higher dimensional space through a non-linear mapping $\Phi(y)$. Linear modeling is simpler in such a higher dimensional space, and equals to non-linear modeling in the original low dimensional space [15].

In all SVM experiments, the radial basis function kernel is used. This kernel is described by $k(y_i, y) = \gamma \exp(||y_i - y||^2)$ where y_i is the input vector used to represent some past measurements in the training, y represents the current context as above, and γ is a meta-parameter known as the kernel width. The kernel width and the amount of previous measurements used as input vector is found by using crossvalidation.

2.2.1.4 Joint Support Vector Regression

As seen in the previous section, the SVR accepts as input a vector that can contain n dimensions. In order to do joint data prediction, a vector containing the previous samples for delay and throughput, $\vec{y} = \{d_{t-1}, \dots, d_{t-k}, th_{t-1}, \dots, th_{t-k}\}$ and as "labels" either delay or throughput is used. All other details regarding this method are identical to the case of SVR. For the Joint SVR method, the amount of previous necessary samples and the width of the kernel were also found by rolling crossvalidation.

2.2.2 Model Update Methods

2.2.3 State Space Model

In order to use new available data for state space model predictions, an algorithm called *Kalman filter* can be used [14]. A Kalman filter takes new available data, combines it with the previous prediction and makes an update to the state vector (hidden vector), thus, updating the state space model in an automatic way. Kalman filtering consists of two steps: *prediction* and *update*.

In the prediction step, the state equation is used to predict a new state. Then, the observation equation can be used to predict the new value. Once a new observation

becomes available, the system state is corrected accordingly to the new sample. The state vector and the covariance matrix are also updated for future predictions (update step). The state is corrected by a variable amount K, which minimizes the difference between the predicted state and the actual state, in a mean square sense.

Model updating is natural with Kalman filtering. If new data is available, the "prediction-update" algorithm is applied, so in the next step the state vector is updated reflecting the last data. If there is no new data, only the prediction step is done. The final result is a mixture of "predict-update" observations when new data is available and only "predict" observations when there is no data available.

2.2.3.1 ARMA Model

Although there exists a particular way to update an ARMA model prediction once new data becomes available [11], the update is cumbersome and is not general enough in case there are two or more consecutive missing samples. It is even more difficult, in the general case, when some new out-of-sample data becomes available in a non periodical way. In order to deal with this, it is more practical to convert an ARMA system to its state space representation and then use Kalman filtering for updating. Although many possible representations exist for an ARMA system, in this work the representation suggested by Hamilton [16] is used. Once the AR system has been converted to the state space representation, Kalman filtering can be used for updating the AR model predictions, in exactly the same way updates are done to a "pure" state space representation. Kalman filter allows one, as said before, to handle non periodic data updates in a natural way.

2.2.3.2 SVM Model

In the case of SVM regression, the updating is straightforward. At each step, if the next sample is not a new available sample, this sample is replaced by its prediction, to be used as part of the input vector of future predictions. If the next sample is available, a prediction is made, and then the new available data is used directly as part of the input vector instead of the prediction in the previous step.

2.3 IDIPS Implementation

The IDIPS architecture is based on three modules that cooperate with each other (see 2.2). The first module, named *Path Information Collector* (PIC) (Sec. 2.3.1), collects path information and converts it into a unified format to store it in the *Knowledge Base* (KB) (Sec. 2.3.2). The KB can be seen as a path performance information storage system optimized for flexibility and lookup speed. Finally, the *Decision Engine* (DE) (Sec. 2.3.3) is in charge of ranking the path when a path ranking request arrives at IDIPS.

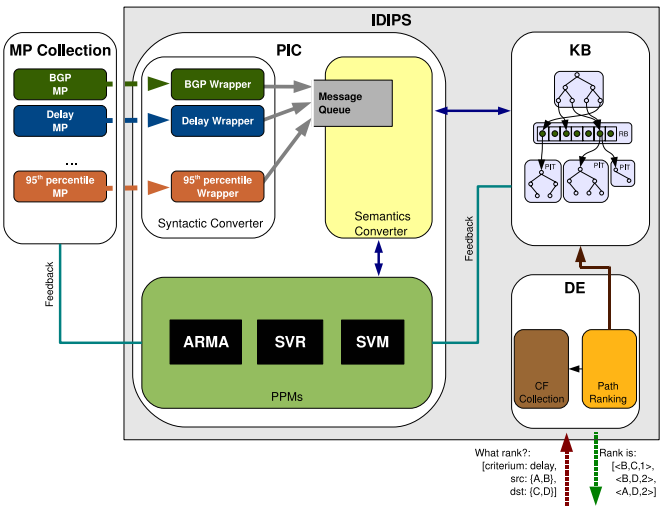


Figure 2.2: IDIPS architecture

2.3.1 Path Information Collector

The PIC has three components, as illustrated in Fig. 2.2: (i) the *Syntactic Converter*, (ii) the *Semantics Converter* and (iii) the *PPMs* (i.e., Path Performance Metrics).

The Measurement Points (MPs) send path information they measured to their specialized wrappers using the raw format of their choice. Wrappers make syntactic conversion from a MP specific format into a unified PIC format. The converted information is then passed to a semantic converter. If required, the information can be processes by machine learning elements in the PPMs. The machine learning elements use this information to refine their predictions. Machine learning elements can send feedback to the MP to adapt their parameters (the feedback arrow). The semantic converter purpose is to collect specialized path information and convert it into path attributes. A path attribute is an integer representation of the path performance information.

2.3.2 Knowledge Base

The KB is the IDIPS part storing path attributes collected at the PIC. To make efficient lookups and reduce memory consumption, paths are decomposed into a source and a destination. To store a path attribute, the source is first inspected to find to which *Responsibility Base* (RB) the path belongs. The path attributes are pushed to the path's responsibility base. A *Path Information Tree* (PIT) is attached to each RB to store the path attributes. A PIT is a tree where the key is a destination and the content is a collection of path attributes. The machine learning model parameters are represented like any other path attribute in the KB.

2.3.3 Decision Engine

The DE compares the paths in order to select the best one according to some criteria. To do so, the DE defines a collection of *Cost Functions*. A Cost Function returns the cost of a $(source, destination)$-pair (i.e., a path) for a given criterion. The cost is a numerical value characterizing a path according to one or more metrics. The cost must respect two constraints. First, the lower the cost, the better the path. Second, costs comparison relationship has to respect transitivity. The transitivity is the key point of Cost Functions as it allows one to estimate the cost of any path independently and order them afterwards. Transitivity allows costs caching and parallel computation. Another important point of transitivity is enabling combinations to create more complicated Cost Functions. To ranks paths, the DE calls the appropriate Cost Function for each possible path to rank. It then creates the ranked paths list such that the best paths are those with the lowest cost and the worst with the highest. Paths in the returned list are grouped by rank. The first group of paths in the list contains all the paths with the same lowest cost value. The second group contains those with the second lowest cost and so on. Rank value can equal cost if there is no privacy requirements (ranking can be used to hide costs in order to hide path information). An example of Cost Function might be found in [17]. A Cost Function can be a machine learning model that get its parameters directly in the KB.

2.4 Experimentation

The PPMs under consideration are delay and throughput. Delay is a measure of the time that takes to data to go to a given destination and to come back. In this study, Round Trip Time (RTT) measured using ICMP ping is used. Throughput is defined as the rate of successful message delivery over a communication channel and is measured in bits per second (bps).

2.4.1 Data Collection

For evaluating the prediction techniques described in Sec. 2.2, we performed an extensive measurement campaign between March 27th 2009 and April 13th 2009. We used five PlanetLab machines as vantage points, mostly located in Europe: two in England (London and Cambridge), one in Norway (Tromsø), one in Turkey (Istanbul), and one in USA (Pasadena).

Delay and throughput periodical measurements were collected jointly by monitoring (i.e., with tcpdump) a large file downloaded from FTP Linux servers. Each vantage point downloaded the file from each FTP server every 20 minutes, obtaining 100 throughput measurements each time. Simultaneously, five pings were made to each destination from the PlanetLab nodes. The total amount of data collected was around 16GB. PPMs measurements were averaged to obtain a single data point each 20 minutes, one for delay and one for throughput. The campaign running time was divided into 20 minutes

slots. This is a reasonable assumption in relatively well connected nodes [18]. All the measurements that fell into a given slot were averaged.

Due to network and server conditions, some of the measurements were unavailable. Indeed, if a particular server did not respond at the time of downloading the file, the measurement towards this server was given up. As a result, there were some gaps in the resulting measurements. After filtering out this data, 45 time series, each for delay and throughput respectively, form the actual data set. Each time series contains 1.296 measurements or samples. In order to fill the gaps, we used the K-nearest neighbors algorithm, with a window of six samples.

2.4.2 Crossvalidation

A crossvalidation method called *rolling window* crossvalidation [19] is used. As defined before, the *forecast origin*, is the point in time n from where the predictions are going to be made, usually the last sample of the training set. The *lead time* is the time h for which a prediction is being made. So, a prediction at *lead time* h from *forecast origin* n is a prediction at $t = n + h$. In the rolling window, an horizon H is defined. This is the maximum lead time for which a prediction is going to be made, given a forecast origin. Then, after H predictions where made, a new sample is used the forecast origin is moved to the next sample, $t = n+1$ and the system is retrained with this new sample. Then, from the new forecast origin, another H predictions are made. This process is repeated M times. Then, for each one of the M times, for a given lead time h, all the errors are averaged, thus, obtaining a measure of error per lead time. Finally, and average of the average per lead time is found, thus the final error is obtained.

2.4.3 Error Measurements

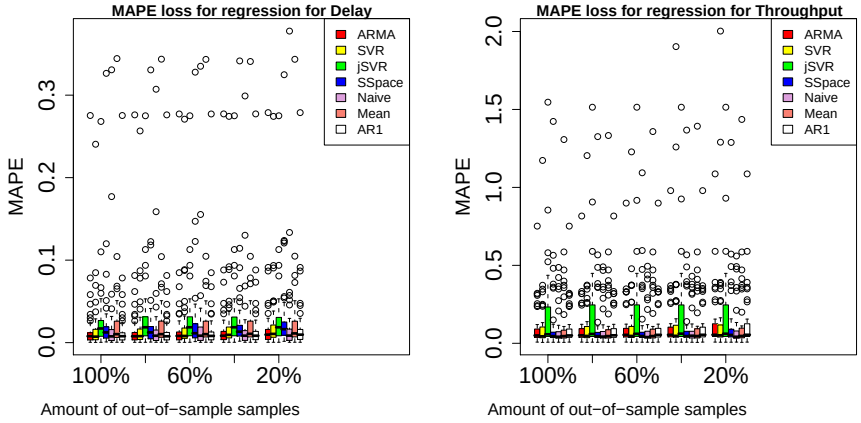
In order to measure the error made by the models, there are as much as 18 error measurements [20]. The *Mean Absolute Percentage Error* (MAPE) was selected for this work. The MAPE measures the difference in a percentual sense, between the prediction from the actual value. It is defined as, where $\hat{y}_i$ is the prediction and y_i is the actual value:

$$MAPE = \frac{1}{N} \sum_{i=1}^{N} \frac{\hat{y}_i - y_i}{y_i} \tag{2.7}$$

2.4.4 Results

For all time series for delay and throughput, four prediction models are found:

1. ARMA model. The autoregression order p is found via rolling crossvalidation



(a) Delay (b) Throughput

Figure 2.3: Box Plots under MAPE error for delay and throughput prediction

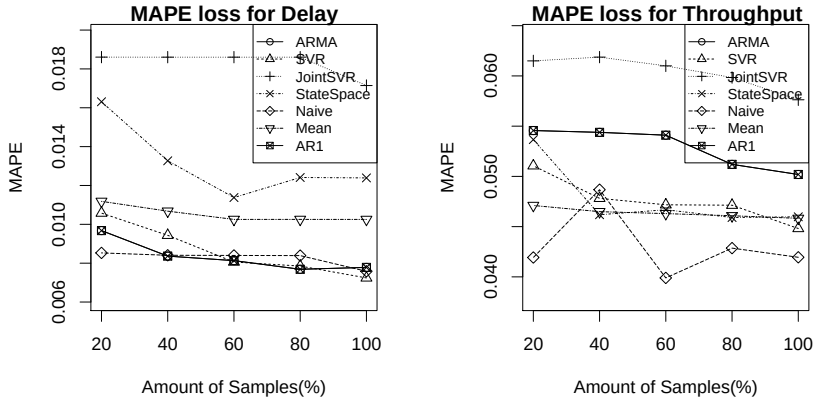
2. Individual SVR model. An SVR is fit for a one dimensional input vector, one dimensional output. If d is the delay and th is the throughput value, a series of vectors with $\{d_{t-1}, \cdots, d_{t-k} : d_t\}$ and $\{th_{t-1}, \cdots, th_{t-k} : th_t\}$ where used. The number of previous samples to use were found by crossvalidation. The kernel width γ is also found by crossvalidation.

3. Joint SVR model. An SVR is fit for a two dimensional input vector and one dimensional output. If d is the delay and th is the throughput value, a series of vectors where used with $\{d_{t-1}, \ldots, d_{t-k}, th_{t-1}, \ldots, th_{t-k} : d_t\}$ and $\{d_{t-1}, \ldots, d_{t-k}, th_{t-1}, \ldots, th_{t-k} : th_t\}$. The number of previous samples to use were found by crossvalidation. The number of previous of used samples is the same for throughput and for delay. The kernel width γ is also found by crossvalidation

4. State Space representation. In this case, a series N of bidimensional vectors $\{d_t, th_t\}$ are used to find the parameters of the State Space model. Then, when the model is left to run, the desired predicted quantity is taken from the bidimensional vector that results after prediction.

When fitting the models, data is split into three types of datasets, *training*, *validation* and *test* sets. When the meta parameters are being found, first, the dataset is split into three sets: 70% for training, 10% for validation, and 20% are held out for future testing. Then, the model parameters are found over the 70%. The meta parameters and the results are validated on the 10% left for validation. Once the meta parameters are found, the final prediction results are found by retraining the model, this time with 80% of the data, and then using the remaining 20% for testing. The results are crossvalidated and each lead time is evaluated 10 times.

In the experiments, the amount of available future samples is varied, from 100% available samples to 20%, in steps of 20%. The accuracy of the model is checked against the full 100% available samples vector. The accuracy of the model under different amounts of available samples is crossvalidated and the results are compared.



(a) Delay (b) Throughput

Figure 2.4: Median of MAPE loss for delay and throughput for different Models

Fig. 2.3 and 2.4 resume the results. In Fig. 2.3, a box plot of the results can be seen. A box plot contains the minimum of the dataset (lower bar) followed by the first quantile (lower part of the box), and then, the median of the data (bold line inside the box). Next, the third quantile can be found (upper part of the box), followed by the maximum of the data within five quantiles (upper bar), and all outliers are shown as dots. Here, it can be seen that most of the methods give approximately the same result, however, some methods give a more variable performance compared to other.

As can be seen from Fig. 2.3, JointSVR regression gives the more variable performance of both methods. ARMA, in contrast, is the least variable method. Individual SVR gives a not so variable performance with all future samples available, but the performance is more variable with less future samples compared with other methods. Also, the variation is higher in the throughput plots. This indicates that throughput is more variable and somewhat more difficult to predict than delay.

In Fig. 2.4, we can see a comparative plot of the median of the MAE error for different methods. The median is chosen because it is more resistant to heavy outliers. As can be seen from Fig. 2.4, the best performing method is individual SVR. However, its performance degrades quickly as future samples are taken away. ARMA method does not present as good performance as SVR but its performance is more stable when future samples are more scarce. The ARMA models better adaptivity could be explained by the fact that Kalman filter gives a higher adaptability compared to SVR.

2.5 Future Work

The most important issue to solve in PPMs prediction is *adaptivity*. Although the models reported in this document include adaptivity, it is possible to see from the data that adaptivity is the model most critical component. Also, since adaptations must be made after measuring, work is underway to find the optimal sampling rate in order to minimize intrusive network measurements.

Regarding IDIPS itself, future works are twofold. First, in terms of implementation, efforts must be done in improving the current. For instance, we want to make IDIPS IPv6-compliant. We also want to develop libraries and interfaces for controlling MPs. Second, in terms of evaluation, we want to determine the effects of oscillation when a decision is taken.

Chapter 3

Delay estimation and delay-based path selection and routing

Internet Coordinate Systems (ICS) have been widely used in large-scale network applications. The success roots in the embedding of Internet nodes into a metric space or a coordinate system. When the coordinates of the nodes are computed, the prediction of the distances (delays) between two nodes is a straightforward application of a distance function where no explicit communication between them is required. This significantly reduces the overhead of active probing and largely improves the efficiency of the network.

However, Internet delay space is not a metric space. The triangle inequality is often violated due to the Internet's structure and routing policies. It has been shown that the violations of triangle inequality (TIVs) in the Internet delay space are not rare and that their impact on ICS are not negligible. Figure 3.1 demonstrates the embedding of three nodes that violate triangle inequality (i.e. $\overline{AB} > \overline{AC} + \overline{CB}$) into a metric space. It can be seen that the longest edge $\overline{AB}$ is shrunk and the other two edges, $\overline{AC}$ and $\overline{CB}$, are stretched in order to satisfy triangle inequality in the embedding space. Therefore, the presence of TIVs degrades the accuracy of an ICS. Many studies have reported that this inaccuracy can badly hurt the performance of neighbor selection, which is crucial for many distributed systems.

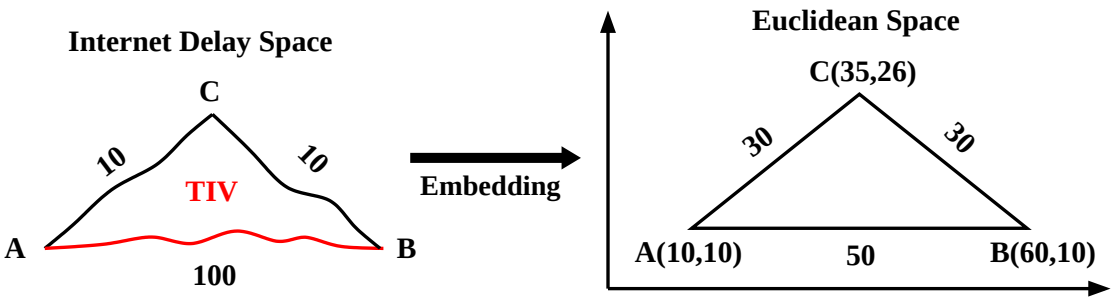


Figure 3.1: Embedding three nodes that cannot form a valid triangle into a metric space.

In this chapter we will show how Machine Learning techniques can be used to im-

prove the accuracy of an ICS. We will derive automatically a criterion that can be used by nodes to better select their neighbours in the ICS and thereby reduce the impact of TIVs.

The knowledge of estimated delays between nodes can be useful to select better paths for real-time applications (refer to chapter 2 on IDIPS for a more detailed explanation on path selection). However, since the Internet was not developed with QoS guarantees in mind, the default route between two nodes is not guided by QoS constraints (and, in particular, by constraints on the delays). In many cases the route between two nodes A and B chosen by the network is not the lowest-delay path and it is possible to find nodes C that are *shortcuts* in term of delays:

$$RTT(A,B) > RTT(A,C) + RTT(C,B)$$

where $RTT(X,Y)$ is the ***RTT***[1] (Round Trip Time) between the nodes X and Y. For any path AB in a network, our objective is to find some nodes C that are shortcuts in terms of delays. If we are able to find these shortcuts we will be able to provide a better service to the applications using the network : instead of sending the data directly from A to B, we will use a node C as relay in order to obtain smaller delays.

In this chapter we will also propose some methods that rely on the nodes running an ICS to detect useful routing shortcuts in networks.

3.1 Problem Formalization

3.1.1 Improving the ICS

We focus on a classical ICS algorithm, Vivaldi [21], which approximates a network by a system of springs and seeks to minimize its energy. The minimization is fully distributed and iteratively done at each node. In each iteration, each node updates its own coordinates by minimizing the embedding error with respect to its neighbors. After some iterations depending on the complexity of the network, the coordinates of the nodes become stable, i.e., the embedding error of the whole network is smaller than a threshold, with a small amount of jitter.

To collect training data, we ran Vivaldi on P2psim and Meridian respectively and recorded the coordinates of the nodes, after they become stable, at K different times or ticks. $K = 100$ in our experiments. From the coordinates obtained, we computed the time-varying distances between each pair of neighboring nodes. We denote the measured distance by d and the estimated distance by $\hat{d}$. Thus, for each edge, we have $\{d, \hat{d}_1, \hat{d}_2, \ldots, \hat{d}_K\}$. For the K estimated distances, we further calculated some statistics including $\hat{d}_{max} = max\{\hat{d}_1, \ldots, \hat{d}_K\}$, $\hat{d}_{min} = min\{\hat{d}_1, \ldots, \hat{d}_K\}$, $\hat{d}_{mean} = mean\{\hat{d}_1, \ldots, \hat{d}_K\}$, $\hat{d}_{median} = median\{\hat{d}_1, \ldots, \hat{d}_K\}$ and $\hat{d}_{std} = std\_dev\{\hat{d}_1, \ldots, \hat{d}_K\}$.

[1]The RTT between two nodes X and Y is the time necessary to travel in the network from X to Y and go back to X from Y.

The input variables to the machine learning algorithm consist of various combinations of these statistical variables and the measured distances. Note that the input variables are normalized in different ways in order to get rid of the influence of particular network conditions. Currently, 64 input variables for each edge are used. A few examples are

$$\frac{\hat{d}_{max} - \hat{d}_{min}}{d}, \frac{\hat{d}_{mean} - \hat{d}_{median}}{d}, \frac{\hat{d}_{mean} - d}{\hat{d}_{max}}, \frac{\hat{d}_{mean}}{\hat{d}_{std}}, \frac{\hat{d}_K}{d}, \frac{\hat{d}_{std}}{d}.$$

Note that the last two variables, $\frac{\hat{d}_K}{d}$ and $\frac{\hat{d}_{std}}{d}$, are equivalent to those used in [22] and [4] respectively. The former, $\frac{\hat{d}_K}{d}$, is a measure of relative estimation error, denoted by REE, while the latter, $\frac{\hat{d}_{std}}{d}$, is the standard deviation of the relative estimation error, denoted by std_REE.

For supervised learning, the outputs or labels of the edges are needed. Here, the outputs are simply TIV or non-TIV, which can be easily obtained from the measured distances.

The problem we want to solve with Machine Learning is to derive automatically a classification model, based on the above-mentioned variables, that would tell with high likelyhood whether or not an edge between two nodes is a TIV-edge.

3.1.2 Finding routing shortcuts

When an edge AB is a TIV-edge, this means that there exists a routing shortcut ACB via some node C in terms of delay. The second subproblem we address then consists in finding candidate C nodes that are likely to be interesting routing shortcuts.

Using only the estimated delays provided by an ICS to find the shortcuts in a network is useless. Indeed, the principle of an ICS is to give to each node of the network a coordinate in a metric space such that the distance in the metric space between the coordinates of two nodes gives an estimation of the delay between these nodes. Since the triangle inequality must hold in a metric space, it is impossible to find three nodes such that

$$EST(A, B) > EST(A, C) + EST(C, B)$$

where $EST(X, Y)$ is the estimated RTT between the nodes X and Y. So, we must combine estimations with measurements in order to obtain a shortcuts detection criterion. In addition of the estimated RTT of each path in the network, we consider that we can obtain the following measurement results. First, if we look for a shortcut for the path AB, we assume that $RTT(A, B)$ can be measured. Secondly, we assume that we can obtain the Vivaldi's measurement results done between the nodes and their neighbors in order to compute the coordinates.

Given these data we want to find criteria that provide a set of C nodes that are probably shortcuts for that path. An extended version of this problem, for further research, would be to rank the C nodes.

3.2 Machine Learning techniques/algorithms used

We attempt to detect TIVs in order to avoid them and thereby reduce their negative impacts on an ICS. First, we apply supervised learning methods to learn a TIV classifier. A discriminative TIV detection criterion, called $OREE$, will be found. Then, $OREE$ is used to remove severe TIV edges from the ICS iteratively. Experimental results will show that $OREE$ is better at TIV detection than other existing methods [22, 4], and the performance of nearest neighbor selection is gradually improved as severe TIV edges are progressively removed from the ICS.

Our first objective is therefore to find variables that are consistently discriminative for TIV detections regardless of the networks. Supervised learning methods are promising techniques that allow us to achieve this goal. This section details the use of a particular supervised learning method, namely decision tree, in finding discriminative variables for TIV detection.

Decision Tree (DT) is one of the most popular supervised learning algorithms. Each decision tree is a classifier in the form of a tree structure, where each interior node specifies a binary test carried on a single input variable and each terminal node is labeled with the value of the output. In the learning phase, a decision-tree builder recursively splits the training samples with binary tests, trying to reduce as much as possible the uncertainty about the output classification in the resulting subsets of the samples. The splitting of a node is stopped when the output in it is homogeneous or some other stopping criterion is met. During learning, a byproduct is the ranking of the input variables according to their importance, which is often used to find discriminative variables. In the classification phase, we start from the root of the tree and move through it until a terminal node, where the classification result is provided.

3.2.1 Methodology: Scenarios and Tools for Learning

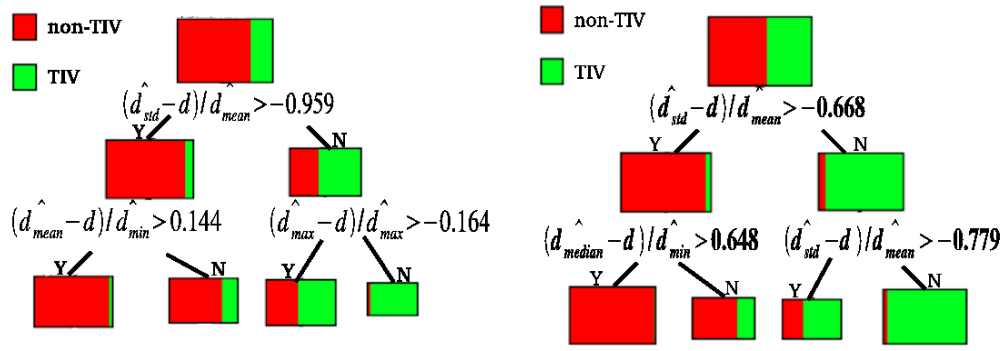
We have learned our decision trees using a data mining software called PEPITO[2] which integrates most popular machine learning algorithms. The training data collected was randomly divided into disjoint learning and test sets of roughly equal sizes. In other words, we used half of the data to build the trees and the other half to evaluate the trees.

3.2.2 Learnt model and discriminative variables

Two different trees were separately built for P2psim and Meridian, shown in Figure 3.2.

By examining the root nodes of both trees in Figure 3.2, it can be seen that the first test is on the same variable, $\frac{\hat{d}_{std}-d}{\hat{d}_{mean}}$, which appears to be the most discriminative. We call it $OREE$ representing Oscillation and Relative Estimation Error, because it consists of

[2]http://www.pepite.be



(a) P2psim (b) Meridian

Figure 3.2: Top three levels of the decision trees built on P2psim and Meridian data set. The colour in the rectangular boxes reflects the proportions of TIV and Non-TIV classes.

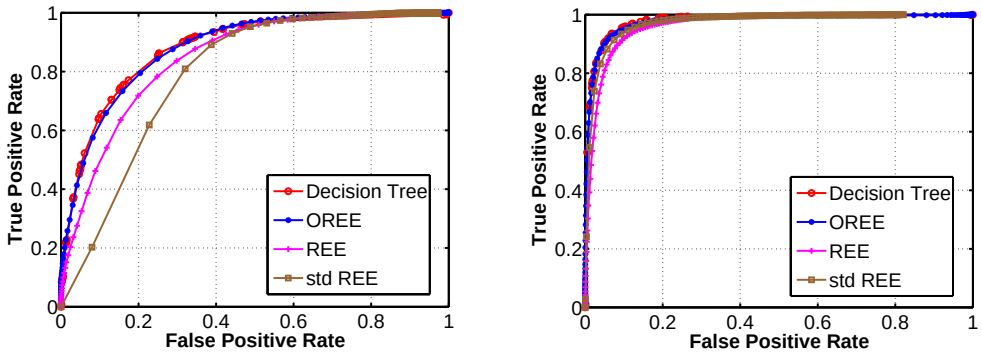
two parts: $\frac{\hat{d}_{std}}{\hat{d}_{mean}}$, a relative oscillation measure and $\frac{d}{\hat{d}_{mean}}$, a relative error measure. On both trees, when the value of $OREE$ is smaller than the cut point, the edge is more likely to be a TIV edge, and vice versa.

3.2.3 Evaluation of the learnt model

ROC (Receiver Operating Characteristic) curves are useful for the evaluation of machine learning techniques. A ROC curve is a graphical plot of the false positive rates (FPR) as x-axis versus the true positive rates (TPR) as y-axis for a binary classifier as its discrimination threshold is varied. In the ROC space, the best classification performance would yield a point in the upper left corner or coordinate (0,1), representing 100% true positives and no false positives. Thus, the higher the ROC curve is, the better the classifier is. Here, we employ ROC curves to compare different TIV detection methods, including decision trees, $OREE$, REE and std_REE, shown Figure 3.3(a) and 3.3(b). It has been observed that:

- The ROC curves of $OREE$ are almost identical to those of the decision trees in both networks. This suggests that the other variables provide little extra information so that their corresponding nodes in the trees can be safely pruned with no performance degradation, using $OREE$ for TIV detection is as discriminative as the whole tree.

- The ROC curve of $OREE$ is consistently the highest in different networks, meaning that $OREE$ is the most discriminative variable.



(a) P2psim (b) Meridian

Figure 3.3: ROC curves of the decision trees, obtained by varying the threshold of the probability of TIV, and of three variables including $OREE$, REE and std_REE, obtained by thresholding the values of the corresponding variables.

3.3 Implementation

3.3.1 Improving the ICS

The use of supervised learning techniques brings out $OREE$, an effective TIV detection criterion. We then incorporate it into the neighbor selection procedure in Vivaldi.

Traditionally, Vivaldi is started with each node selecting m random neighbors. After a period of time, each node probes another m random nodes and gets totally $2 \times m$ neighbor candidates. For each edge, the value of $OREE$ is computed from the K most recent coordinates of the nodes and the current coordinates of the probed node. As edges with small $OREE$ value are more likely to be TIV edges, the neighbors of a node are updated by selecting the m candidates with large $OREE$ and abandoning the others. This neighbor update procedure is repeated every T seconds. In our experiments, $m = 32$, $K = 100$ and $T = 100$.

3.3.2 Finding routing shortcuts

We have developed two basic shortcut detection criteria.

Our first criterion is called *EDC* (Estimation Detection Criterion). To decide if a node C is a shortcut for a path AB, this criterion compare the mesured RTT of the direct path between A and B and the estimated RTT of the alternative path using C as relay. Formaly, if

$$RTT(A, B) > EST(A, C) + EST(C, B)$$

then C is considered as a shortcut for the path AB. The potential problem with this criterion is that shortcuts cannot be represented in the metric space used by the ICS:

they are a source of estimation errors and using the values of the estimated RTT when there are shortcuts is not necessary a good idea.

The second criterion is called *ADC* (Approximation Detection Criterion) and uses only the order between the estimated RTTs. For a path AB and a node C, we define C_A (resp. C_B) as the A's (resp. B's) Vivaldi neighbor that is the nearest to C according to the estimated RTTs. Since A and C_A (resp. B and C_B) are neighbors, we assume that $RTT(A, C_A)$ (resp. $RTT(B, C_B)$) is known and can be used by the criterion to approximate the RTT of the alternative path: if,

$$RTT(A, B) > RTT(A, C_A) + RTT(C_B, B)$$

then C is considered as a shortcut for the path AB. The potential problem with this criterion is that it is sometimes impossible to find a neighbor of A and/or a neighbor of B that is near C. In such cases, we obtain a bad approximation af the RTT of the alternative path and the detection result can be wrong.

3.4 Experimentation and evaluation

3.4.1 Improved ICS

Our improved Vivaldi, called "TIV avoided Vivaldi", is tested on *P2psim* and *Meridian* respectively.

We first compute TIV severity of each edge. The definition of TIV severity in [22] is adopted, which is

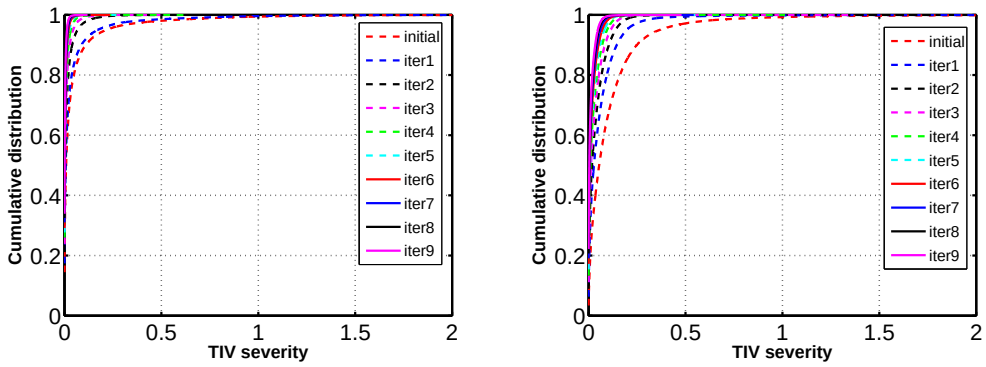
$$\frac{\sum_{c \in S} d_{a,b}/(d_{a,c} + d_{c,b})}{|S|},\ if\ d_{a,b} > d_{a,c} + d_{c,b}, \tag{3.1}$$

where $d_{x,y}$ denotes the measured delay between x and y. S is the set of all nodes in the delay space and $|S|$ is the number of nodes.

Figure 3.4 shows the cumulative distributions of TIV severity after each neighbor update. A decreasing trend on TIV severity is observed, meaning that $OREE$ is capable of identifying those edges with high TIV severity and eliminating them from ICS.

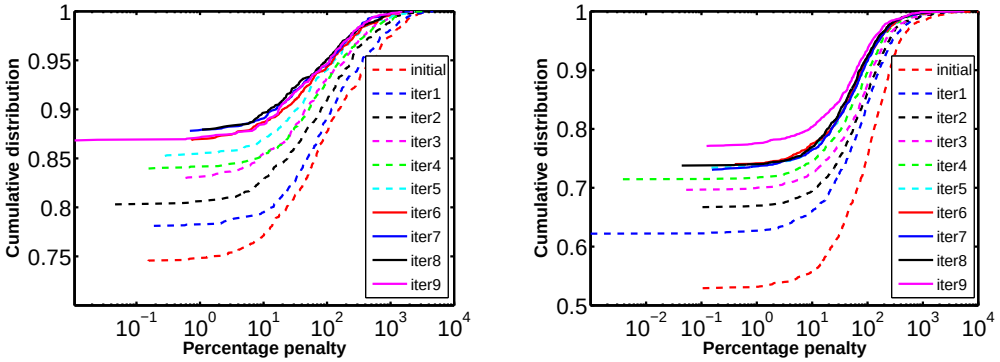
Then, we evaluate the performance of nearest neighbor selection. To this end, a subset of the nodes are randomly selected as candidates for the nearest neighbor selection. For each node which is not in the candidate set, the nearest neighbor in the candidate set is detected in the original delay space and in the embedded space respectively. Denote $dist\_to\_selected$ the distance to the nearest neighbor in the candidate set in the embedded space and $dist\_to\_optimal$ in the delay space. A penalty for nearest neighbor selection is defined as

$$\frac{(dist\_to\_selected - dist\_to\_optimal) * 100}{dist\_to\_optimal}$$



(a) P2psim (b) Meridian

Figure 3.4: Distribution of TIV severity



(a) P2psim (b) Meridian

Figure 3.5: Nearest neighbor selection penalty

This metric attempts to answer the question "how far is my nearest distance to a set of candidates?".

In our experiments, we randomly select 100 candidate nodes for *P2psim* and 125 for *Meridian* respectively. These numbers are set in order to lower the probability that a candidate node is chosen by a test node as its neighbor.

Figure 3.5 plots the cumulative distributions of nearest neighbor selection penalty when neighbor sets are iteratively updated. We can see that the performance of nearest neighbor selection is clearly improved.

3.4.2 Finding shortcuts

To evaluate the performance of our routing shortcut criteria, we use the classical true positive rate and false positive rate indicators. For a path AB, a good shortcut detection criterion must detect a node C as a shortcut if it is a shortcut for the path AB (i.e. if it is a *positive*) and must reject a node C if it is not a shortcut for the path AB (i.e. if it is a *negative*). The percentage of positives detected as shortcuts is the *true positive rate*

	EDC			ADC		
	TPR	ITPR	FPR	TPR	ITPR	FPR
P2PSim	53%	83%	2%	65%	84%	9%
Meridian	54%	64%	9%	70%	76%	25%
Planetlab	37%	75%	1%	60%	81%	5%

Table 3.1: EDC and ADC detection results using Vivaldi to estimate delays

(TPR) and the percentage of negatives detected as shortcuts is the *false positive rate* (FPR). A good detection criterion must provide a high true positive rate and a low false positive rate.

The problem is that a shortcut is not necessary useful. For example, for a path AB such that $RTT(A, B) = 100\,ms$, A node C such that $RTT(A, C) + RTT(C, B) = 99\,ms$ is a shortcut that provides an absolute gain of $1\,ms$ and a relative gain of 1%. Since using C as relay for sending data from A to B will add an additional forwarding delay, detecting such shortcuts is useless. We define an *interesting shortcut* as a shortcut that provides at least an absolute gain of $10\,ms$ and a relative gain of 10%. We also define the *interesting true positive rate* (ITPR) as the percentage of interesting shortcuts detected as shortcuts by the criterion.

To test our criteria, we used three delay matrices obtained by doing measurements in real networks. These three matrices are named P2PSim, Meridian and Planetlab and give respectively delay measurements results between 1740, 2500 and 180 nodes. In these matrices, the percentage of paths for which there exists at least a shortcut is respectively 86%, 97% and 67% and the percentage of paths for which there exists at least an interesting shortcut is respectively 43%, 83% and 16%. So, searching shortcuts in the networks modelled by these matrices can provide an improvement in term of delays for lots of paths.

We have simulated the behaviour of Vivaldi on these three networks by using the P2PSim[3] discrete-event simulator. Each node has computed its coordinates in a 9 dimensional Euclidean space by doing measurements with 32 neighbors. Then, we simply applied our detection criteria using the estimated delay matrices computed with the coordinates obtained at the end of the simulations of Vivaldi.

The true positive rates and false positive rates obtained with our criteria applied using the estimated delays produced by the simulations are given in table 3.1. We see that the percentage of interesting shortcuts detected as shortcuts (ITPR) is very good in most of the cases for both criteria. Furthermore, the percentage of non-shortcuts detected as shortcuts is generally quite low. So, these results are satisfactory. Considering these results, we prefer the EDC criterion: ADC is always able to detect a little bit more shortcuts than EDC, it also give more false positives.

It is probably possible to obtain more results by tuning Vivaldi's parameters or improving the Vivaldi's algorithm. For example, some approaches have recently been pro-

[3] `http://www.pdos.lcs.mit.edu/p2psim/index.html`

posed in order to mitigate the impact of the shortcuts on the quality of the embedding. One of these approaches simply consists in applying a non linear transformation[23] like $y = x^{1/n}$ (where n is a parameter) to the delays before trying to estimate them. We have observed that $n = 1.5$ is the value of the parameter that gives the better results according to the estimation errors. By using the estimation obtained with this optimised version of Vivaldi, we obtained much better detection results than with a classic implementation of Vivaldi. For example, on the Planetlab matrix, the EDC criterion gives an ITPR of 94% and a FPR of 3% (the corresponding values obtained with a classical implementation of Vivaldi are 75% for the IFPR and 1% for the FPR).

3.5 Conclusion and Future Work

In this work, we used supervised learning methods, namely decision trees, to successfully find a discriminative variable $OREE$ for TIV detection. Based on $OREE$, we design a TIV avoidance algorithm which effectively removes those severe TIV edges and improves the performance of nearest neighbor selection.

As regards routing shortcuts, we intend to tests other criteria for finding routing shortcuts (for example, a combination of ADC and EDC in order to exploit their advantages) and to observe the impact of the different Vivaldi's parameters and improvements on the detection results. At the end of this phase of the work, we will have chosen a criterion and an implementation of Vivaldi (improvements integrated, values of the parameters, ...). For any path in the network, we will be able to provide a list of nodes that are probably shortcuts for it.

The next step will be to try and rank the nodes of the list. For some paths, there exists hundreds of shortcuts and providing a list with hundreds of nodes is not really useful: we want to provide only the best or, at least, one of the best shortcuts for the given path. Obviously, the last step of the work will be to use the detection results order to improve the quality of the routing in the network.

Chapter 4

Routing resilience use cases

The purpose of this section is to document techniques that improves the quality of the recovery process by proposing:

i) a selective Routing Information Base (RIB)/Forwarding Information Base (FIB) update process towards router's line cards. The preferential selection criteria is the number of times the corresponding forwarding entries are looked up in the line cards' Local FIB (LFIB) i.e. the utilization rate of the forwarding table entries. By observing that some prefixes "carry" more traffic than others the concept developed here consists in determining the amount of traffic (i.e. number of packets) each entry "carries" so as to deduce a traffic-informed preferential selection criteria for the forwarding entries resulting from the RIB/FIB update process. The latter process involves typically a large amount of entries when after (local) failure occurrence henceforth the update process is not atomic: multiple update quantum of times shall elapse before the forwarding table entries are fully updated after such event. The non-local failure case requires further investigation and is not addressed in the present version of this document.

ii) upon failure occurrence, the selection of recovery link(s) that may themselve(s) be under failure but not yet known as such by a router processing a Link State topology update results in situation where re-routing action does not "recover" traffic but further delays its recovery. This phenomenon typically occurs when two (or more) links share a common risk. The proposed mechanisms consists here in detecting and identifying the set of links sharing a common risk. The purpose is to avoid selection of the links (sharing a common risk) during routing table entries re-computation after detecting the failure occurrence of one of their members by means of a probabilistic model: link failures can probabilistically be part of larger failure events involving multiple links i.e. single link failure events do not account for 100of the failure cases.

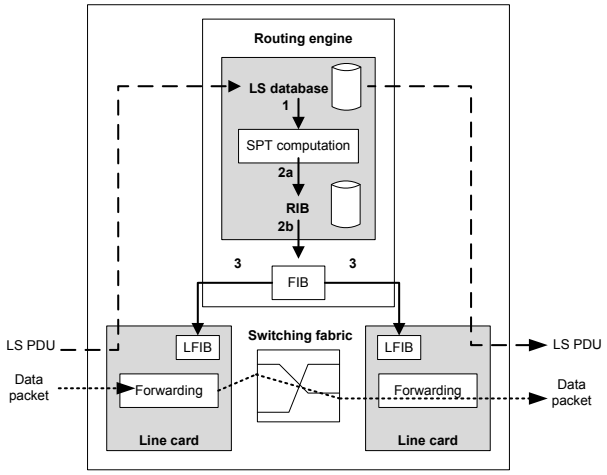


Figure 4.1: The router update process

4.1 Minimizing packet loss during re-routing

4.1.1 Formalisation of the technical problem

Routers running Link State (LS) routing protocols, flood LS PDUs (LS Protocol Data Units) over the network. These packets contain information about the local links and MA (multi-access) networks a router is connected to. All received LS PDUs are collected into a database (*the LS database*) which allows a router to have a view on the network link topology and to calculate shortest paths towards different destinations (IP addresses) or network parts (IP network prefixes). The LS database is updated by either detecting a local connectivity change (e.g. failing link or interface), or by receiving an LS PDU from a peering router. Two types of LS PDUs can be received:

- *LS Refresh*: A node in the network has sent a refresh of its status, involving no changes in the network.
- *LS Update*: A node in the network has detected a state change in its link connectivity to an adjacent node or network (addition or removal).

Only the second option is of interest for us, as it is the direct trigger for the router update process which we intend to investigate. The resulting update process can modeled in three steps (see Figure 4.1):

1. Re-computation of the *shortest path tree* (1), based on the updated LS database.
2. Update of the *central Routing Information Base (RIB)* and the *central Forwarding Information Base (FIB)*, based on the shortest path computation (2a and 2b).
3. *Distribution of central FIB* towards the line cards' local FIB (LFIB) (3).

The recomputation of the shortest path tree is usually optimized to be recalculated in its entirety and takes about 30 to 50 μs per destination prefix. Optimizations can be done using incremental SPF (iSPF) calculation schemes (see [24] and [25]). The second step consists of updating the central RIB and FIB, using the calculated shortest paths. This uses about 50 to 100 μs per destination prefix (see [26]). Typically this step happens in (pseudo-)parallel with step 3, which is about distributing the central FIB entries towards the line cards' LFIB. Running step 2 and 3 in (pseudo-)parallel, means that they both use the central CPU in interleaved time slots, swapping between both processes for updating and distribution. This process can be compared to the usual process scheduling in time-sharing OSes such as Linux, whereas commercial routers make use of a hard real time OS. The consecutive time the central CPU is spending to a task of central RIB/FIB updating or line card distribution is determined by the used quantum of the swapping process. The quantum time can typically be configured between 10 and 800 ms (for Linux, see [27]). Similar quantum time values were found in [26].

In practice the update process consists of a series of update-distribution batches, where in a first quantum a fixed set of prefixes are updated towards the central RIB/FIB, followed by a quantum where the same set of prefixes is distributed towards the LFIBs. By default, the cardinality of the set (the number of prefixes that are updated or distributed during a batch) is fixed during the update process. This is shown in Figure 4.2, and will be further quantified in the next section.

The process described can be formalized using the following naming conventions and modeling assumptions (see Figure 4.1):

- The set of *affected*[1] *traffic flows*[2]: $F_n = \{f_1, ..., f_n\}$ in MB/s.
- The ordered *associated traffic flow rate* $bw(f_i) : F_n \rightarrow R$ of the affected flows in MB/s.
- The *update time* t_u: the time needed to update one prefix in the central RIB/FIB.
- The *distribution time* t_d: the time needed to distribute a prefix from the central FIB towards the line cards.
- The number x_u of *prefixes that are updated/distributed* in one batch
- The *update quantum* resulting from updating x_u prefixes is the time $t_{qu} = x_u.t_u$.
- The *distribution quantum* resulting from distributing x_u prefixes from the central RIB/FIB towards the line cards is the time $t_{qd} = x_u.t_d$.
- The *swapping time/cost* t_s between interleaved quantums.

An example using the introduced terminology can be found in [7].

[1] Flows are 'affected' if their routing is influenced by the received LS Update

[2] A traffic flow is a function $f : V \times V \rightarrow \mathbb{R}$ which attaches a load to every edge of the network. The function is constrained by a capacity constraint, a flow symmetry constraint and a flow conservation constraint. (see [28]).

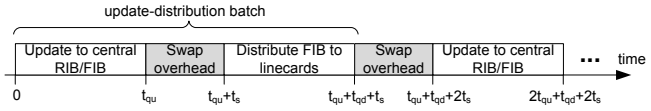


Figure 4.2: Timeline of the router update process

Recovery time and recovered traffic The recovery of a set of traffic flows (i.e. traffic directed towards a set of destination prefixes comprised as part of different RIB/FIB entries to be updated) is completed when the last corresponding RIB/FIB entry has been updated and distributed from the central RIB/FIB to the line card. The corresponding point in time where the entire set of affected traffic flows is recovered, is referred to as the *total recovery time*. A single traffic flow can be considered as recovered, once its corresponding RIB/FIB entry has been both updated and distributed from the central RIB/FIB to the line cards.

The recovery time for a single traffic flow, given a constant x_u value, is thus dependent on the update-and-distribution batch b_i comprised in it. Only when the corresponding batch (and all previous batches) are completed, the flow has been recovered. Given n flows and the fact that only x_u flows can be updated in one batch, the batch number b_i is determined by $b_i = \lceil i/x_u \rceil$. A traffic flow is recovered after all previous batches have been updated and distributed, having a swapping overhead on every load of a process. This results into the formula for the recovery time of flow f_i.

$$r(f_i) = b_i.(x_u(t_q + t_d) + 2t_s) - t_s$$

Packet loss As long as an affected traffic flow is not recovered upon a failure, packet loss occurs (as packet are still forwarded using the entry computed using the "outdated" routing information). The loss is proportional to the throughput of the traffic flow during the event of the update. This means that the total packet loss during the switchover operation of all traffic flows is proportional to the product of the recovery time and their corresponding average flow rate during the router update process[3]. In other terms, the drop in average capacity consumed by each outgoing flow (i.e. average bandwidth consumption), can be equated to losses:

$$loss(F_n) = \sum_{i=1}^{n} r(f_i).bw(f_i)$$

[3]This assumes that we exactly know the average bandwidth used during the router update. Strictly speaking it is proportional to the the amount of packets (taken in byte length) for the corresponding prefix lost during time between the failure occurrence and the FIB entry update times. The use of an average rate is a first-approach simplification that requires further validation (traffic burstiness plays an important role in the estimation). It is also important to notice that the loss per prefix occur independently of the preferential RIB/FIB update sequence but determine by means of i) average rate estimation per prefix (prefix preferential selection using cumulated statistics) with the assumption that the observed average will persist during the recovery period, ii) rate prediction from prior samples the associated rate to the each of prefixes during the "recovery period" (short-term prefix preferential selection). In practice, the bandwidth accounting is an approximation using binning techniques. At the same time this approximation is the estimation of the needed (available) capacity on the alternate set of one or more link(s) towards the destination.

Goal of the use case The section above illustrates that the order in which traffic flows are updated, can be of utmost importance for minimizing the associated packet loss of the update operation. Randomly updating RIB/FIB entries, as is usually the case, can result into the situation where higher bit rate flows need to wait on others before they are updated, resulting in high cumulative loss. The goal of this use case is thus to *minimize packet loss during the switchover operation* by *predicting short-term traffic bitrate trends* (i.e. bitrate trends per routing table entry hence per IP destination prefix) in order to determine which RIB/FIB shall be preferentially updated.

4.1.2 Machine Learning techniques/algorithms used

The techniques used to tackle the use case can be of three types:

1. Techniques to re-order the set of (aggregated) traffic flows[4] (typically with respect of there bitrate)
2. Techniques to optimize the order, optimize and steer the quantum batch size
3. Techniques to predict short-term variations of traffic flows

Traffic flow sorting (type 1) Once average rate data about the traffic flows is known, the most obvious way of using this information is to sort F_n in descending order of average flow rate. The resulting set $O_n = \{o_1, ..., o_n\}$ can now be defined as follows:

$$\forall o_i : \exists j \in \{1, \ldots, n\} : o_i = f_j$$

$$i > j \Rightarrow bw(o_i) \geq bw(o_j)$$

Upon the RIB information update, the corresponding loss can thus be deduced and used as a mean to minimize average traffic losses during the central RIB/FIB to LFIB update process. It is now easy to see or to prove that $loss(O_n) \leq loss(F_n)$.

Optimizing the number of prefixes in a quantum (type 2) The previous sections (i.e. 4.1.1) made clear that the effect of a fixed x_u and related quantums during the router update process can lead to a higher impact than expected. The x_u acts as a sort of a bin packer for the updates/distributions to be executed. Using a given F_n (being sorted or not), this section evaluates the possible gain of allowing x_u to be dynamically determined per update-distribution batch and optimized depending on the specific character of F_n.

The goal of this configuration strategy is to minimize the function $loss(F_n)$ by finding the appropriate sequence of sets of prefixes. Instead of a single fixed x_u this results

[4] FIB entries correspond with IP destination prefixes, the resulting entity to order is thus a set of traffic flows, further referred to as *traffic flow aggregates*

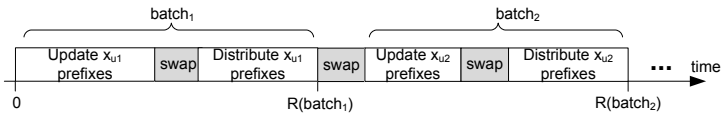


Figure 4.3: Timeline for router updates having a variable x_u

into a sequence $X_u = (x_{u1}, x_{u2}, \ldots, x_{ul})$[5] of the number of prefixes to be updated during every adjustable quantum. Associated to these sequence is the sequence of update-distribution batches $B_u = (batch_{u1}, \ldots, batch_{ul})$. This means that every f_i is contained in an update-distribution batch of B_u. Unfortunately the smaller the x_{u_i}'s are chosen, the more swapping time is sneaking into the total recovery time (see Section 4.1.1) of the router update, possibly resulting into additional loss. We have thus to consider an adaptive x_u setting, that minimizes the packet loss by keeping the total recovery time as small as possible (update-distribution batches should be as large as possible to reduce swapping time). However, one should also avoid that the recovery of certain traffic flows is delayed longer than needed, because the longer the update-distribution batches are, the longer prefixes need to wait until their update is both updated and distributed, and thus recovered.

The associated recovery time of a flow f_i is again determined by the $batch_k$ to which f_i is attached. However, contrary to the definition in 4.1.1, the length of update-distribution batches is not fixed anymore. Therefore, one needs to find first in which batch a flow f_i is contained (for further details see [7]).

We can observe that in the middle of the ordered process of updating F_n, with our current batch containing a set of flows to be updated $b_{current} = (f_i, \ldots, f_{i+s})$[6], we have two options:

1. Extend the current batch with the next flow f_{i+s+1} (*extension*)
2. Finish the current batch and put the next flow into a new update-distribution batch (*splitting*)

We now can can compare the additional cost of extension vs. the additional cost of finishing the update-distribution batch to guide us into the decision above.

The extension cost ec_{is} can be formulated as follows:

$$\begin{aligned} ec_{is} &= (t_u + t_d)bw(f_{i+s}) + \ldots + (s+1)(t_u + t_d)bw(f_i) \\ &= (t_u + t_d)\sum_{t=i}^{i+s}(i + s - t + 1)bw(f_t) \end{aligned}$$

The formula above expresses the fact that, by extending the current batch, the recovery time of every flow in the current batch will result into an additional delay compared

[5] X_u is actually an integer partition of n. his means that $\sum_{i=1}^{l} x_{ui} = n$.

[6] Flows prior to f_i have already been updated in an ordered manner (f_1 to f_n)

to the minimal delay it can experience (compared to the earliest recovery time[7]). That additional delay when multiplied with the associated bandwidth allows to deduce the additional loss caused by the update-distribution batch extension. For example, the recovery of the first flow f_i in the given batch was already delayed with s update-distribute batches (as it was not directly distributed but put in the same batch of s next flows), and by adding an additional element (extending the batch), this operation will delay it with an additional update-distribution batch. On the contrary, the recovery of the last flow of the current batch will only be delayed with one update-distribution batch in case of extending the current batch.

Finishing the current batch on the other hand, also has an associated cost, as it will introduce additional delay for the coming flows, resulting from the additional swapping cost. This termination condition can be formulated as follows:

$$fin_{i+s} = 2t_s \sum_{t=i+s+1}^{n} bw(f_{i+s+1})$$

Our configuration strategy now consists in identifying the action with the least associated cost. The actions being: extending the current batch if $ec_{is} < fin_{i+s}$, or finishing the current batch in the other case. This action scheme can be applied recursively until the update of F_n is finished. An example of applying this heuristic can be found in [7].

Techniques for predicting short-term bitrate consumption (type 3) Predicting the trend (more precisely the variation over time) of the bitrate of traffic flow aggregates can be formulated into two ways. It can be formulated either as a regression problem (predict the exact or relative bitrate increase/decrease) or as a classification problem (the class of exact/relative bitrate increase/decrease). Predicting these variations can be performed on a long-term or a short-term. As stated above the purpose here is to predict short-term variations.

Clustering techniques such as Self Organizing Maps (SOMs) will be tested in order to subdivide the input space such as to compress traffic flow monitoring information. A SOM is an unsupervised machine learning method often used for statistical data analysis and data clustering. This type of artificial neural network was first described by Teuvo Kohonen (see [29]). In essence, SOMs map the input space to a lower-dimensional space (projection on the target space) which is easier to comprehend. This mapping is made such that it preserves the statistical structure of the input space and in addition preserves certain topological properties of the input space. More information about this can be found in [29].

Support Vector Machines (SVMs) and Support Vector Regression (SVR) techniques will be tested as supervised learning methods for predicting the class or the exact number of the bitrate increase of traffic flow aggregates (time series prediction). Support Vector techniques project data into a high dimensional space by the use of (possibly

[7]The earliest recovery time for a flow is when it is the last flow in an update quantum having no earlier update quantums.

non-linear) basis functions. In the projected space these techniques search for the separating hyperplane that maximizes the margin between instances of the classes of the *classification problem*. In case the space is not separable, the concept of soft-margins extends the applicability of the approach. Only support vectors, vectors that are on the edge of class separation within a predefined margin, determine the final SVM configuration. Similarly in the context of *regression*, those (support) vectors within a tube of predefined margin, determine the configuration of the SVR. More technical details about SVMs and SVR can be found in [30] and [31].

Reservoir Computing will be tested as a supervised learning technique for predicting the class or the exact number of the bitrate increase of traffic flow aggregates (time series prediction). RC projects the input space on a high dimensional space represented by a recurrent network (the reservoir). Supervision happens by learning a (typically linear) combination of the reservoir signals to be mapped to the target values (labels). More details about RC can be found in [32].

4.1.3 Implementation

Input The machine learning component involved with optimizing the router update process requires the following data in order to function adequately. These data are extracted from the captured IP packets:

- timestamp of the packet: the time at which the monitoring point captured the packet
- the packet size (in bytes)
- the IP source and destination address extracted from the IP header
- the protocol field extacted from the IP header
- the Time-To-Live-field (TTL) extracted from the IP header
- the TCP source and destination port number (UDP source and destination port number) extracted from the TCP (UDP) header
- the TCP flags extracted from the TCP header

Output The machine learning component will try to minimize the packet loss by either:

1. Perform a prediction on the estimated bitrate per traffic flow aggregate. In this case the predictions will be either made by means of a classification (integer value) or by an absolute estimate (real value, see previous sections).
2. Return a suggestion regarding both:

(a) the order of the traffic flow aggregates to be updated (traffic flow aggregate sequence)

(b) the size and the order of the update quantums and the corresponding traffic flow aggregates (sequence of sets of traffic flow aggregates)

4.1.4 Experimentation

Performance objectives and evaluation criteria The performance of several techniques is primarily measured in terms of decrease in packet loss (see 4.1.1). In order to be able to evaluate the performance of different machine learning techniques for time series prediction or bitrate trend prediction of traffic flow aggregates (Section 4.1.2), the following error measures are used:

- (regularized) Mean Square Error (MSE):

$$MSE(reg) = \frac{\sum^{n}(prediction - target)^2}{n}(+modelcomplexitypenalty)$$

- Mean Absolute Error (MAE):

$$MAE = \frac{\sum^{n}|prediction - target|}{n}$$

- Pearson's Correlation coefficient (R):

$$R = \frac{Cov(prediction, target)}{Var(prediction)Var(target)}$$

- Signal-to-noise ratio (SNR):

$$SNR = \frac{E(prediction^2)}{Var(error)}$$

- Variance-to-noise ratio (VNR):

$$VNR = \frac{Var(prediction)}{Var(error)}$$

The Mean Absolute Deviation (MAD) between several sample sets will be used as an additional indication on stability of the techniques.

Methodology: Scenarios and Tools The router update process itself is simulated in a custom C++ environment which is able to take into account the given input variables such as to evaluate the resulting packet loss and needed recovery time for a given packet trace (PCAP-file) of an IP router as available by the MAWI working group ([33]). For evaluating the machine learning techniques mentioned in Section 4.1.2, MATLAB is used in combination with available toolboxes (Neural Networks, Statistics, System Identification, e.o.) and libraries (e.g. LIBSVM).

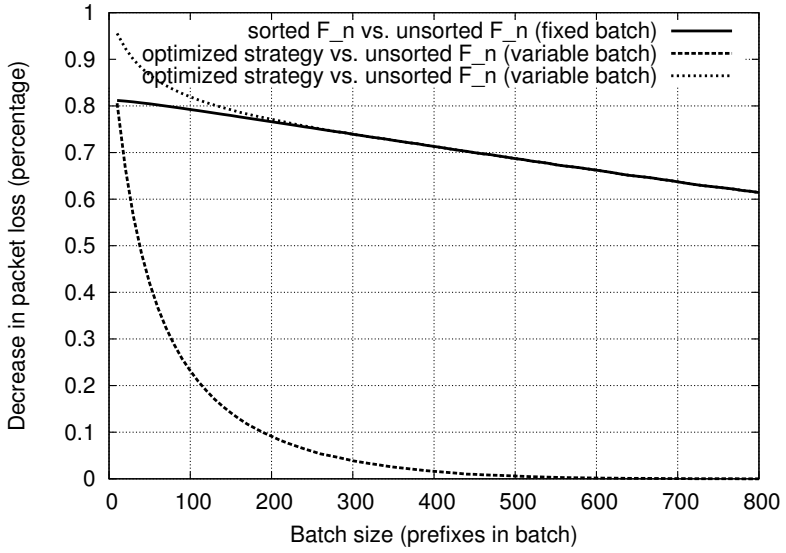


Figure 4.4: Decrease in packet loss vs. (minimal) quantum size

Experimental results Optimizing the quantum size and the update order of the update process has shown to enable significant decreases in packet loss during the switchover process of an IP router. The gain of sorting the traffic flow aggregates and/or using a heuristic for optimizing the number of prefixes in a quantum dynamically is shown in Figure 4.4. Further results can be found in [7].

Future Work Currently, the gain of using SOMs, SVMs, SVR or Reservoir Computing is being evaluated.

4.2 Data mining with OSPF updates to identify shared risk link group (SRLG)

4.2.1 Formalization of the technical problem

From section 4.1, we know that LS updates are used to notify state changes in the link connectivity for a router with its adjacent node or network. We also know how the local RIB and FIB entries are updated with the re-computation of the shortest path tree once a link failure is notified through LS update. However, in a network there might exist shared risk link group (SRLG), i.e., set of links whose failure occurs simultaneously. For example two separate logical links might share the same physical link or fiber optic cable (both links share the same risk). Therefore, any form of cable cut for that particular physical link may manifest two simultaneous link failures and eventually the two links are part of the same SRLG ([34], [35], [36]). Identification of SRLGs in the network topology reduces the failure recovery time and therefore total amount of packet losses ([34], [35], [36]). This is because, otherwise, each link failure within an SRLG

will introduce a separate failure recovery process. The proposed probabilistic model does not assume that single link failures account for 100 percent of the failure cases but that some link failures are resulting from occurrences of events affecting more than one link (i.e. typical occurring for links sharing a common risk). The problem here thus consist in inferring from the observation from a minimal set of topological changes (link state updates) whether they affects a single link or a set of links including that link. Formally, this means that observing a topological change for link X can either mean link X might have failed independently of link Y or that Link Y might have also failed knowing that X failed (or the other way around depending on the observation sequence) without having received the topology update for link Y. Thus for a router, if the member links of an SRLG are known, it can locally infer the probabiliy of failure of other member links of that particular SRLG by observing a sample of the topology updates resulting from the common failure. Hence, the router can perform the shortest path re-computation once for all possible simultaneous failure scenarios and update the corresponding RIB, FIB entries in a single round instead of repeatedly invoking the same process for each individual failure.

Therefore, detection and identification of SRLG has become an important aspect of network protection and restoration. There exist several methods for SRLG detection and identification as well as association of SRLGs (e.g. shared risk link grouping technique [37] [38]). However, the methods investigated so far are dependent on underlying physical network, e.g., optical network and requires cross layer information transfer among the network nodes. The process eventually complicates the routing protocol and waste network bandwidth for SRLG exploration. Instead, in this particular use case, we propose a novel methodology to use machine learning technique to detect and identify SRLGs in a network. In our proposed scheme, we investigate the LS updates and the correlation among themselves to learn about the possibility of the existence of SRLGs in the network and their identification. The process requires no knowledge about the physical topology and does not inject any control traffic in the network. Our method requires only a machine learning engine (MLE) in the router which passively gathers information from OSPF traffic (LS updates in particular). The Machine Learning Engine (MLE) runs an online training algorithm that progressively learns (i.e., detects and identify) about the existing SRLGs in the network. The MLE unit then informs the router information base about its findings and help router to identify SRLG failure scenarios in order to take appropriate protection measure.

4.2.2 Machine learning technique/algorithm used

There are two distinctive phases for our proposed algorithm as follows:

1. Learning Phase: we use statistical method to train the MLE regarding SRLGs. We introduce a modified Bayesian network ([39], [40]) for training. The following example describes the method in detail. We assume a simple network as shown in Fig. 4.5.

Suppose a particular node observes the following LS update pattern given in Fig.

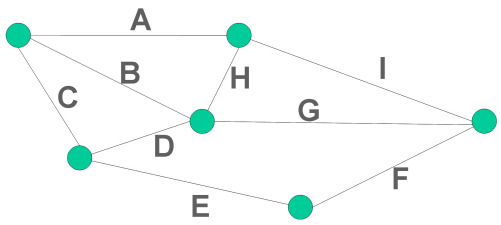


Figure 4.5: Example network with links A, B, ..., I

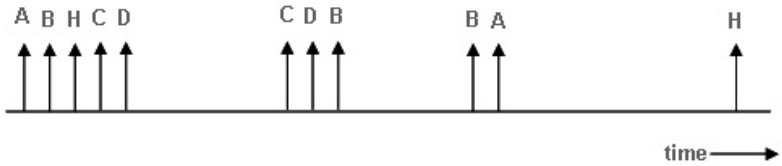


Figure 4.6: LS update pattern with respect to time

4.6 where each of the arrow indicates an LS update corresponding to a particular link failure.

It is clear from Fig. 4.6 that some of the LS updates can be grouped together, which indicates with a certain probability that they may topological information they describe forms a common SRLG. However, we have to keep in mind that a certain number of links may form the same SRLG irrespective of their LS update arrival order. Also same group may contain multiple SRLG that happened to occur simultaneously. Moreover, same link may be a member of multiple different SRLG. Our statistical machine learning algorithm should be able to address all of these above described problems. For this purpose, we introduce an adaptive hierarchical Bayesian tree (AHBT) for learning the existence of SRLGs.

In the initial phase, we assume none of the links forms any SRLG (except their own SRLG). Also each of the links forms a unique node in the Bayesian tree shown in Fig. 4.7. Every node in the Bayesian tree is associated with a counter that stores the number of occurrence for the associated link. We assign a time threshold (refer Fig. 4.5) to group LS updates. Each group has a unique position in the Bayesian tree hierarchy. Groups with 2 links are positioned in the second tier as shown in Fig. 4.2.2. Similarly groups with 3 links are placed in tier 3 (refer Fig. 4.2.2)) and so on.

There are two types of transition possible from each node in the tree.

The tree nodes are associated with transition probability. A transition from a node to itself and a transition from a node to its parent. The transition probabilities are computed using the counters from the associated nodes. The update of the transition probabilities are shown in Fig. 4.7. Here the transitions from A to AB and B to AB are treated separately. However, ordered pair AB and BA is merged into the same group [A, B]. In the learning phase, whenever a group is identified, it is placed in the appropriate hierarchy of the Bayesian tree and all its associated children's transition probability, as well as the counters get updated.

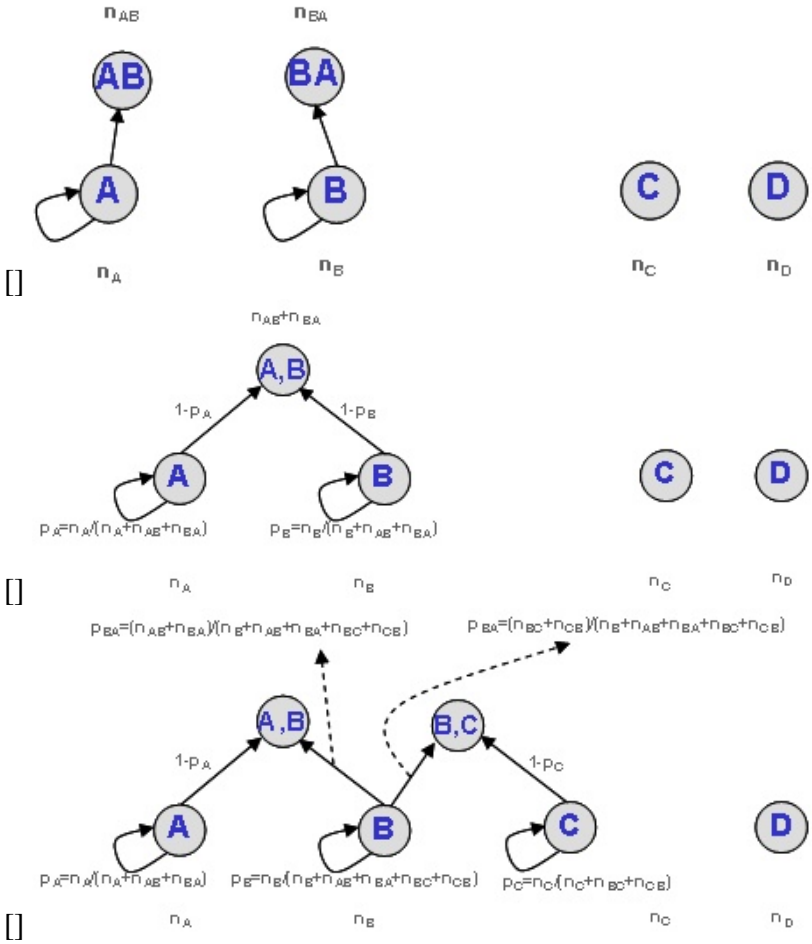


Figure 4.7: AHBT - Example machine learning steps for SRLG

2. Decision making Phase: In this phase a threshold value is assigned for decision making (pth). For a given observation A, probability that ABC forms a SRLG is computed as follows:

$$p(ABC/A) = p(A \rightarrow AB)p(AB \rightarrow ABC) + p(A \rightarrow AC)p(AC \rightarrow ABC)$$

Similarly $p(AB/A)$ and $p(AC/A)$ is also computed. If $p(ABC/A) > p^{th}$ then links A, B and C are considered to form an SRLG. Otherwise if $p(AB/A) > p^{th}$ but $p(ABC/A) < p^{th}$ then links A and B are assumed to be within same SRLG.

The information identifying the SRLG then be transferred to the routing engine for pruning the topological database before re-computation of the RIB entries. The corresponding data structure includes the list of links associated with a unique identifier. Further modification: Our proposed scheme works fine if the time threshold for grouping LS updates is optimal. Determining optimal time threshold is extremely difficult. We aim to improve the process by computing the transition probabilities as a dependent variable which varies with the correlation between inter arrival times (a) between same links and (b) between different links as shown in Fig 4.8.

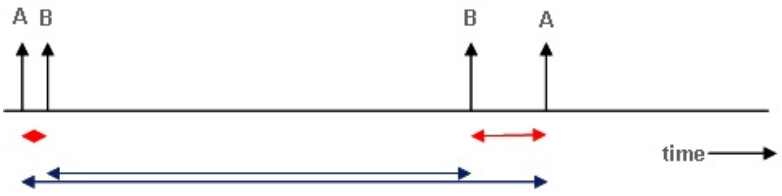


Figure 4.8: Inter arrival time depiction

Chapter 5

Profile-based accountability

5.1 Formalization of the technical problem

In this section, we discuss the role of the profile-based accountability use case within the context of congestion and fairness. In this context, profile-based accountability focuses on a local approach to subscriber accountability. Similar to [41] and [43], the proposed approach goes beyond individual flows by tracing subscriber traffic rates and by characterizing them into account the local congestion the network incurs. Different from [41] and [43] is that we look on the local level rather than on the Internet-wide level. As a result, we are less concerned with the assumptions needed for an Internet-wide solution (e.g., re-ECN). Instead focus on what can be derived from the locally available information instead of considering an ideal solution is available. Our approach consists in monitoring the aggregated traffic of individual users/subscribers over time and congestion created by these users/subscribers. The accumulated information of monitoring serves as input to a machine learning algorithm.

The aim of profile-based accountability is to 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. There are two major tasks expected from the machine 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. Each profile indicates specific network traffic behavior associated with the usage and demand on the network resources. 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.

5.2 Machine Learning

Profile-based accountability is divided into two parts, profiles can categorize *subscribers* or profiles can be modelled to characterize *actions.* Subscriber profiles characterize the household network resource usage pattern. Action profiles (computed with respect of subscriber profiles) defines the behavior on a restricted set of activity a subscriber undertakes within a certain period of time.

For this purpose, two distinct machine learning techniques are considered. A clustering technique based on the Fuzzy K-Mean algorithm is used to detect and identify commonalities in users/subscribers *actions* (unsupervised learning). A Hidden Markov Model (HMM) is learned for the supervised classification of subscribers/users (supervised learning).

5.2.1 Fuzzy K-Mean

Fuzzy clustering algorithms are mathematical models for detecting similarities between elements of a group of objects. Information about the objects to be analyzed is input to the algorithm in form of d-dimensional vectors. The vector represents a particular class of object which has as its components d features of the objects. The output of the algorithm yields data vectors that are assigned to the same cluster centers.

Let $X = \{x_1, \ldots, x_n\}$ be a set of *n* vectors in R^d representing the data. A fuzzy clustering of *X* into *K* clusters consists of functions $\{u_1, \ldots, u_c\}$ where $u_i : X \rightarrow [0, 1]$ and $\sum_{i=1}^{c} u_i(x) = 1$, for all $x \in X$. These functions are called membership functions which has value between 0 and 1. The Fk-M algorithm is designed to produce a fuzzy clustering in the same way as the K-Means algorithm is meant to produce hard clusters, through the minimization of the objective function:

$$\sum_i \sum_k (u_{ik})^m \left| x_k - v_i \right|^2$$

where u_{ik} is the value of the i^{th} membership function on the k^{th} data point x_k. The vectors $\{v_1, \ldots, v_c\}$ are the clusters centers. In order to minimize the objective function, the cluster centers and membership functions are designated so that high memberships occur for points close to the corresponding cluster centers. The minimization of the objective functions leads to the following set of equations

$$v_i = \frac{\sum_k (u_{ik})^m x_k}{\sum_k (u_{ik})^m}$$

and

$$u_{ik} = \frac{\left(\frac{1}{|x_k - v_i|^2}\right)^{1/(m-1)}}{\sum_i \left(\frac{1}{|x_k - v_i|^2}\right)^{1/(m-1)}}$$

There is no closed form solution for these equation but they serve as the basis for an interactive procedure which converges to a local minimum for the objective function. Clustering technique has being successfully applied for flow and network application identification [44], [45], [46].

5.2.2 Hidden Markov Model

Hidden Markov Model (HMM) is a type of finite state machine having a set of initial state probabilities (π), transition probabilities (A), hidden states (Q), output probabilities (B), and output alphabet or observations (O). An HMM model is said to be a triple $\lambda = (A, B, \pi)$ whereas the states Q, and outputs O are understood and each states produces an output with a certain probability (B).

The definition of HMM in terms of this triple is:

- A is the probability that the next state is q_j given that the current state is q_i. Thus, $A = \{a_{ij} = P(q_i \; at \; t+1 | q_i \; at \; t)\}$, where $P(a|b)$ is the conditional probability of a given b, $t \geq 1$ is the time, and $q_i \in Q$.
- B is the probability that the output is is o_k given that the current state is q_i. Formally, $B = \{b_{ik} = P(o_k|q_i)\}$, where $o_k \in O$.
- $\Pi = \{p_i = P(q_i \; at \; t = 1)\}$ gives the initial probabilities.

Three canonical problem are associated to the understanding of how HMM operates and its application as a supervised machine learning technique.

1. Given the observation sequence $O = \{O_1, O_2, \ldots, O_T\}$ and the model parameters $\lambda = (A, B, \pi)$, compute the probability of a particular output sequence $P(O|\lambda)$. This problem is solved by the Forward and Backward algorithms.
2. Given the observation sequence and the model parameters $O = \{O_1, O_2, \ldots, O_T\}$ and the model parameters $\lambda = (A, B, \pi)$, find the most likely sequence of states $Q = \{q_1, q_2, \ldots, q_T\}$, which could have generated a given output sequence. This is solved by the Viterbi algorithm and posterior decoding.
3. Given an output sequence, find the most likely model parameters $\lambda = (A, B, \pi)$ and output probabilities $P(O|\lambda)$. The solution is given by the Baum-Welch algorithm.

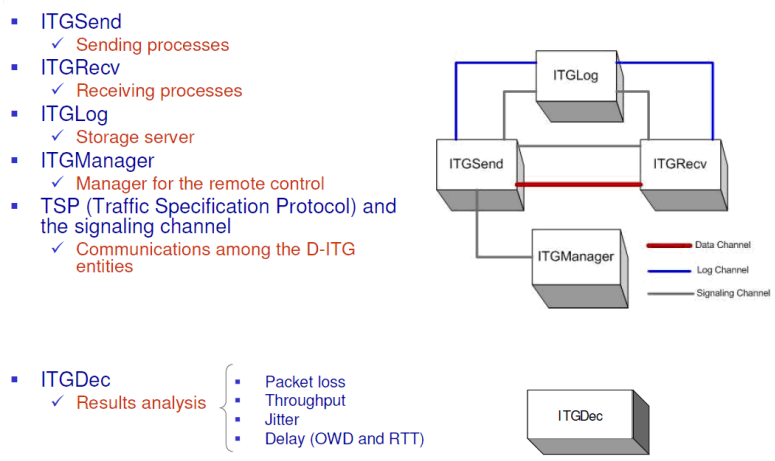


Figure 5.1: Structure of the D-ITG modules and flow operation

The solutions of these problems are at the core of HMM as a machine learning method. Mathematical details of the HMM, demonstrating the solution of this problem and how they can be applied to be used as a supervised learning techniques can be found in the literature [47], including application to network problems [48].

5.3 Implementation

The initial phase of the profile based accountability relies on simulations to validate the dual approach of classifying action profiles and identifying subscribers profiles (described in D21). Traffic is generated using the D-ITG traffic generator (see). It is generated from one server (ITGSend) toward a receiver server (ITGRecv), which in its turns sends a logging record towards a third server (ITGLog). This logging server is necessary due to the amount of traffic being simulated and the huge processing power required by the receiving server if it had to both receive the incoming traffic and log the flows.

This approach splits the whole problem in two parts one which requires unsupervised machine learning technique and the other that requires supervised learning. Initially, we modeled several distinct action profiles. Each profile consist of several flows generated according to some random process, for example, some long TCP and UDP flows or some sequence of several short TCP flows. Action profiles by definition represent a short time-sliced network behavior of a subscriber, several action profiles are simultaneous generated on the same network connection between ITGSend and ITGRecv, under different network condition, i.e., congested or not congested. Once this first part of the simulation is done an analysis of the log files, irrespective to each subscriber "network action" is analyzed and features extracted. The features should reflect the actions and network impact of the protocols and flows, measurements such as number of flows, average bit rate of the flows, or of the time-sliced connection, average packet jitter, packet drop, congestion or no congestion, etc. These measurements are used as input of a feature vector, which should characterized the network "action" the subscriber is

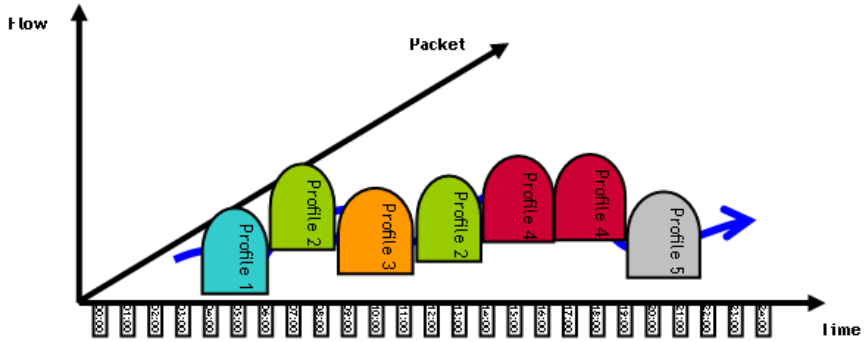


Figure 5.2: Diagram of the subscribers' action profiles over time

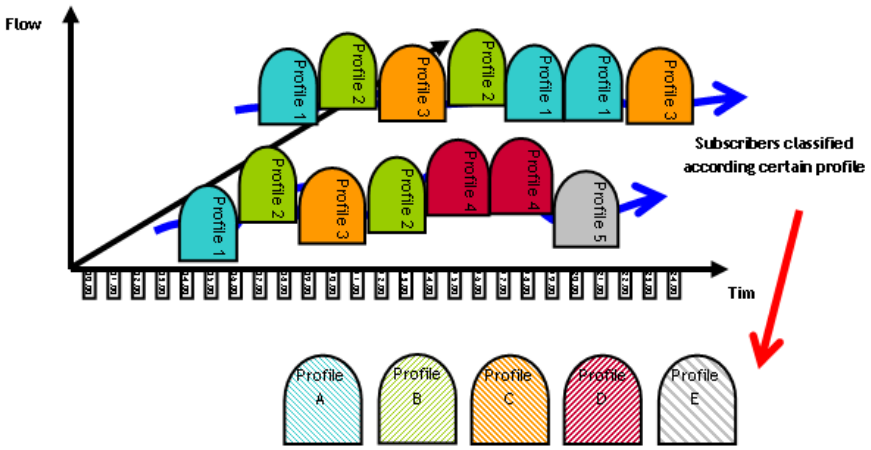


Figure 5.3: Diagram of the subscribers' profiles inferred from a time sequence of action profiles

undertaking during a defined time-sliced. Using clustering algorithm described on the last session, learning and classification of action profile are obtained, which will be used as the base to build a time series of action undertaking by subscribers.

A time sequence of action profiles by different uses over a period of time, possible weekly, will ultimately determine the subscribers'classification in our modeling. The definition of subscribes will depend on an expert and from the objectives network operators have, to create a basic criteria for profile to be learned and classified. In this simulation phase, we will synthetically create different time sequences, which will represent distinct subscribers to be used as a base for a learning set. Thus, using supervised machine learning algorithm such as HMM, a complete learning, classification and test cycle can be created to validate the overall modeling for profile-based accountability.

5.4 Experimentation

Focus is on characterization of the interaction between users and the network load, diversified on either short-term actions (resulting in action profiles) and aggregated, long-term subscriber behavior (resulting in subscriber profiles). Final goal (scenario 2

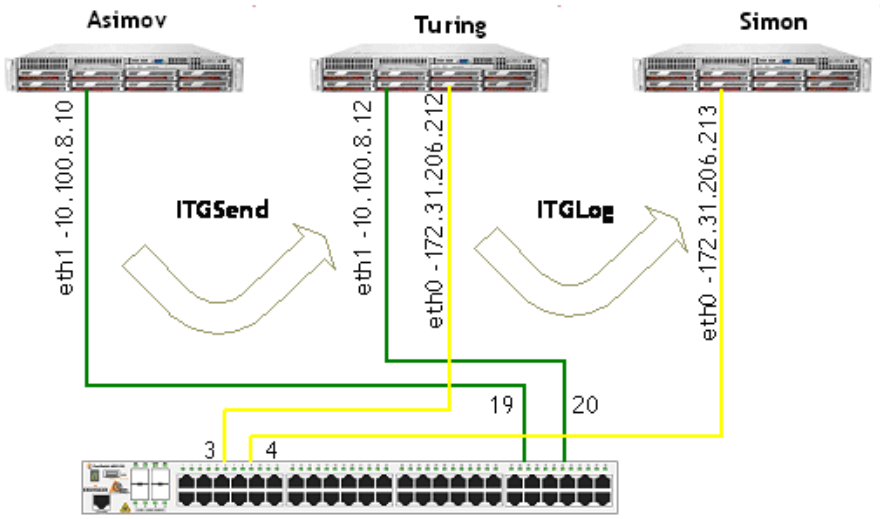


Figure 5.4: Experimental setup

in deliverable D2.1) is an online assignment of profiles to user actions and user subscriptions, by applying machine learning techniques to the packet stream associated with single flows, multiple flows, single users and multiple users. This also enables to define and validate feedback mechanisms to enforce users to remain within a given profile. As an intermediary goal until September 2009, we set as scope an offline, centralized, batch-mode version of our PBA-engine, without feedback, corresponding to scenario 1 in deliverable D31.

5.4.1 Performance Objectives and Evaluation Criteria

Traffic Emulating Service: We rely on D-ITG (http://www.grid.unina.it/software/ITG/) for traffic emulation of subscribers and background traffic. We run D-ITG on our lab network of four machines.

- **Inputs:** D-ITG ASCII scripts, representative for a wide variety of user behavior: different number of flows, UDP vs. TCP, bursty ON-OFF activity vs. continuous flow activity, and random vs. non-random start and stop of new flows. Number of users is expanded beyond the number of IP addresses in the lab by means of port number. The scripts are composed manually, but a program to generate/parse randomized scripts can be developed if useful.
- **Outputs:** Actual traffic over the lab network

5.4.2 Methodology: Scenarios and Tools

The list of other tools it interacts with: Currently, we experiment with the native D-ITG monitoring tool. Drawback of this is that it has no means of adaptive sampling.

In scenario 2, we envisage an active monitoring and sampling version of this, ideally by utilizing the results of the experimental use case (a1) of adaptive traffic sampling.

Traffic Sampling Service: For D-ITG-generated traffic, a D-ITG logging tool (ITGLog) is available. The resulting log file contains timing and packet size information for all packets, and can be further parsed with the D-ITG decoder tool (ITGDec), to isolate the features we focus on: bandwidth, packet delay, jitter and packet loss.

- **Inputs:** D-ITG logging files, in a D-ITG-native format.
- **Outputs:** CSV (comma separated value) ASCII files, with per-flow statistics on the desired time slot scale.

The list of other tools it interacts with: ITGSend and ITGRecv at input, output via ITGDec.

5.4.3 Experimental Results

Profile Learning and Classification:

- **Inputs:** CSV-files with statistics. Envisaged machine learning algorithm in this step is clustering. Statistics include bandwidth, packet delay, jitter and packet loss on a fixed time slot scale. Possibly, feedback can be given to adopt an optimal timescale.
- **Outputs:** Mapping of given scripts and users to different clusters.

5.4.4 Future Works

Perform the whole experimentation based on real traffic data.

Chapter 6

Recommendations for integration into common ECODE architecture

We will first summarize the ECODE Architecture described in Deliverable D2.1 and depicted in figure 6.1. Then we will describe how we currently view the mapping of all the use cases described so far onto this general architecture.

6.1 ECODE Functional Architecture Reminder

A standard router comprises ”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”. The routing engine includes a routing information processor and ”Routing Information Base”.

The RIB refers to the ”Routing Information Base”. It stores the routes and 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 refers to the ”Forwarding Information Base”. It 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.

In addition to the Forwarding, and Routing engines, ECODE introduces the Machine Learning Engine (MLE) and the Monitoring Engine (ME):

The ”Machine Learning Engine” (MLE): part of the control plane, aims as processing by means of learning methods, the input from the network (via forwarding and control components) to subsequently decide on forwarding and routing execution.

The ”Monitoring Engine” (ME): part of the forwarding plane, aims at collecting packet/flow/path performance information such as delay, bandwidth, packet loss, etc.

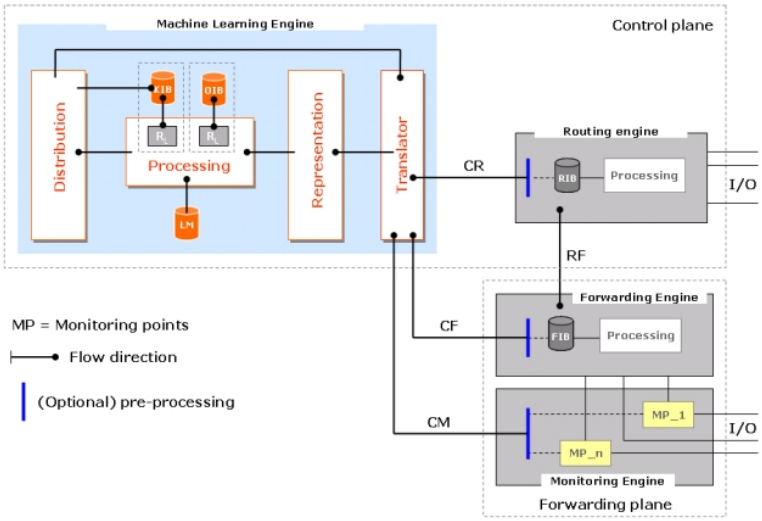


Figure 6.1: ECODE component framework

Once collected, this information is buffered in the Monitoring Data Register part of the Reporting module (see also Fig.3). The ME parameters such as the sampling parameters (e.g. sampling rate), and filtering parameters are controlled by the Machine Learning Engine (MLE) by means of the ME Controller. The functional description of the ME is further detailed here below.

The interaction between the various engines is performed through dedicated interfaces. We consider four distinct interfaces:

- RF (for Routing Ð Forwarding): through this interface, the Routing and Forwarding engines can communicate with each other and exchange information if requires.
- CR (for Cognitive Ð Routing): through this interface, the MLE may retrieve data from the Routing Engine and communicate to the Routing Engine the decision it takes.
- CF (for Cognitive Ð Forwarding): similarly to the CR, the MLE may retrieve data from the Forwarding Engine and communicate to the Forwarding Engine the decision it takes.
- 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-procesing 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 LM (for Learning Methods). 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. 6.1 provides a representation of the interface between forwarding and routing engines and the Translator function of the Machine Learning Engine (MLE). The interface CM is here depicted for a single line card comprising a set of N interfaces/ports, n of them (n =< N) being equipped with a Monitoring Point (MP). Thus, a node may comprise multiple instances of the CM interface. Fig.1 provides also a view of the elements composing the Machine Learning Engine (MLE). As illustrated, the latter comprises four different modules: the ”Translator”, the ”Representation”, the ”Processing”, and the ”Distribution”.

The ”Translator” comprises a.o. 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 ”Processing” includes the Learner and the Performer (functions) and 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 module 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,

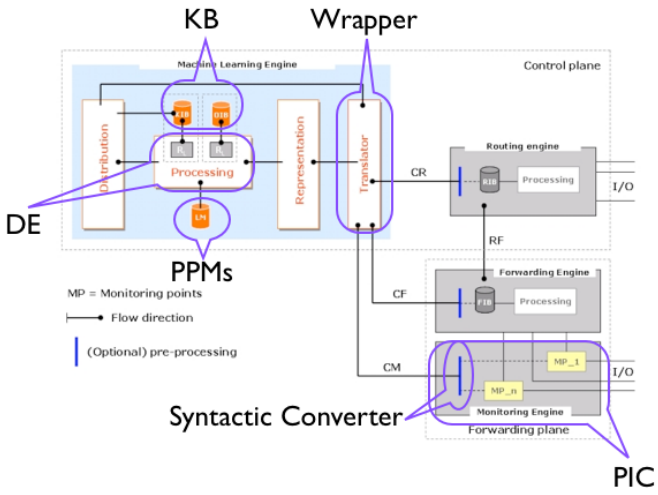


Figure 6.2: IDIPS integration into the ECODE architecture

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 an extra CC interface (not shown on fig. 6.1).

6.2 Use case b1: Path availability

6.2.1 Intelligent Path Ranking Using IDIPS

In this section, we describe how IDIPS, as defined in Chapter 2, will be integrated withing the ECODE architecture. This is depicted in Fig. 6.2.

Clearly, the PIC is located between the forwarding plane of a router, where MPs are implemented and the Machine Learning Engine. Indeed, MPs send the path information they measured to the Machine Learning Engine where the Transaltor is in charge of converting the output format provided by the MPs into a specific format that can be used by the Machine Learning Engine.

The KB, that stores path attributes is, actually, part of the Observation Information Base (OIB) within the Machine Learning Engine. Indeed, the OIB stores bounded sequences of observations that can be used for processing.

The various models used for preciting path performance metrics discussed in Chapter 2 are stored in the Learned Models (LM), within the Machine Learning Engine.

Finally, the DE corresponds to the Processing in the Machine Learning Engine.

6.2.2 Delay estimation and delay-based path selection and routing

The Internet Coordinate System presented in chapter 3 can be seen as a special routing engine, located in the RE block of figure 6.1. Indeed, routing entities that exchange their coordinates and measure delays between themselves to converge towards their own coordinates, participate in a routing protocol, whose outcome not the building of a graph topology, but instead the embedding/positioning of routing entities in a metric space. On this basis, delays between any node pair can be estimated. In the best case, for some node pairs only, the estimation is superseeded by an actual delay measurement. This happens between pairs of nodes that are neighbours in the ICS.

The machine learning techniques presented in section 3.2 have been used to derive a rule used by ICS nodes to improve their neighbour selection and thereby improve the ICS precision and stability. At this stage this rule, derived by this off-line centralized learning method, is basically integrated in the ICS, and therefore also part of the Routing Engine.

The routing shortcut discovery based on the ICS, explained in section 3.3.2 is also part of the Routing Engine. Classical shortest paths (e.g., Dijkstra) are here substituted by the criteria we have designed to find shortcuts, based on all estimated delays and some measured delays.

6.3 Use case b2: Routing resilience

6.3.1 Minimizing packet loss during re-routing

The functionality to implement the strategies as defined for this use case assume the following overall procedure within the ECODE component framework. A continuous process ensures that packet data is entering the monitoring engine which on its turn sends the required fields to the responsible module in the Machine Learning Engine. This is illustrated by the dashed lines and numbers roman capitals in Figure 6.3:

1. Incoming packets arriving on forwarding plane I/O
2. Selected data is forwarded from the Monitoring Point to the Machine Learning Engine (MLE). The data is structured as a set of timestamped ([IP Source Address], IP Destination Address) pairs, where the IP Source address is optional, together with a set of attributes Attributes. The input data can be seen as the set timestamp, ([IP SA], IP DA), Attributes.
3. The MLE builds up its model using this data: data is first processed by the translator (format translation) and then semantically processed by the representation module before being processed as observations by the MLE processor.

Besides this continuous process, the router update process is a triggered process following the following event chain:

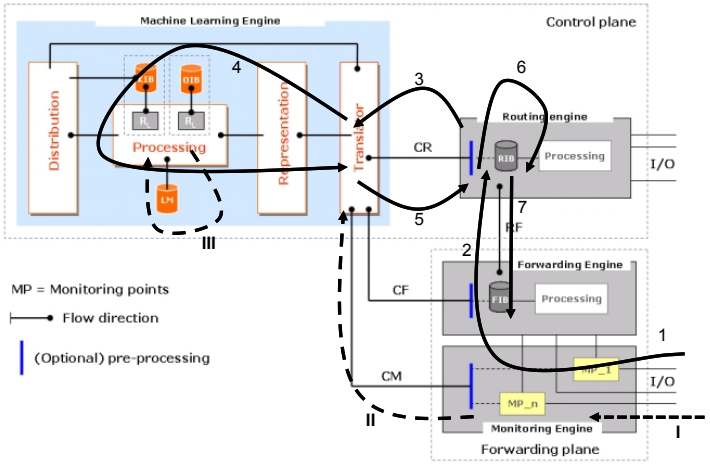


Figure 6.3: Integration of the router update process in the ECODE component framework

1. A (local) network failure is detected by the forwarding plane (remote failure detection will be covered when distributed machine learning techniques will be investigated)

2. The failure event triggers a change in the local neighbor relationship table over which routing adjacencies are established (maintained as part of the routing engine (RE)

3. The RE requests the MLE to provide the information required for the routing engine to perform preferential sequence during the RIB/FIB update process (e.g. prediction data)

4. The MLE processes the request

5. The MLE provides the RE with the needed information (the structure of the corresponding data is a preferential selection criteria per (set of) prefix).

6. The RE performs the RIB/FIB entries update process using the received information

7. The RE sends the renewed FIB entries to the forwarding engine

The above process is documented in Figure [7] with the full lines and arabic numbers. Note that step 3 to 5 can also happen by a process where the MLE continuously pushes information towards the RE.

6.3.2 Data mining with OSPF updates to identify shared risk link group (SRLG)

The functionality to implement the MLE algorithm as defined above assumes the following overall procedure within the ECODE component framework. Refer Figure 6.3 of Section 4.1 for further details:

1. OSPF LS updates are filtered and reports delivered to the translator.
2. MLE processes the sequence of LS updates after syntaxic translation and semantic transformation into events processable
3. The MLE processor which runs the Bayesian tree to identify SRLG (sequences can be stored in the OIB for subsequent processing).
4. The MLE stores the outcome and associated report in MLE database (KIB).

This is for the learning phase. For decision making, routing engine requests for SRLG identification from MLE with input as a single recently observed LS update. MLE runs the decision making algorithm and returns report to RIB.

6.4 Use case b3: Profile-based accountability

The functionality to implement the profiling strategies as defined for this use case assumes the following overall procedure within the ECODE component framework. The model consists in deriving during the learning phase the characteristic volume of congestion created per subscriber so as to derive congestion profiles. Conformance verification of this profile is then performed by comparing the deviation of the measured volume of congestion compared to the acceptable volume of congestion as determined (learned) for the subscriber profile to which the corresponding user belongs. The ratio between the observed (measured) and the classification value with respect to the congestion volume is a (per-user) accountability metric.

A continuous process ensures that number of packets marked as congested can be recorded per (source,destination). This assumes that the "forwarding plane" is able to detect congestion occurrences (on output I/Os) and mark packets accordingly (ECN bits, for instance).

1. Packets marked as "congested" are instantaneously accounted in number of bytes.
2. Selected records are forwarded to the Machine Learning Engine (MLE). These records are structured as a set of timestamped (IP Source Address, [IP Destination Address]) pairs, where the IP destination address is optional, together with a set of attributes, Attributes, including the "number of bytes of congestion" generated by IP source address(es) identifying a specific subscriber. The

input data to the Machine Learning Engine (MLE) is thus structured as the set $\{timestamp, (IP\ SA, [IP\ DA]), Attributes\}$.

3. The MLE builds up its learning model using this data:

- Records are first processed by the translator (format translation) and then semantically processed by the representation module before being processed as observations by the MLE processor. Typically, a set of records have been registered for a given "source IP address".
- At the *profile learning* stage the processor learns a classification model (whose classes define congestion profiles) based on the measured congestion volume per subscriber, identified by (a set of) IP source addresses. At the *execution* stage, the classification model is used to determine how subscriber profiles/users account for the congestion they created, i.e., how they deviate from the acceptable congestion level (associated to their profile) over time. This allows to pin down specific subscribers without having to maintain all "congestion states" and how they account for the explicit (or implicit) congestion feedback they receive from the network.

Note that various actions could be subsequently executed from the decision taken following the above identification. One can distinguish between *local* and *remote* actions/executions:

- Local actions depend mainly on the local forwarding plane capabilities. Examples includes: a node implementing *DiffServ* may be re-parametrized including the token bucket rate *r* and the burst size *b* of meters (that measure the traffic temporal properties, e.g., rate, for conformance against pre-defined traffic profile), the drop precedence bit marker thresholds, etc. but also enforce different conditioning actions that may be applied to in-profile packets and out-of-profile packets. Nodes implementing an Active Queue Management (AQM) scheme could also be re-parametrized, e.g., in case of Random Early Detection (RED), the average queue length and the min/max queue length threshold, and in case of Weighted RED (WRED), the respective weights in addition to the queue lengths parameters. Note that self-tunable/self-adaptable AQM (+ ECN) has already been subject to in-depth investigation. As such the proposed technique provides for retro-action detection/identification-analsys-decision-execution loop based on machine learning and can simply re-use one of these techniques.
- Remote actions (triggered locally) can take various forms. They can involve notification/tracing to the source of traffic (traceback) and/or trigger of the providers' accounting system.

Chapter 7

Conclusion

In this deliverable we have described the research work achieved so far in task 3.2 that is dedicated to the experimentation on the technical objective 2 (TO2) addressing machine learning techniques for path availability estimation (chapters 2 and 3), for improving network recoverability and resilience (chapter 4), and profile-based accountability (chapter 5).

In each chapter, the problem addressed by the use case is first formalized. Then the relevant machine learning (ML) techniques are briefly presented and used to provide appropriate solutions. The proposed ML-based algorithms are evaluated by simulations. The simulated codes used for this purpose constitute first high-level prototypes.

Chapter 6 explains how each use case fits in the general ECODE architecture presented in deliverable D2.1.

The main contributions can be summarized as follows:

- Measuring a path performance (as shown in chapter 2) according to one or several metrics, such as delay or bandwidth, is becoming more and more popular for applications. However, constantly probing the network is not suitable. To make measurements more scalable, the notion of clustering has emerged. We demonstrate in [2] that clustering can limit the measurement overhead in such a context without loosing too much accuracy. The paper shows that measurement reduction can be observed when vantage points collaborate and use clustering to estimate path performance. The paper also shows how effective is the overhead reduction and what is the impact in term of measurement accuracy. In addition, in [1], we demonstrate how to reduce the path performance metric measurements by applying machine learning techniques. We express the problem as a time series regression problem and propose several adaptative models for predicting those metrics. Based on a large dataset collected through the PlanetLab testbed, we evaluate our models and demonstrate their efficiency.

- When network nodes run an ICS (as shown in chapter 3, the measured distances (typically delays) between some of pairs of nodes are embedded into a metric

space (or coordinate system). When the coordinates of the nodes are known, the prediction of the distances (delays) between two nodes is a straightforward application of a distance function where no explicit communication between them is required. This significantly reduces the overhead of active probing and largely improves the efficiency of the network. We have first used Machine Learning techniques to improve the accuracy of an ICS [3, 4, 5]. We derive automatically a criterion that can be used by nodes to better select their neighbours in the ICS and thereby reduce the impact of Triangular Inequality Violations (TIVs), which are detrimental to an ICS. The knowledge of estimated delays between nodes can also be useful to select better paths for real-time applications. We have proposed some methods that rely on the nodes running an ICS to detect useful routing shortcuts in networks [6].

- The IP router RIB/FIB update process was formalized (as shown in chapter 4) to enable optimizing it in terms of packet loss upon failures (chapter 4). Traffic-informed router update models were evaluated using strategies with either fixed or optimized variable sizes for the update-distribution batches. The resulting models were implemented in a simulation environment and were quantitatively characterized. Depending on the context, we showed that the formulated strategies can result into a decrease of packet loss of 10 to 80 percent using small router process quantums [7]. Because a traffic-informed router update models can only be effective if the traffic statistics that are being used are accurate, the work is being extended such as to predict the short-term trend of aggregated network traffic flows. Currently, experiments are being carried out using Self-Organizing Maps (SOMs), Feed-Forward Neural Networks (FFNN) and Support Vector Regression (SVR) techniques.

- IP routers exchange link state advertisements (LSAs) to know about network failures and to initiate recalculation of routing paths in case of network failures (see chapter 4). Path computation and routing table updates takes time and induce packet loss in the network. On the other hand, by design, IP networks have Shared Risk Link Groups (SRLGs) that might give several failure indications. Current OSPF protocol cannot identify SRLGs and separately responds to each failure notification, which leads to higher packet losses. Within the ECODE project framework, a process is being developed to identify the SRLGs from the time sequences of LSAs that arrive in the process of failures. Here a router identifies an SRLG locally and reduces protection switching time by simultaneously updating the forwarding table for all the links that fail under the same SRLG. A simple Bayesian network based approach to model the SRLG identification have been developed. The Bayesian network based model includes a state transition approach that can perform online learning and infers probabilistic determination procedure for SRLGs. Further the study of the covariance among the LSA inter arrival times are being considered to enhance the probability based state space Bayesian network model and to include temporal data to infer the state transition probabilities. A realistic simulation scenario using GTNetS is being carried out.

- The aim of profile-based accountability (chapter 5) is to 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. There are two major tasks expected from the machine 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. Each profile indicates specific network traffic behavior associated with the usage and demand on the network resources. 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. Two distinct machine learning techniques are considered. A clustering technique based on the Fuzzy K-Mean algorithm is used to detect and identify commonalities in users/subscribers *actions* (unsupervised learning). An Hidden Markov Model (HMM) is learned for the supervised classification of subscribers/users (supervised learning).

- As shown in chapter 6, the functionalities of all the use cases fit well into the current ECODE architecture, which confirms its suitability.

Our future work will be organized along two axes:

- For each use case, we will continue to investigate machine mearning techniques to further improve our results.
- More work is also required to produce prototype codes for all use cases and integrate them into the common ECODE architecture. When these real implementations will be up and running, a second stage of tests and validations will be carried out to check whether the results promised by simulations are confirmed.

Bibliography

[1] Narino Mendoza J.P., Donnet B., and Dupont P. A comparative study of path performance metrics predictors. In Workshop on Advances in Learning for Networking, in conjunction with ACM SIGMETRICS/Performance 2009, In Proc. of ACM SIGMETRICS/Performance 2009, Seattle, USA, 15 June 2009, 2009.

[2] Donnet B. Saucez D. and Bonaventure O. On the impact of clustering on measurement reduction. In Proc. of IFIP/TC6 Networking 2009, 12-14 May 2009, Aachen, Germany, LNCS 5550, Springer., 2009.

[3] Y. Liao, M. A. Kaafar, B. Gueye, F. Cantin, P. Geurts, and G. Leduc. Detecting triangle inequality violations in internet coordinate systems by supervised learning. In Proc. IFIP Networking Conference, LNCS 5550, pages 352–363, Aachen, Germany, May 2009.

[4] M. A. Kaafar, F. Cantin, B. Gueye, and G. Leduc. Detecting triangle inequality violations for internet coordinate systems. In Proc. International Workshop on the Network of the Future, Dresden, Germany, June 2009.

[5] Y. Liao and G. Leduc. Triangle inequality violation avoidance in internet coordinate systems. In Trilogy Future Internet Summer School, Louvain-la-Neuve, Belgium, August 2009. Poster.

[6] Cantin F., Gueye B., Kaafar M.A., and Leduc G. Overlay routing using coordinate systems. In Poster at ACM Co-Next 2008, Madrid, Spain, ACM Press., 2008.

[7] Wouter Tavernier, Dimitri Papadimitriou, Didier Colle, Mario Pickavet, and Piet Demeester. Optimizing the ip router update process with traffic-driven updates. In DRCN 2009, Washington D.C., 2009.

[8] V. Bui, W. Zhu, A. Pescape, and A. Botta. Long horizon end-to-end dealy forecasts: a multi-step-ahead hybrid approach. In Proc. IEEE Symposium on Computers and Communications (ISCC), July 2007.

[9] M. Yang, J. Ru, H. Chen, A. Bashi, and N. S. V. Rao. Predicting Internet end-to-end delay: a statistical study. Annual Review of Network Management and Security, 58, April 2006.

[10] M. Yang, J. Ru, X. R. Li, H. Chen, and A. Bashi. Predicting Internet end-to-end delay: A multiple-model approach. In Proc. IEEE INFOCOM, April 2006.

[11] G.E.P. Box and G. Jenkins. Time series analysis, forecasting and control. Holden-Day, Incorporated, 1990.

[12] S.S. Soliman and M.D. Srinath. Continuous and discrete signals and systems. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1998.

[13] Z. Ghahramani and G. E. Hinton. Parameter estimation for linear dynamical systems. CRG-TR 92-2, University of Toronto, Department of Computer Science, February 1996.

[14] R.E. Kalman. A new approach to linear filtering and prediction problems. Journal of Basic Engineering, 82(1):35–45, 1960.

[15] K.-R. Müller, A. J. Smola, R. Rötsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In Proc. 7th International Conference on Artificial Neural Networks (ICANN), October 1997.

[16] J.D. Hamilton. Time Series Analysis. Princeton University Press, 1994.

[17] D. Saucez, B. Donnet, L. Iannone, and O. Bonaventure. Interdomain traffic engineering in a locator/identifier separation context. In Proc. Internet Network Management Workshop (INM), October 2008.

[18] Y. Zhang and N. Duffield. On the constantcy of Internet path properties. In Proc. ACM Workshop on Internet Measurement (IMW), October 2001.

[19] L.J. Tashman. Out-of-sample tests of forecasting accuracy: an analysis and review. International Journal of Forecasting, 16(4):437–450, 2000.

[20] J.G. De Gooijer and R.J. Hyndman. 25 years of time series forecasting. International journal of forecasting, 22(3):443–473, 2006.

[21] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In Proc. ACM SIGCOMM, Portland, OR, USA, August 2004.

[22] G. Wang, B. Zhang, and T. S. E. Ng. Towards network triangle inequality violation aware distributed systems. In Proc. the ACM/IMC Conference, pages 175–188, San Diego, CA, USA, October 2007.

[23] B. Deng et X. Li X. Wang, Y. Chen. Nonlinear modeling of the internet delay structure. In Proc. ACM CoNEXT Student Workshop, Madrid, Spain, December 2008.

[24] John M. McQuillan, Ira Richer, and Eric C. Rosen. An overview of the new routing algorithm for the arpanet. SIGCOMM Comput. Commun. Rev., 25(1):54–60, 1995.

[25] Paolo Narváez, Kai-Yeung Siu, and Hong-Yi Tzeng. New dynamic algorithms for shortest path tree computation. IEEE/ACM Trans. Netw., 8(6):734–746, 2000.

[26] Pierre Francois, Clarence Filsfils, John Evans, and Olivier Bonaventure. Achieving sub-second igp convergence in large ip networks. ACM SIGCOMM Computer Communication Review, 35(3):33–44, July 2005.

[27] Daniel P. Bovet and Marco Cesati. Understanding the linux kernel. O'Reilly, o' edition, d 2003.

[28] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, February 1993.

[29] T. Kohonen. Self-organization and associative memory: 3rd edition. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[30] Corinna Cortes and Vladimir Vapnik. Support-vector networks. Machine Learning, 20(3):273–297, 1995.

[31] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. Support vector regression machines. In NIPS, pages 155–161, 1996.

[32] David Verstraeten, Benjamin Schrauwen, Michiel D'Haene, and Dirk Stroobandt. An experimental unification of reservoir computing methods. Neural Networks, 20(3):391–403, 4 2007.

[33] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. Traffic data repository at the wide project. In ATEC '00: Proceedings of the annual conference on USENIX Annual Technical Conference, pages 51–51, Berkeley, CA, USA, 2000. USENIX Association.

[34] X. Pan and G. Xiao. Heuristics for diverse routing in wavelength-routed networks with shared risk link groups. Photonic Network Communications, (11):29–38, 2006.

[35] Dahai Xu, Yizhi Xiong, Chunming Qiao, and Guangzhi Li. Trap avoidance and protection schemes in networks with shared risk link groups. Journal of Lightwave Technology, 21(11):2683.

[36] Lei Guo, Hongfang Yu, and Lemin Li. A new shared-path protection algorithm under shared risk link group constraints for survivable wdm mesh networks. Optics Communications, 256(21):285–295, 2005.

[37] Inference of shared risk link groups. `http://www.cse.wustl.edu/~jain/oif/ftp/oif2001.066.1.pdf`.

[38] B.Rajagopalan. Link Bundling in Optical Networks. Internet-Draft draft-rs-optical-bundling-01.txt, Internet Engineering Task Force, October 2000. Work in progress.

[39] Irad Ben-Gal. Bayesian networks. John Wiley and Sons.

[40] David Heckerman. Tutorial on Learning with Bayesian Networks. MIT Press, 1998.

[41] Briscoe B. Flow rate fairness: dismantling a religion. ACM SIGCOMM Computer Communication Review, vol. 37(no. 2), April 2007.

[42] Briscoe B. A fairer, faster internet protocol. IEEE Spectrum, 45(no. 12):42–47, December 2008.

[43] Jacquet A., Briscoe B., and Moncaster T. Freedom to use the internet resource pool. Proceedings of the 2008 Workshop on Re-Architecting the Internet (ReArch'08), November 2008.

[44] Bernaille L., Teixeira R., and Salamatian K. Early application identification. In Proceedings of ACM CoNEXT 2006, pages 1–12, Berkeley, CA, USA, 2006.

[45] Li W. and Moore A. A machine learning approach for efficient traffic classification. In In Proceedings of IEEE MASCOTS 2007, 2007.

[46] McGregor A., Hall M., Lorier P., and Brunskill J. Flow clustering using machine learning techniques. In Proceedings of PAM 2004, 2004.

[47] Rabiner Lawrence R. A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE, 1989.

[48] Wright C., Monrose F., and Masson G. Hmm profiles for network traffic classification. In Proceedings of the Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC), October 2004.