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

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D3.5

"Experimental evaluation of 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-D3.5** Title: **Experimental evaluation 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, Steven Latré, Yongjun Liao, Stijn Melis, Juan Narino, Dimitri Papadimitriou, Damien Saucez, Wim Van de Meerssche, Wouter Tavernier, Bruno Willemaers** Dissemination level: **Public (PU)** Date of preparation: **16th July 2010** **Version: 1.1**

D3.5 - 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.

In a previous deliverable D3.4, we described the mapping of these use cases onto the proposed ECODE architecture. In particular we have shown 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.

In this deliverable, we focus on the improved design of the learning algorithm, their assessments mostly obtained by simulations, and early implementations in the Xorp environment.

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

- We explain how IDIPS (our path ranking service) has been implemented within XORP, an extensible open source routing platform. Our implementation is based on three modules: Ranking, Prediction, and Measurement. The Ranking module is in charge of dispatching requests from Clients to other modules. The Measurement module is used to measure path performance metrics, while the Prediction module uses a Machine Learning technique (Normalized Least Squares) to predict path performance metrics, so that we reduce the amount of traffic injected in the network. In addition, we provide a first description on how we plan to evaluate IDIPS on the iLab platform.

- Internet Coordinate Systems (ICS) are promising techniques to predict unknown network distances (typically delays) from a limited number of measurements. Most ICS algorithms are based on metric space embedding and suffer from the inability to represent distance asymmetries and Triangle Inequality Violations (TIVs). To overcome these drawbacks, we formulate the problem of network distance prediction as a machine-learning problem, namely guessing the missing elements of a distance matrix, and solve it by matrix factorization. A distinct feature of our approach [1], called Decentralized Matrix Factorization (DMF), is that it is fully decentralized. The factorization of the incomplete distance matrix is collaboratively and iteratively done at all nodes with each node retrieving only a small number of distance measurements. There are no special nodes such as landmarks nor a central node where the distance measurements are collected and stored. We compare DMF with two popular ICS algorithms: Vivaldi and IDES. Experimental results show that DMF achieves competitive accuracy with the double advantage of having no landmarks and of being able to represent distance asymmetries and TIVs. The knowledge of estimated delays between nodes can also be useful to select better paths for real-time applications. We had proposed some methods that rely on the nodes running an ICS to detect routing shortcuts in networks. We have now evaluated more precisely the quality of the results provided by these methods. Finally we explain a first implementation of an ICS within XORP. We also analyze the memory and performance cost of our module, and we explain how to improve this module for better integration in the ECODE architecture.

- An accurate understanding or characterization of network traffic dynamics can improve network efficiency. Long-term traffic characterization enables network operators to dimension their networks accordingly; short-term network traffic trends enable dynamic rerouting techniques to efficiently use alternative paths in a network. We use state-of-the-art network traffic models to model network traffic in a very short timeframe (sub-second) as observed by an IP router during the process of updating routing entries after failure detection. We study the highly sensitive interaction of network traffic with the IP router update process in detail and present a mathematical model to charac-

terize this process. The goal of this model is to optimize the process of updating routing entries in an IP router with minimal packet loss. For this, two optimization heuristics are formulated and are evaluated together with state-of-the-art alternatives in a realistic simulation environment. The trade-off of several parameters in the model is characterized, and we show that a gain in performance (decrease in packet loss) can be achieved.

- In the OSPF data mining use case for shared risk link groups (SRLG) identification, we use machine learning technique at the routers to study the link state protocol (e.g., OSPF) data to predict the existence of SRLG in the network. In particular, we use the correlation between different link state updates (LSUs) issued by different network nodes (routers) upon failure. The concerned network router then runs a novel Bayesian network-based statistical learning process to construct a state space model for representing and learning about the possible existence of SRLGs. The decision of this online learning is transferred to the routing information base (RIB) so that it can accordingly modify the routing table for the entire SRLG upon failure detection of one of the candidate link of that particular SRLG and hence reduce the protection switching time.

- With respect to the profile based accountability use case, this document presents an analytical model of the problem. This model identifies the different functions that need to be implemented through machine learning solutions. Different algorithmic options for implementing these functions are described and the way they can be applied to the specific problem is detailed. Furthermore, we discuss the experimental set-up that has been built and that includes a traffic generator, specifically built in the context of the ECODE project, that allows generating traffic traces on the iLab.t Virtual Wall based on the behaviour of actual applications. The results of initial tests that apply the machine learning algorithms on the obtained traffic traces are presented.

List of Authors

UCL	Benoit Donnet, Juan Narino, Damien Saucez
ULg	François Cantin, Pierre Geurts, Guy Leduc, Yongjun Liao, Bruno Willemaers
IBBT	Didier Colle, Goutam Das, Steven Latré, Wim Van de Meerssche, Stijn Melis, Wouter Tavernier
ALB	Dimitri Papadimitriou

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.

This deliverable D3.5 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;
- Design, implement and test a new Internet Coordinate System, which is a distributed machine learning engine 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 (IBBT)

- 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 an implementation of the IDIPS architecture, in XORP, 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 designing, evaluating and implementing a new Internet Coordinate System as a novel distributed machine learning engine, with the goals of estimating delays accurately and finding low delay (shortcut) paths in a scalable manner.

The network recovery and resiliency use case (b2) is addressed in chapters 4 and 5. In chapter 4, 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. In chapter 5, routing update messages are used to identify Shared Risk Link Groups (SRLGs) and thereby 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 6 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, an analytical model of the problem is proposed, which identifies the different functions that need to be implemented through machine learning solutions. Different algorithmic options for implementing these functions are described, the experimental set-up on the iLab.t Virtual Wall is discussed, and results of initial tests are presented.

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) [2, Chapter 2] is a service aiming at ranking paths according to their performance. To be able to make path ranking, IDIPS needs, from one hand, to collect path performance information, store it, and, on the other hand, collect path ranking requests and process them with a ranking algorithm.

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.

In this chapter, we explain how IDIPS has been implemented within XORP [3], an extensible open source routing platform.

We first focus on the implementation (Sec. 2.1) and, next, explain our plans for evaluating our implementation under the iLab infrastructure (Sec. 2.2).

2.1 XORP Implementation

2.1.1 Overview

Fig. 2.1 gives a global overview of the three IDIPS modules and how they interact with each others. In our implementation, we consider three mod-

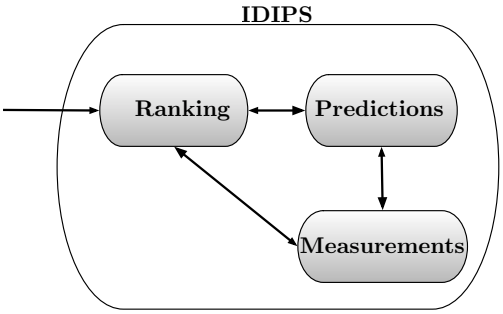


Figure 2.1: IDIPS global view

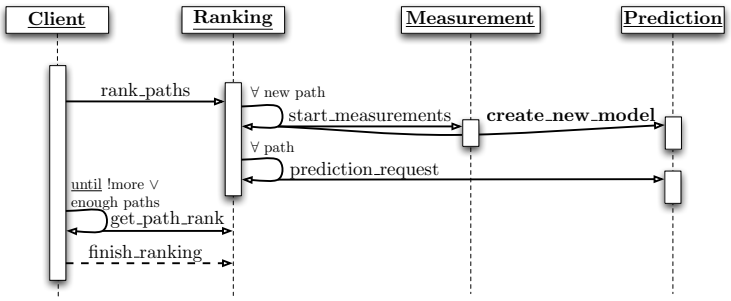


Figure 2.2: IDIPS flow diagram

ules:

- *Ranking* (Sec. 2.1.2). This module is in relationship with the *Client* (i.e., the "application" asking for a path ranking) and aims at providing the raking of the paths.
- *Prediction* (Sec. 2.1.4). This module is in charge of prediction path performance metrics (delay, bandwidth, ...) and aims at reducing the need of network measurements
- *Measurement* (Sec. 2.1.3). This module performs network measurements to evaluate path performance metrics.

We see that the Ranking module is in charge of dispatching request to the Prediction and Measurement modules.

2.1.2 Ranking

The Ranking module is in charge of receiving and processing the ranking requests received from the Clients. A request sent by a Client is always composed of three information:

- a list of sources.
- a list of destinations.
- a ranking criterion.

The sources and destinations correspond to the source and destination addresses the Client wants to compare in order to select the best pair. The Ranking module generates all possible paths from these addresses. A *path* is a (source, destination) pair, the source and destination belonging to the lists provided by the Client request. The criterion refers to the technique that must be used to rank the paths. A typical ranking technique is the delay, but one could imagine other techniques like monetary cost or any combination of metrics (see [4] for details).

Fig. 2.2 illustrates the lifetime of a ranking request. As illustrated, it works in three steps. First, the Client sends the request with the parameters discussed above. Next, the Ranking module computes the paths, their performance according to the criterion requested by the Client, and rank them. As a consequence, the Client can request for obtaining one-by-one the paths, the best being the first. It is worth to notice that the paths can be retrieved by the Client only after the Ranking module succeeded in determining the performance and the ranking itself.

The Ranking modules conserves a map of *ranking instances*. Each ranking instance is bounded to a request via a unique identifier (a *Transaction ID* or *TID*). A ranking instance consists of a list of paths and a ranking criterion. Once the performance of each path in a ranking instance is known, the ranking is computed and the instance is tagged as `ready`. When a ranking instance is ready, the Client can retrieve the rank of each path. If a client tries to get the path ranks before the instance is marked as `ready`, it generates an error.

Instead of providing directly the list of paths with their associated rank, the path ranking retrieval work on a per-path basis. It means that the Client can only ask one path at a time. Basically, the Client asks for the best remaining paths (i.e., the paths that it has not yet retrieved). The Ranking module then pops the best path still in the list and removes it from the list. The idea behind this technique is that Clients often only need the best path, not all of them. Once the Client has enough information, it might simply ask the Ranking module to terminate. This destroys the ranking instance.

The architecture we propose has the key advantage of being very simple to implement and lightweight on the Client side. However, it means that a ranking result cannot be used several times. If a Client needs to use it several times, it has to implement locally a cache for storing all the ranked paths. Remind that a ranking instances lives until it is explicitly destroyed by a Client or if all the paths have been returned to the Client.

Every ranking request is considered as being different but it is possible that a path appears in several requests. To avoid duplicating the path information, the path management inside IDIPS is decoupled from the ranking requests.

It is then possible to create a path independently of a ranking request. Creating a path consists first of abstracting it. A path is a collection of attributes (e.g., a source, a destination, ...) that is uniquely identified by an integer. Every time an operation has to be performed on a path, the action is started by providing the identifier, exclusively. If the action needs more information, it resolves the path from the identifier. When a path is created, a prediction model and a measurement instance are created. The prediction module is discussed later in Sec. 2.1.4 and the measurements are explained in Sec. 2.1.3. If a received ranking request involves a path that has not been created yet, it is created at the runtime and all the measurement and predictions models are created. We do not recommend to write clients that use path not created before.

2.1.3 Measurement

The Measurement module aims at performing path performance measurements for paths. Interesting metrics are delay, jitter, throughput, ...

For each metric, the Measurement module maintains a map associating a path identifier with the metric value of the last measurement. Periodically, a method traverses the map and determines which path has to be measured again. The path is then measured. To measure the path, its identifier has first to be resolved into a (source, destination) pair. Therefore, before doing a measurement, the Measurement module queries the Ranking module (which stores all the paths and their identifier) to obtain the pair. A resolution cache could be implemented but we do not have observed the need of such a cache yet.

If a module needs to know the current value of a metric for a given path, it can send a `measurement_request` to the Measurement module. The Measurement module will give the last observation it has. It is worth to notice that sending a measurement request will not trigger a measurement, it is only a lookup in the measurement cache that maintains the last observation for each path.

Measurements are performed for a given path until an explicit `stop_measurement` request is received for this path.

2.1.4 Prediction

In order to predict any of the path performance metrics, it is necessary to have a model per (source, destination) pair. This could lead to scalability issues, since some time series prediction methods have high computational and storage requirements. Most of today time series prediction methods, like ARMA, Kalman Filtering, Support Vector Regression, require of two things:

- A large amount of data samples to learn from. This is usually what is known in machine learning terms as the *train set*. This would have to be obtained live, once the router is running.
- Estimation of the parameters of the models, which is known as *training*. This can be computationally expensive, depending on the size of the training set. This is further compounded by the fact that the training must be run for each model to be predicted, which can take a considerable time if the system considers a large amount of models.
- Some of the models do not handle well changes in the statistical properties of the data (nonstationarity). As a result, changes in the signal impact negatively the prediction.

The alternative selected for our XORP implementation is to do prediction via adaptive filtering, particularly, *Normalized Least Mean Squares* (NLMS) adaptive filtering. Adaptive filtering presents some key advantages in terms of scalability compared to other methods:

- No large amount of data samples is required. An adaptive filter adjusts its parameters as data becomes available. Although certain amount of data samples is required for convergence, it is much less than for other models.
- The memory requirements are very low. It is enough to keep track of some coefficients and some constants.
- The training phase is not necessary. The filter finds its parameters on the fly, as data arrives.
- Adaptive filters change their parameters automatically to follow changes in the statistical properties of the data.
- Forecasting is very fast. Only one dot product is required.

The drawback of adaptive filters is that accuracy is lower when compared to other methods. However, preliminary experiments with actual data

show that the accuracy obtained with this method is still within acceptable bounds.

In the most simplest terms, in order to do prediction an NLMS filter can be seen as an autoregressive system whose weights are adjusted dynamically in order to improve prediction. If w_i are the weights, and x_i are the previous samples of the signal we are trying to predict, the prediction, defined as $\hat{y}_t$, is found as:

$$\hat{y}_t = \sum_{i=0}^{N} w_i \cdot x_i \tag{2.1}$$

The error of the prediction $e_t = \hat{y}_t - y_t$ is used for updating the weights in a way that the predictions adapt in an optimal way, in a least squares sense (LMS). In order to avoid numerical instability, the weights w_i are normalized by the energy of the signal. The weights update equation is:

$$w_{t+1} = w_t + \left(\frac{\beta \cdot x_t}{\left(\epsilon + \sqrt{|x_t|^2}\right)} * e_t \right) \tag{2.2}$$

where each term is explained below:

- w_t and w_{t+1} are the weights at time step t and $t+1$.
- β: This is the constant that indicates the rate of adjustment of the weights, also known as the *learning rate*.
- ϵ: This is a small constant term that is added just in the case that via numerical instabilities, the denominator becomes too small, so a small adjustment is provided such that the weights do not grow too fast.
- $sqrt|x_t|^2$: This term is added to normalize the weights, in a way that big changes on the input signal scale will not imply huge changes on the weights, thus bringing undesired instability.

In this way, by changing the weights in an optimal way that minimizes the least square error of the prediction with respect to the actual value is minimized, the filter adapts to changes in the input signal. This represents an advantage since some of the data present abrupt jumps due to changes in the routes, as a result, adaptivity is a desirable feature. Fig. 2.3 illustrates how NLMS filtering works.

The architecture of the Prediction module is shown on Fig. 2.4.

In order to keep the system scalable, a map was chosen as the main data structure. This allows one for fast data lookups and for maintaining the

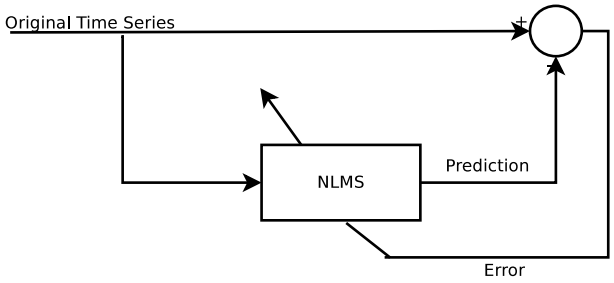


Figure 2.3: Schematic diagram of NLMS filtering

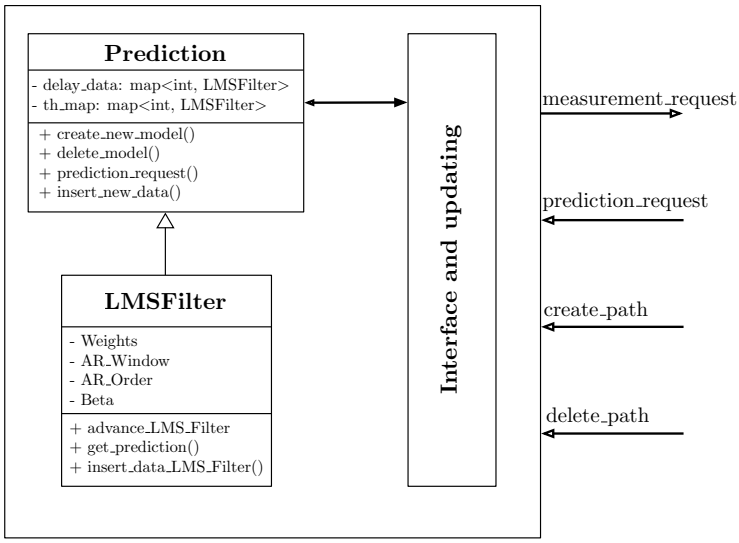


Figure 2.4: Prediction module architecture

relevant models. There are two maps, each one being dedicated for LMS instances for a path performance metric (i.e., delay or throughput) for (source, destination) pairs.

The "Prediction module" is in charge of managing maps and routing correctly requests to the LMS filter. The "Interface and updating" is in charge of handling communication with other modules via XRL (see [5] for details).

2.2 iLab Planned Experimentations

IDIPS will be evaluated on the *iLab.t* testbed. The evaluation will involve two IDIPS servers and 50 Clients, as illustrated on Fig. 2.5. Path performance will be both synthetic and real. The considered path ranking criterion will be the delay.

We plan our experiment being on four phases:

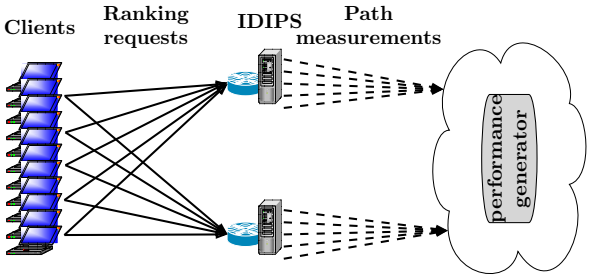


Figure 2.5: IDIPS testbed on the iLab.t platform

1. Phase 1: This first phase is a proof of concept to see if the implementation works as expected on iLab.t. It is based on a single Client, a single IDIPS server and four paths. The Client sends one request to the IDIPS server, asking the ranking of four paths. The *path performance generator* (dummynet or any equivalent tool provided by the iLab.t infrastructure) is in charge of providing various path dynamics. During this first phase, the paths are purely emulated, so that we are sure that all parameters are under control.

2. Phase 2: This second phase aims at determining the scalability of the system. It involves up to 50 Clients, a single IDIPS server and up to 1,000 paths. The Clients send their ranking paths requests in parallel. It is worth to notice that the paths to rank might be shared between Clients but might also be specific to a given Client. The idea is to consider several runs with various number of Clients and paths to rank in order to see if the system scales. In particular, we are interested in studying the evolution of processing time and memory usage.

3. Phase 3: In this third phase, our aim is to validate the consistency of the IDIPS ranking. It involves a single Client, two IDIPS servers, and up to 100 paths. The Client sends simultaneously the same requests to both IDIPS server. The ranking replied by both servers should be identical or, at least, the same according to the selected criterion.

4. Phase 4: The last phase is identical to the previous but “in the wild”, i.e., we will repeat the first three phases with measurements to real paths through the Internet instead of of artificially generating path performance metric.

Chapter 3

Delay estimation and delay-based path selection and routing

Internet Coordinate Systems (ICS) are promising techniques to predict unknown network distances from a limited number of measurements. Most ICS algorithms are based on metric space embedding and suffer from the inability to represent distance asymmetries and Triangle Inequality Violations (TIVs). To overcome these drawbacks, we formulate the problem of network distance prediction as guessing the missing elements of a distance matrix and solve it by matrix factorization. A distinct feature of our approach, called Decentralized Matrix Factorization (DMF), is that it is fully decentralized. The factorization of the incomplete distance matrix is collaboratively and iteratively done at all nodes with each node retrieving only a small number of distance measurements. There are no special nodes such as landmarks nor a central node where the distance measurements are collected and stored. We compare DMF with two popular ICS algorithms: Vivaldi and IDES. The former is based on metric space embedding, while the latter is also based on matrix factorization but uses landmarks. Experimental results show that DMF achieves competitive accuracy with the double advantage of having no landmarks and of being able to represent distance asymmetries and 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. Therefore, in this chapter we will also propose some methods that rely on the nodes running an ICS to detect useful routing shortcuts in networks.

Finally we will explain a first implementation of an ICS in the XORP environment.

3.1 Network Distance Prediction Based on Decentralized Matrix Factorization

3.1.1 Introduction

Predicting network distances (e.g. delay) between Internet nodes is beneficial to many Internet applications, such as overlay routing [6], peer-to-peer file sharing [7], etc. One promising approach is Network Coordinate Systems (NCS), which construct models to predict the unmeasured network distances from a limited number of observed measurements [8].

Most NCS algorithms embed network nodes into a metric space such as Euclidean coordinate systems in which distances between nodes can be directly computed from their coordinates. For example, GNP [9] is the first system that models the Internet as a geometric space. It first embeds a number of landmarks into the space and a non-landmark host determines its coordinates with respect to the landmarks. Vivaldi [10] is a decentralized NCS system that extends GNP by eliminating the landmarks. It simulates a system of springs where each edge is modeled by a spring and the force of the spring reflects the approximation error.

However, network distances do not derive from the measurements of a metric space. For example, Triangle Inequality Violations (TIVs) have been frequently observed and the distances between two nodes are not necessarily symmetric due to the network structure and routing policy [11, 12, 13, 10, 14]. No algorithm based on metric space embedding can model such distance space. IDES [15] is one of the few algorithms using non-metric space embedding techniques. It is based on matrix factorization which approximates a large matrix by the product of two small matrices. A drawback of IDES is that, similar to GNP, IDES also relies on landmarks. It factorizes a small

[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.

distance matrix built from the landkmarks at a so-called information server and other non-landmark nodes compute their coordinates with respect to the landmarks. Phoenix extends IDES by adopting a weight model and a non-negativity constraint in the factorization to enforce the predicted distances to be positive [16]. In practice, NCS systems with landmarks are less appealing. They suffer from landmark failures and overloading. Furthermore, the number of landmarks and their placement affect the performance of NCS.

In this paper we propose a novel approach, called Decentralized Matrix Factorization (DMF), to predicting network distance. Unlike IDES, we seek to factorize a distance matrix built from all nodes in a fully decentralized manner. Each node retrieves distance measurements from and to a small number of randomly selected nodes[1] and updates its coordinates simultaneously and iteratively. There are no special nodes such as landmarks or a central node where distance measurements are collected and stored. In doing so, our DMF algorithm overcomes the drawbacks of metric space embedding and is able to represent asymmetric distances and TIVs, while it is fully decentralized and requires no landmarks. Experimental results show that DMF is stable and achieves competitive performance compared to IDES and metric space embedding based algorithms such as Vivaldi.

The rest of the paper is structured as follows. Section 2 formulates the problem of network distance prediction by matrix factorization. Section 3 describes the DMF algorithm. Section 4 presents the evaluation and the comparison of DMF with other competing methods. Section 5 gives the conclusions and discusses future work.

3.1.2 Matrix Factorization for Network Distance Prediction

Matrix Factorization seeks an approximate factorization of a large matrix, i.e.,

$$D \approx \hat{D} = XY^T,$$

where the number of columns in X and Y is typically small and is called the dimension of the embedding space. Generally, the factorization is done by minimizing $||D - \hat{D}||^2$, which can be solved analytically by using singular value decomposition (SVD) [17]. In many cases, constraints can be imposed in the minimization. A popular and useful constraint is that elements of X and Y are non-negative. This so-called non-negative matrix factorization (NMF) can only be solved by iterative optimization methods such as gradient descent [18]. Note that matrix factorization has no unique solution as

$$D \approx \hat{D} = XY^T = XGG^{-1}Y^T,$$

[1] We will refer to the selected nodes as neighbors in the rest of the paper.

where G is any arbitrary non-singular matrix. Therefore, replacing X by XG and Y by $G^{-1}Y$ will not increase the approximation error.

In using matrix factorization for network distance prediction, assuming n nodes in the network, a $n \times n$ distance matrix D is constructed with some distances between nodes measured and the others unmeasured. To guess the missing elements in D, we factorize D into the form $D \approx XY^T$ by solving

$$\textbf{min } ||W. * (D - XY^T)||^2, \tag{3.1}$$

where $.*$ is element-wise product and W is the weight matrix with $w_{ij} = 1$ if d_{ij}, the distance from i to j, is measured and 0 otherwise. X and Y are of the same size $n \times l$ with $l \ll n$. l is referred to as the dimension of the embedding space and is a parameter of the factorization algorithm.

With missing elements, the minimization of eq. 3.1 can only be solved by iterative optimization methods. After the factorization, each node is then associated with two coordinates x_i and y_i, where x_i is the ith row of X, called *outgoing vector*, and y_i is the ith row of Y, called *incoming vector*. The estimated distance from i to j is

$$\hat{d}_{ij} = x_i \cdot y_j, \tag{3.2}$$

where $\cdot$ is the dot product. If done properly, the estimated distance, $\hat{d}_{ij}$, approximates the measured distance, d_{ij}, within a limited error range. Note that $\hat{d}_{ij}$ is not necessarily equal to $\hat{d}_{ji}$.

The above process is centralized and requires a large number of distance measurements to be collected and stored at a central node. To solve this problem, IDES [15] proposed to select a small number of landmarks and compute, at a so-called information server, the factorization (by using SVD or NMF) of a small distance matrix built only from measured distances between the landmarks. Once the landmark coordinates have been fixed, a non-landmark host can determine its coordinates by measuring its distance to and from each of the landmarks and finding coordinates that most closely match those measurements. As mentioned earlier, the use of landmarks is a weakness of IDES. In the next section, we will propose our approach based on a decentralized matrix factorization that eliminates the need for landmarks.

3.1.3 Decentralized Matrix Factorization for Network Distance Prediction

The problem is the same as in eq. 3.1, but we seek to solve it in a decentralized manner. Similar to IDES, each node records its outgoing vector x_i and incoming vector y_i and computes distances from and to other nodes by using eq. 3.2. The difference is that x_i and y_i are initialized randomly and updated continuously with respect to some randomly-selected neighbors.

In particular, to update x_i and y_i, node i randomly selects k neighbors, measures its distances from and to them, and retrieves the outgoing and incoming vectors. Denote $X_i = [x_{i_1}; \ldots; x_{i_k}]$ and $Y_i = [y_{i_1}; \ldots; y_{i_k}]$ the outgoing and incoming matrices built from the neighbors of i, i.e., x_{i_j} and y_{i_j} are the outgoing and incoming vectors of the jth neighbors of i. Let $d^i_{to} = [d_{i,i_1}, \ldots, d_{i,i_k}]$ and $d^i_{from} = [d_{i_1,i}, \ldots, d_{i_k,i}]$ the distance vectors to and from the neighbors of i. Then, x_i and y_i are updated by

$$x_i = \arg\min_x \quad ||xY_i^T - d^i_{to}||^2, \tag{3.3}$$

$$y_i = \arg\min_y \quad ||X_i y^T - {d^i_{from}}^T||^2. \tag{3.4}$$

Eqs. 3.3 and 3.4 are standard least square problems of the form $min||Ax-b||^2$, which has an analytic solution of the form:

$$x = (A^T A)^{-1} A^T b. \tag{3.5}$$

To increase the numerical stability of the solution, instead of solving eqs. 3.3 and 3.4 with eq 3.5, we penalize x_i and y_i and solve regularized least square problem of the form $min||Ax-b||^2+\lambda||x||^2$, which also has an analytic solution

$$x = (A^T A + \lambda I)^{-1} A^T b, \tag{3.6}$$

where λ is the coefficient of the regularization term. In the experimental section, we will show the influence of the regularization terms on the performance of DMF.

To summarize, the update equations of x_i and y_i are

$$x_i \quad = d^i_{to} Y_i (Y_i^T Y_i + \lambda I)^{-1} \tag{3.7}$$

$$y_i \quad = d^i_{from} X_i (X_i^T X_i + \lambda I)^{-1} \tag{3.8}$$

The DMF algorithm is given in Algorithm 1[2]. We initialize the coordinates with random numbers uniformly distributed between 0 and 1. Empirically, we found that DMF is insensitive to the random initialization of the coordinates.

3.1.4 Experiments and Evaluations

In this section, we evaluate DMF[3] and compare it with two popular NCS algorithms: Vivaldi and IDES. The former is based on metric space embed-

[2]Note that we can also adopt the weight model and the non-negativity constraint as in [16]. However, the constrained minimization in eqs 3.3 and 3.4 have no more closed form solutions and has to be solved by iterative optimization methods. As claimed in [15], which is confirmed by our experiments, the non-negativity constraint does not improve the accuracy a lot, but significantly increases the computing time.

[3]A matlab implementation of DMF used to generate the results in the paper can be downloaded from `http://www.run.montefiore.ulg.ac.be/~liao/DMF`.

Input: D, l, k, λ
D: distance matrix with missing elements
l: dimension of the embedding space
k: number of neighbors of each node
λ: regularization coefficient
Output: X,Y
foreach node i **do**
  Randomly select k neighbors from the network.
  Randomly initialize x_i and y_i.
  while forever **do**
    retrieve d^i_{to}, d^i_{from}, X_i, Y_i;
    update x_i by eq. 3.7
    update y_i by eq. 3.8
    sleep some time
  end
end

Algorithm 1: DMF: Decentralized Matrix Factorization with Regularization.

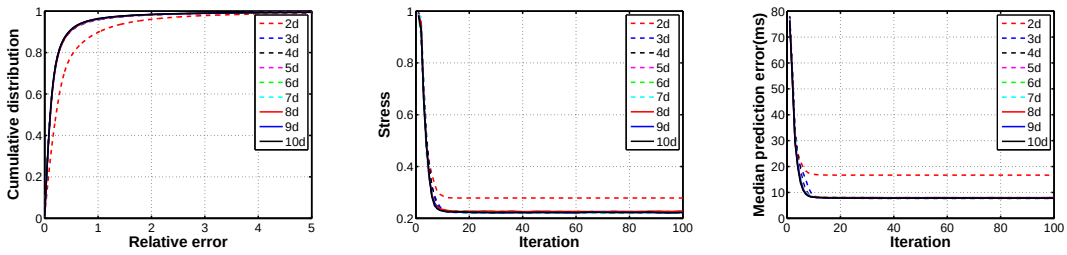
ding, while the latter is also based on matrix factorization but uses landmarks. All the experiments are performed on two typical data sets collecting real Internet measurements: the P2psim [19] data set which contains the measured distances between 1740 Internet DNS servers, and the Meridian [20] data set which contains the measured distances between 2500 nodes. While DMF can in principle handle asymmetric distance matrices, in our experiment, we took $d_{i,j} = d_{j,i}$ and defined these distances as the half of the round-trip-time between nodes i and j. The same assumption is adopted in Vivaldi and has the advantage of greatly simplifying the implementation of the algorithm, as measuring one-way delay is difficult in practice.

In the simulations, we randomly selected a node and updated its coordinates at each step. An iteration of a simulation is defined by a fixed round of node updates. Since Vivaldi updates its coordinates with respect to only one neighbor in contrast to DMF that does it with respect to all neighbors, an iteration in Vivaldi is defined by $n \times k$ node updates whereas in DMF an iteration is n node updates, where n is the number of nodes and k is the number of neighbors. In doing so, we ensure that, on average, all nodes have a chance to update their coordinates with respect to all neighbors. Note that IDES is not an iterative method. The coordinates of the nodes are unchanged.

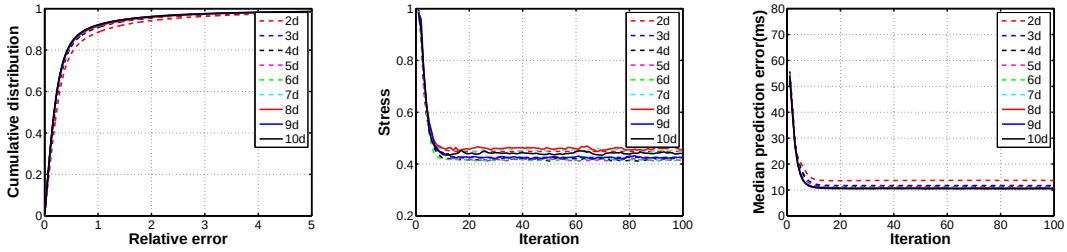
We examine the following classical evaluation criteria.

- *Cumulative Distribution of Relative Estimation Error* Relative Estimation Error (REE) is defined as

$$REE = \frac{|\hat{d}_{i,j} - d_{i,j}|}{d_{i,j}}.$$



(a) P2psim: REE (b) P2psim: Stress (c) P2psim: MAEE



(d) Meridian: REE (e) Meridian: Stress (f) Meridian: MAEE

Figure 3.1: The effect of the dimension (l) on the performance of DMF. ($k = 32$, $\lambda = 50$)

- *Stress* measuring the overall fitness of the embedding is defined as

$$stress = \sqrt{\frac{\sum_{i,j} (d_{i,j} - \hat{d}_{i,j})^2}{\sum_{i,j} {d_{i,j}}^2}}.$$

- *Median Absolute Estimation Error* (MAEE) is defined as

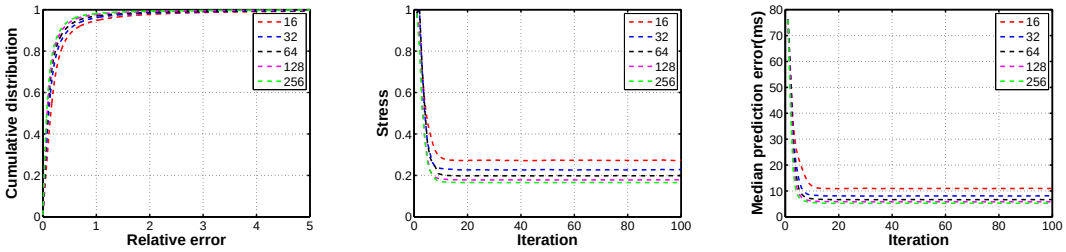
$$MAEE = median_{i,j}(|d_{i,j} - \hat{d}_{i,j}|).$$

Note that our DMF algorithm utilizes only a small percentage of the distance measurements in the datasets to estimate the coordinates of the nodes, but the evaluation of the above criteria is done using all distance measurements.

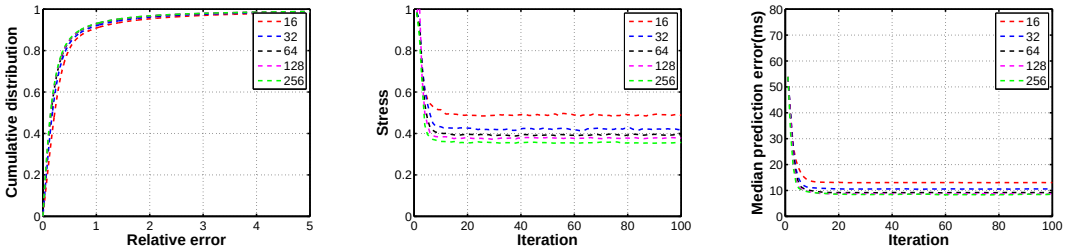
3.1.4.1 Parameter Tuning

DMF has three parameters to be defined: l, the dimension of the embedding space, k, the number of neighbors of each node, and λ, the regularization coefficient. We study the influence of these parameters on the performance of DMF. To this end, we tune one parameter at a time while fixing the other two. Results are shown in Figures 3.1, 3.2 and 3.3.

It can be seen that l does not seem to affect the performance of DMF as long as $l \geq 3$ which coincides with the conclusion drawn in [10] about Vivaldi. We nevertheless recommend $l = 10$ as it does not pose any problem

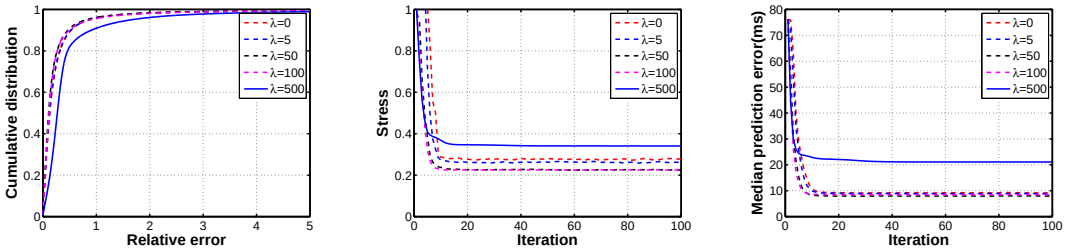


(a) P2psim: REE (b) P2psim: Stress (c) P2psim: MAEE

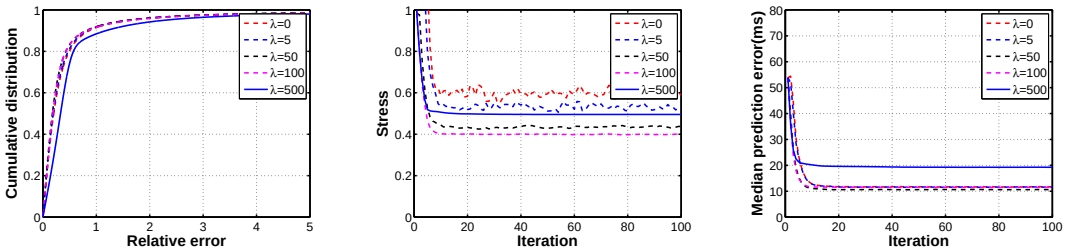


(d) Meridian: REE (e) Meridian: Stress (f) Meridian: MAEE

Figure 3.2: The effect of the number of neighbors (k) on the performance of DMF. ($l = 10$, $\lambda = 50$)

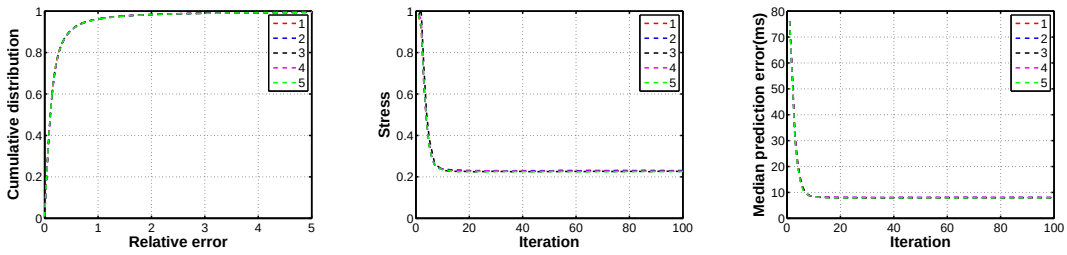


(a) P2psim: REE (b) P2psim: Stress (c) P2psim: MAEE

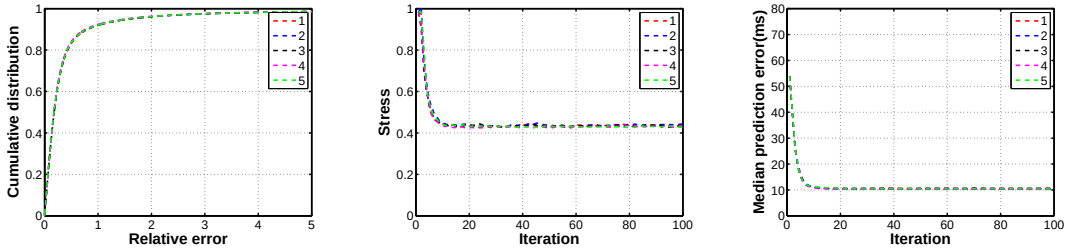


(d) Meridian: REE (e) Meridian: Stress (f) Meridian: MAEE

Figure 3.3: The effect of regularization coefficient (λ) on the performance of DMF. ($l = 10$, $k = 32$)



(a) P2psim: REE (b) P2psim: Stress (c) P2psim: MAEE



(d) Meridian: REE (e) Meridian: Stress (f) Meridian: MAEE

Figure 3.4: Results of different simulations. The simulations differ in the initializations of the coordinates, in the selections of the neighbors by each node and in the orders in which the nodes are updated. It can be seen that the results are insensitive to these differences.

and as the same number is used in IDES. On the other hand, k has a clear impact, as a larger k gives better accuracy, which is obvious because a larger k means fewer missing elements thus better estimation of the coordinates. However, a larger k also means more probe traffic and a higher overhead. Following Vivaldi, we suggest $k = 32$ as a good tradeoff between accuracy and measurement overhead. For λ, too little or too much regularization only decreases the accuracy of DMF, and 50 seems to be a good choice for both P2psim and Meridian datasets.

Note that the results are very stable from one simulation to another, as highlighted in Figure 3.4. The algorithm does not seem very sensitive to the random initialization of the coordinates and to the particular selection of neighbours. In the following, unless otherwise stated, $l = 10$, $k = 32$ and $\lambda = 50$ are used by default and the results of all the experiments are derived from one simulation.

3.1.4.2 Analysis of Convergence and Stability

We further evaluate the convergence and the stability of DMF. From Figures 3.1, 3.2, 3.3 and 3.4, it is clear that DMF converges fast, empirically in less than 20 iterations for both P2psim and Meridian datasets.

To further verify the stability of DMF, we performed a 2D factorization

($l = 2$) and plotted the X and Y coordinates at different times of the simulation, shown in Figure 3.5. It can be seen that the coordinates are very stable with little drift after the embedding errors become stable. Figure 3.6 shows the histogram of the differences between the predicted distance matrix at the 20th and the 100th iterations.

3.1.4.3 Comparisons with Vivaldi and IDES

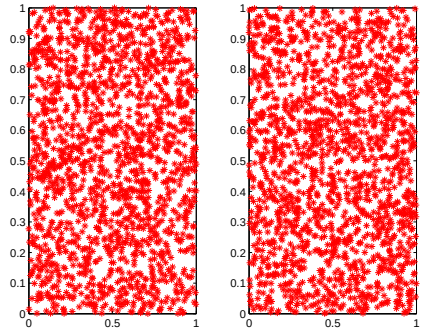
Lastly, we compare DMF with Vivaldi and IDES, as shown in Figure 3.7. Vivaldi is a decentralized NCS algorithm based on Euclidian embedding. Similar to DMF, each node updates its coordinates with respect to k randomly selected neighbors. Here, we took $k = 32$ following the recommendation in Vivaldi. For IDES, a number of landmarks are needed. Although [21, 15] claimed that 20 randomly selected landmarks are sufficient to achieve desirable accuracy, we nevertheless deployed 32 landmarks in our experiments for the purpose of comparison. The dimensions of the embedding space are 10 for all algorithms.

From Figure 3.7, it can be seen that both DMF and Vivaldi achieve similar accuracy and slightly outperform IDES. The worse performance by IDES is likely due to the use of the landmarks. Since in IDES, a non-landmark node only communicates with landmarks, no links between non-landmark nodes are used by the NCS. In contrast, DMF and Vivaldi are completely decentralized with links between nodes randomly selected and evenly distributed in the whole network.

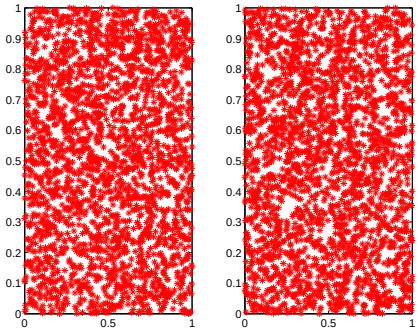
3.1.5 Conclusions and Future Works

In this paper, we proposed a novel approach, called DMF, to predicting unknown network distances. Essentially, we consider it as a learning problem where coordinates of network nodes are learned from partially observed measurements and the unknown distances are approximated from the learned coordinates. Different from all previous works, the learning of the coordinates is done by DMF which requires no landmarks. Since DMF is not based on metric space embedding, it has the potential to overcome common limitations such as the inability to represent TIVs and asymmetric distances. Experimental results show that the performance of our approach is comparable with two popular NCS algorithms: Vivaldi and IDES.

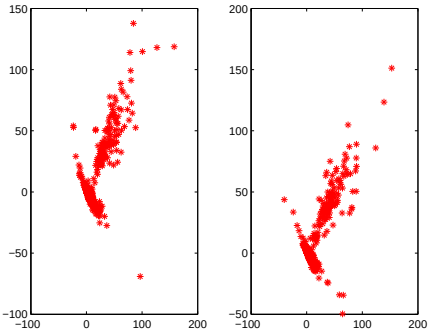
Finally it has to be noted that both P2psim and Meridian datasets are symmetric with $d_{ij} = d_{ji}$. We are currently testing DMF on more datasets, especially those with heavily asymmetric distances.



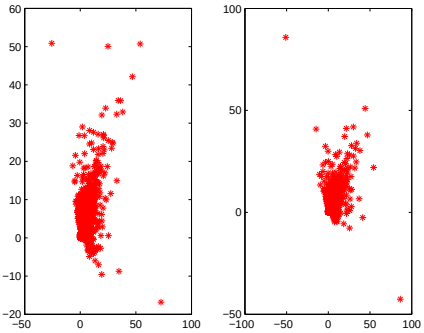
(a) P2psim: Initialization



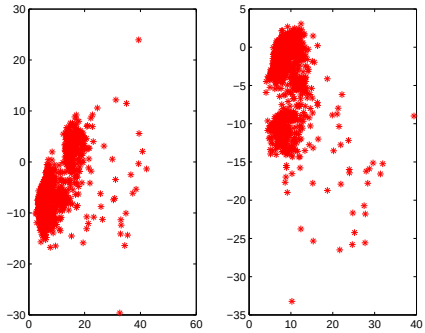
(b) Meridian: Initialization



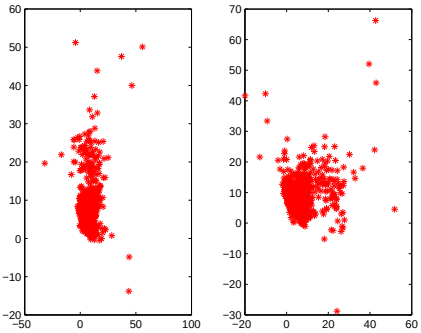
(c) P2psim: 1st iteration



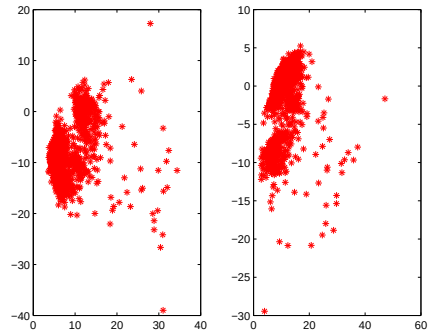
(d) Meridian: 1st iteration



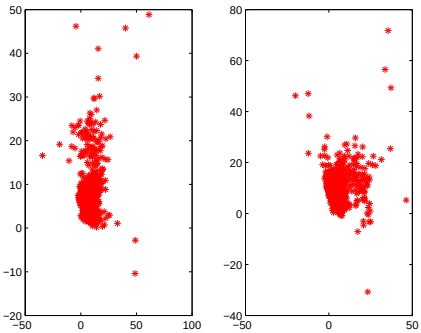
(e) P2psim: 20th iteration



(f) Meridian: 20th iteration

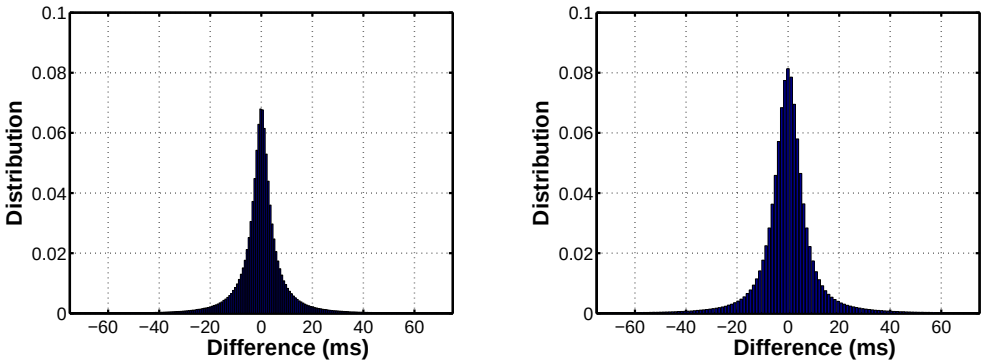


(g) P2psim: 100th iteration



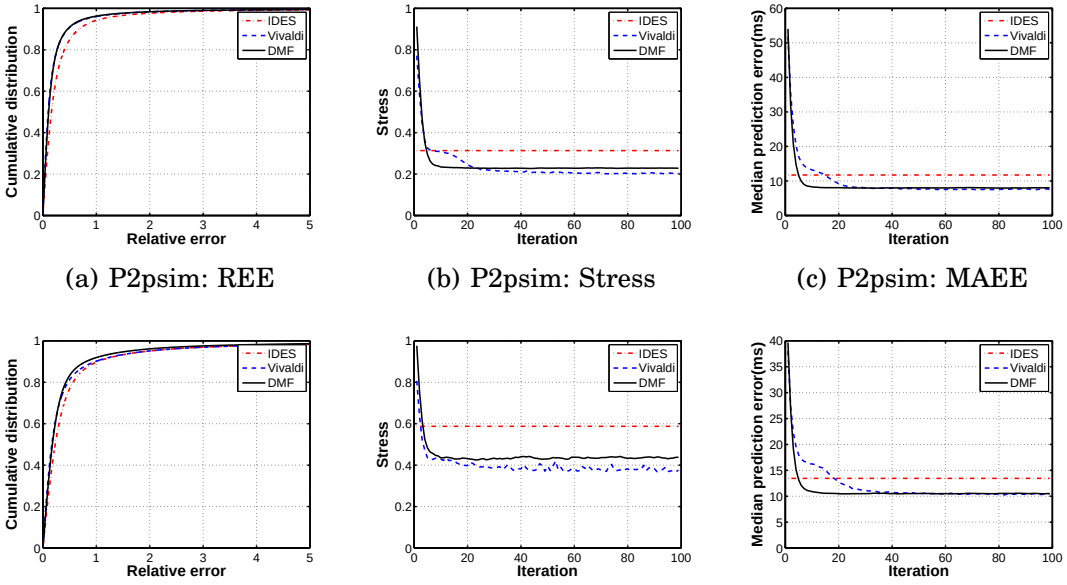
(h) Meridian: 100th iteration

Figure 3.5: The evolution of the coordinates, X(left subplot) and Y(right subplot).



(a) P2psim: difference (b) Meridian: difference

Figure 3.6: The differences between the predicted distance matrix at the 20th and the 100th iterations.



(a) P2psim: REE (b) P2psim: Stress (c) P2psim: MAEE

(d) Meridian: REE (e) Meridian: Stress (f) Meridian: MAEE

Figure 3.7: Comparison with IDES and Vivaldi. ($l = 10$, $k = 32$, $\lambda = 50$ for DMF)

3.2 Finding routing shortcuts

The knowledge of estimated delays between nodes can also be useful to select better paths for real-time applications. In the previous deliverable, we have proposed some methods that rely on the nodes running an ICS to detect routing shortcuts in networks. In this section we evaluate more precisely the quality of the results provided by these methods.

3.2.1 Problem Formalization

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 to 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. As such criteria can provide a large set of nodes, we need also a way to rank the C nodes in order to find the best shortcuts as fast as possible.

3.2.2 Implementation

Without loss of generality we consider a classical ICS algorithm, Vivaldi [10], and 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 compares the

mesured RTT of the direct path between A and B and the estimated RTT of the alternative path using C as relay. Formaly, a node C is considered as a shortcut for the path AB if

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

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: a node C is considered as a shortcut for the path AB if,

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

The problem with such criteria is that they do not provide a set of nodes containing only the best shortcuts: they provide a possibly large set of nodes containing nodes that are important shortcuts, node that are less important shortcuts and even nodes that are not shortcuts (detection errors). So, we need a way to rank the C nodes of a set in order to find quickly and easily the best shortcuts in that set. Since we want to find the node C providing the smallest RTT for a path between A and B, we will rank the C nodes by order of provided gain. For a path AB, the *absolute gain* (G_a) and the *relative gain* (G_r) provided by a node C are

$$G_a = RTT(A,B) - (RTT(A,C) + RTT(C,B)) \qquad G_r = \frac{G_a}{RTT(A,B)}$$

If C is a shortcut for the path AB, then G_a and G_r will have positive values and the most interesting shortcut is the one that provides the biggest value for these parameters. However, we cannot compute the values G_a and G_r for any node C in a set provided by one of our criteria. Indeed, generally, we do not know the real RTT of the alternative path that uses node C : we only have Vivaldi's estimations for that path. As we have used an estimation / approximation for the RTT of the alternative path in the shortcut detection criteria, we will also use that estimation / approximation in the ranking criteria. The values used to rank the C nodes of a set will be denoted *estimated absolute gain* (EG_a) and *estimated relative gain* (EG_r). The definitions of these values depend of the shortcuts detection criterion used to obtain the set of C nodes. For EDC, the definitions are

$$EG_a = RTT(A,B) - (EST(A,C) + EST(C,B)) \qquad EG_r = \frac{EG_a}{RTT(A,B)}$$

and, for ADC, the definitions are

$$EG_a = RTT(A,B) - (RTT(A,C_A) + RTT(C_B,B)) \qquad EG_r = \frac{EG_a}{RTT(A,B)}$$

For a path AB, we will rank the nodes C of the set provided by a shortcuts detection criterion by decreasing order of estimated gain. If the nodes for which the estimated gains are the biggest are the ones for which the (real) gains are the biggest then we will find the nodes providing the most interesting shortcuts in the top of the ranking.

3.2.3 Experimentation and evaluation

3.2.3.1 Performance of our routing shortcut detection criteria

In the previous deliverable, to evaluate the performance of our routing shortcut criteria, we used 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* (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. Since a shortcut is not necessary useful[2], we defined an *interesting shortcut* as a shortcut that provides at least an absolute gain of $10\,ms$ and a relative gain of 10%. We also defined 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 10 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.

[2]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.

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

	P2PSim		Meridian		Planetlab	
	All	Int.	All	Int.	All	Int.
EDC	21%	36%	35%	41%	19%	49%
ADC	54%	68%	78%	80%	52%	70%

Table 3.1: EDC and ADC best shortcuts detection results

By analysing the obtained results we saw that the percentage of interesting shortcuts detected as shortcuts (ITPR) was good in most of the cases for both criteria. Furthermore, the percentage of non-shortcuts detected as shortcuts was generally quite low. We concluded that the EDC criterion was better than ADC : ADC was always able to detect a little bit more shortcuts than EDC but it also gave more false positives.

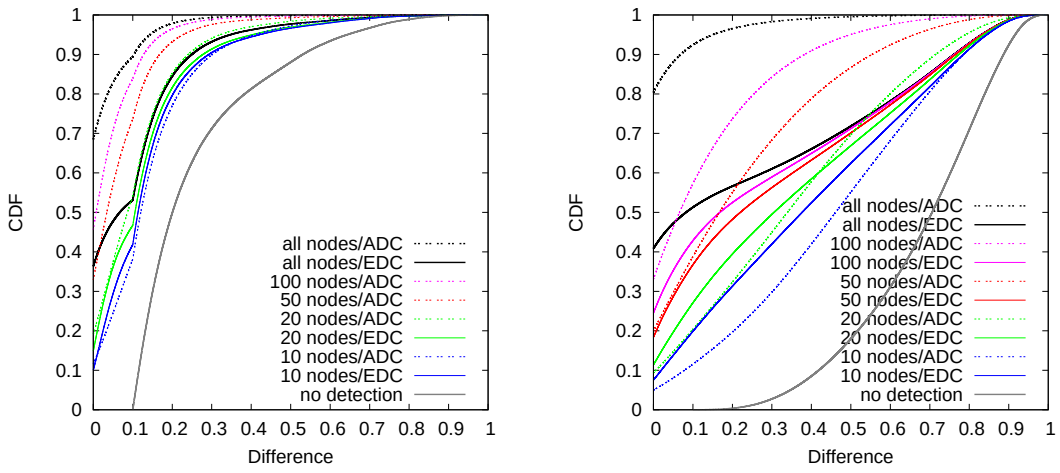
3.2.3.2 Detection of the best shortcuts

Being able to detect a large part of the shortcuts existing in a network is one thing, but what matters most is to detect the most interesting shortcuts (those that provide the most important gain). Considering only the paths for which there exists at least a shortcut (resp. an interesting shortcut), the percentages of paths for which the most interesting shortcut is detected by the criteria are given in table 3.1 in the columns named "All" (resp. in the columns named "Int.").

As regards the detection of the best shortcuts, we see in table 3.1 that the results obtained with the criterion ADC are better than those obtain with EDC. Considering only the paths for which there exists at least an interesting shortcut, the EDC criterion is able to find the best shortcut for 40% of the paths (on average) while the ADC criterion is able to find the best shortcut for more than 70% of the paths in each matrix. Regarding those results ADC seems to be a better criterion than EDC.

However, we must perhaps moderate our conclusion. Firstly because ADC returns large sets of C nodes compared to EDC (including a non-negligible number of false positives): at the limit, a criterion that detect as shortcut each C node will detect the best shortcut for each path but is completely useless. So, if we choose to use ADC[4], we absolutely need a criterion to rank the C nodes of a returned set in order to keep only a subset of the nodes. Moreover, EDC can give better results than we think. Indeed, even if a criterion cannot find the best shortcut for a path, it may be able to find another shortcut that provides almost the same gain. We will analyse that in the next section: for each path, we will compute the difference between the G_r provided by the best *existing* shortcut and the G_r provided by the best *detected*

[4]Such criterion can also be useful for EDC because, even if the sets of C nodes are generally smaller than those returned by ADC, they can contain tens or hundreds of nodes.



(a) P2PSim (b) Meridian

Figure 3.8: Difference of G_r between the best shortcut and the best shortcut detected

shortcut.

3.2.3.3 Ranking of the detected nodes

Considering the results of the previous section, ADC seems to be better than EDC. However, we have seen that ADC returns a set of C nodes containing a non negligible number of false positives and, more generally, a large number of nodes. Thus, having a criterion to rank the C nodes of a set is important.

For a given matrix and a given detection criterion, we proposed in section 3.2.2 to rank the C nodes of each path on basis of the EG_r they provide. We will now evaluate if this gives a ranking with the C nodes providing the best G_r in the first positions. To do this evaluation, for each path for which there exists at least an interesting shortcut, we will only consider that the first k C nodes of the ranking are detected by the criterion (for several values of the parameter k). For these subsets of the C nodes we will compute the difference between the G_r provided by the best existing shortcut and the G_r provided by the best shortcut in the subset. We will thus obtain one value (the result of the difference) for each path of the considered matrix. Then, we will build the CDF of these values obtained for all the paths of the matrix. These CDF (for different values of the parameter k) are given on figure 3.8.

The graphs named "no detection" on figures 3.8(a) and 3.8(b) give the CDF of the G_r provided by the best existing shortcut for each path of the matrix (for which there exists at least an interesting shortcut). Indeed, if there is no detection criterion applied, there is no shortcut detected and the

difference between the G_r provided by the best existing shortcut and the G_r provided by the best detected shortcut is the G_r provided by the best existing shortcut. By applying a shortcut detection criterion, we will be able to detect some shortcuts and, thus, to reduce that difference for some paths. Since the computed difference is smaller for more paths, the CDF will rise faster on the graphs.

The graphs named "all nodes/XDC" on the subfigures of figure 3.8 give the CDF computed by considering all the nodes selected by the shortcut detection criterion XDC (i.e., ADC or EDC). This is equivalent to using k = ∞. These are the best results that the given detection criteria applied on the given matrix can provide. We can see that ADC still gives better results than EDC considering the difference between the G_r provided by the best existing shortcut and the G_r provided by the best detected shortcut: with ADC, the difference is small for most of the paths (there are only a small part of the paths for which the difference is bigger than 0.2[5]) while this difference is generally bigger with EDC.

These very good results with ADC are obtained by considering all the nodes detected by the shortcut detection criterion. Let us see what is the situation if we keep only the first nodes of the rankings. The graphs named "k nodes/XDC" on the subfigures of figure 3.8 give the CDF computed by considering only the first k nodes of the rankings obtained by using the shortcut detection criterion XDC (ADC or EDC) as detected. The first thing we see is that even if we take only a few nodes in the ranking (i.e. 10 nodes), we obtain already good improvement compared to the situation whithout shortcut detection. We see also that ADC gives better results than EDC only if we keep a sufficient number of nodes: more than 50 nodes for Meridian, more than 20 nodes for P2PSim and more than 5 nodes for Planetlab[6]. Moreover, if we keep a sufficient number of nodes (100 nodes for Meridian, 20 nodes for P2PSim and 10 nodes for Planetlab), we obtain a result with ADC that is better than what we can obtain by considering all the nodes with EDC. The number of nodes to keep in order to obtain good results may seem important for Meridian but it represents only 4% of the total number of nodes.

Given those results we can say that, with ADC, considering only 5% of the total number of nodes in each matrix (that represents 125 nodes for Meridian, 87 nodes for P2PSim and 9 nodes for Planetlab), we are able to provide a significant improvement of the RTT for lots of AB pairs.

[5]That means that, for a small part of the paths AB, it is still possible to find another shortcut C' in order to improve of more than 20% of $RTT(A, B)$ the gain provided by the alternative path proposed by our shortcut detection criterion.

[6]Since there are only 180 nodes in the Planetlab matrix, the sets of C nodes returned by the criteria are quite small and keeping all the detected nodes is not really a problem. So, the quality of the ranking is less important for that matrix and we will not show the graphs here.

3.2.4 Conclusion and Future Work

We intend to test 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 and we will be able to provide a ranking of these nodes.

The last step of the work will be to use the detection results in order to improve the quality of the routing in the network.

3.3 Implementation of an Internet Coordinates System within XORP

3.3.1 Introduction

We first explain how we implemented an ICS module in XORP. Then we discuss some relevant aspects of the current implementation, some of them being related to already known problems such as peer-to-peer network bootstrapping, better choices of peers for ICS or measurement reliabilty. We also analyze the memory and performance cost of our module. Finally we explain how to improve and evolve this module for better integration and tight interaction with ECODE architecture.

The implementation currently done within XORP is the implementation of the Vivaldi algorithm[10]. By writing it, we aimed for two main goals:

- To actually write a first implementation of an Internet coordinate system. Even if it will not be the coordinate system eventually used within the ECODE projet, its implementation is a first but big step towards the final implementation of an ICS within the ECODE architecture, eventually embedding "our" coordinate system.

- To obtain programming skills with XORP and share this knowledge. Besides the implementation, a wiki dedicated to the XORP programmers of the ECODE projet has been initiated. Furthermore, mail exchanges between partners is frequent and a mailing list will probably be created for this purpose.

3.3.2 General overview

3.3.2.1 Module place within XORP architecture

Fig. 3.9 comes from the XORP documentation and represents the high-level architecture of XORP. In this diagram, each box stands for a XORP module and therefore a particular process (sometimes two, due to the differentiation of IPv4 and IPv6). Notice the three families of processes: unicast routing, multicast routing and management.

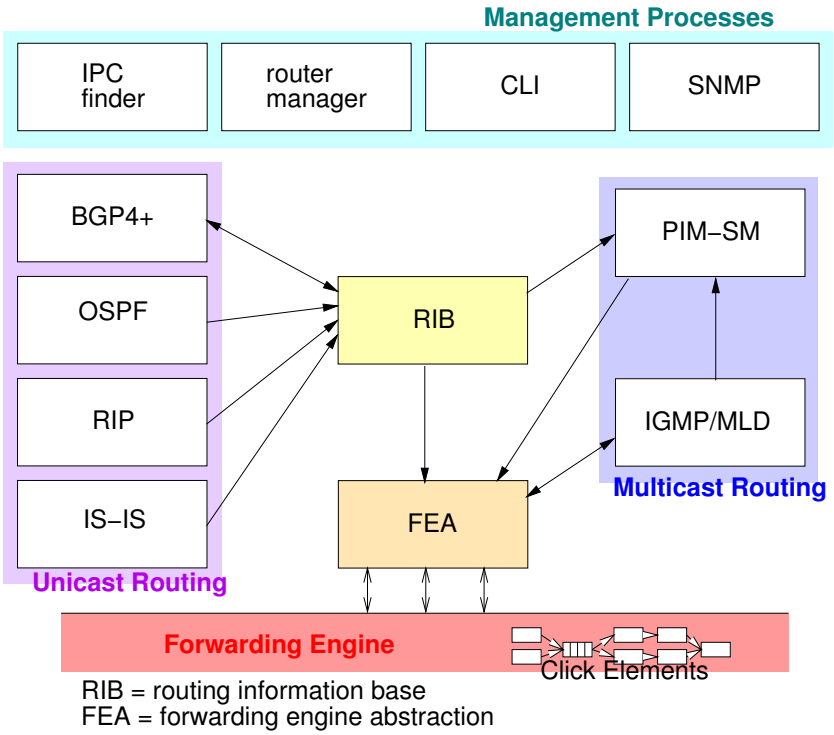


Figure 3.9: Overview of the XORP module organization.

A module such as an ICS module does not really fit in the routing processes (unicast or multicast) nor in the management processes. It is not surprising since the ECODE project aims to add some kind of cognitive level to a routing plateform such as XORP. So this module will be better suited in a new family of, for example, cognitive processes. Incidentally, other modules developed for the sake of ECODE will probably also fit in this class.

This module is currently mostly independent from other ones, having relations for the time being with only the Forwarding Engine Abstraction[7] (FEA) and some of the management processes, such as the Command Line Interface (CLI) or the IPC Finder[8].

[7]Which is somewhat similar to usual forwarding engine but permits to abstract the real operating system or the distributed organization.

[8]Which permits to organize the Inter-Process Call (IPC) scheme of XORP.

3.3.2.2 Organization of the Vivaldi Module

We will refer to Fig. 3.10 to explain the organization of the Vivaldi Module within XORP. While quite simplistic in regard to the class organization, it should be sufficient to give a good grasp on this module operation.

In Fig. 3.10, boxes model the instantiated classes while the Vivaldi module is running. The black arrows represent the method calls between these objects, and the larger gray ones, the inter-process calls.

For the sake of simplicity, IPv4 and IPv6 are not distinguished, although our Vivaldi processes would be different for these two flavours of IP. Anyway, most of the code is commonly used by the Vivaldi module for IPv4 or IPv6, thanks to the C++ templates.

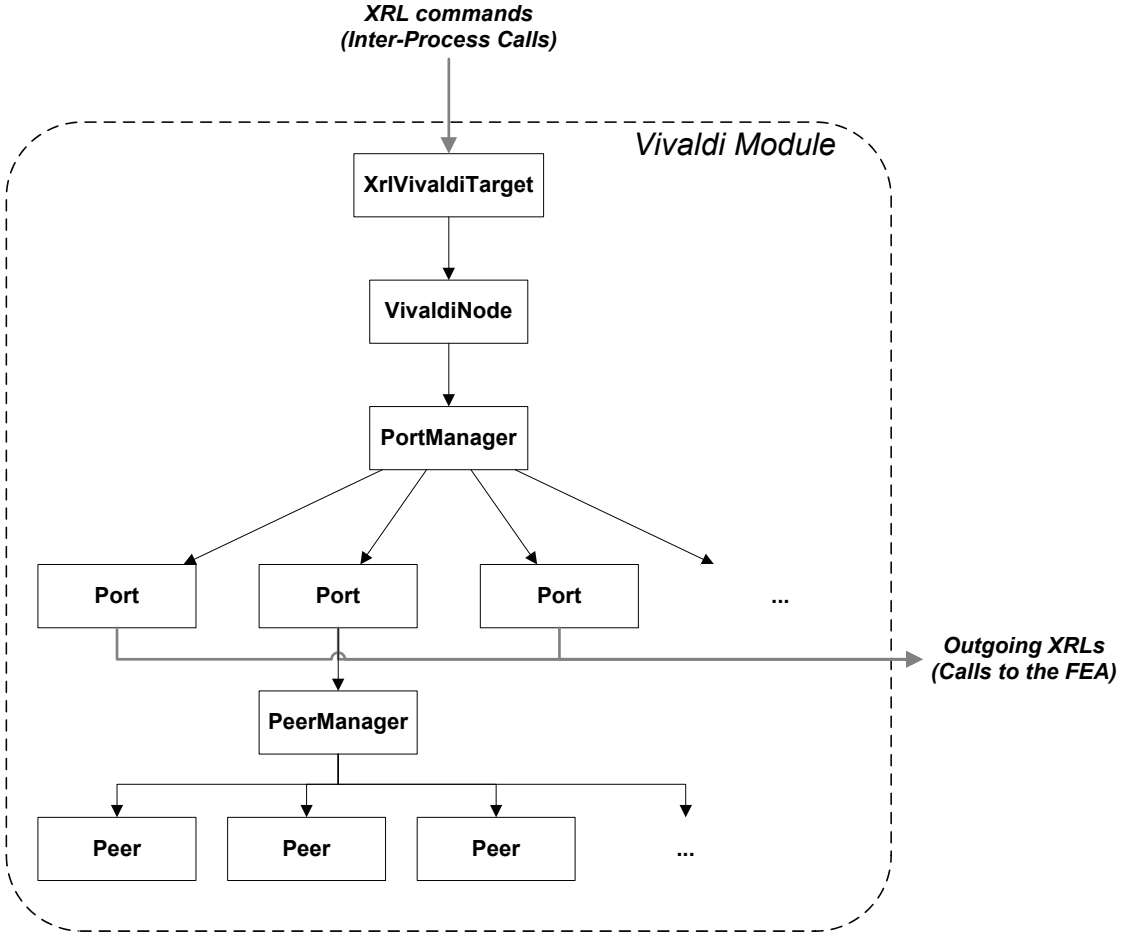


Figure 3.10: Snapshot of objects of the Vivaldi module.

Descending through the diagram of Fig. 3.10, we can reach a fast and intuitive comprehension of the overall working process of our module.

All commands and data[9] addressed to our module use inter-process calls

[9]Ingoing data received from sockets bound to our module arrive through this interface,

(IPCs) addressed to methods defined within the **XrlVivaldiTarget**[10]. Among other things, we can request to:

- start and stop the Vivaldi module,
- enable and disable operation of Vivaldi on particular network interfaces,
- ask for coordinates computed on relevant network interface.

The **VivaldiNode** is somehow the central class of our module. It holds an instance of the **PortManager** class which, as the name suggests, is responsible for *Vivaldi Ports*.

A *Vivaldi port* – an instance of the **Port** class – is responsible for the Vivaldi operation for a network interface on which Vivaldi is activated. By Vivaldi operation, we essentially mean coordinate computation but it also includes peer exchanges and measurement. So, there will be obviously one instance of this class for each of the network interface where Vivaldi is active.
Therefore, one router could obtain multiple coordinates. If one wishes instead to work with a unique coordinate per router, it is better to enable Vivaldi operation on the router address defined on its loopback interface (if defined, it will be most of the time identical to the "router ID"[11]).

Each of the **Port** instance binds itself to a UDP socket on the relevant network interface. This socket can then be used for sending probes, responses or other messages used by the Vivaldi module. All this (socket operations and traffic sending) is accomplished through XRL's, i.e., inter-process calls, to the Forwarding Engine Abstraction (FEA). The FEA is the module responsible for these operations in XORP.

A **Port** instance holds a **PeerManager** which is responsible for the **Peer**s associated with this Vivaldi Port. A **Peer** object essentially keeps track of address, coordinates and error of another node present in the network. We also keep the count of outgoing messages sent to this peer to detect vanished peer.

---

as it is usually the case with XORP modules.

[10] As the name suggests, XRL's (for XORP Resources Locators) are the preferred –and most recommended– means for inter-process communication.

[11] For more information on loopback interface and their interest, see for example [22].

3.3.3 Discussions

3.3.3.1 About message exchanges

For the sake of Vivaldi operation, we defined 6 types of messages. These messages are encoded using a simple Type-Length-Value (TLV) format. Incidentally, multiple messages can and will be aggregated into a unique datagram addressed to a remote node. These 6 types are, grouped by relevant pairs:

- ECHO and REPLY,
- ASK_CLOSE_PEERS and REPLY_CLOSE_PEERS,
- ASK_RANDOM_PEERS and REPLY_RANDOM_PEERS.

An ECHO message will simply be composed of a timestamp. When received by the remote node, this one will include this timestamp within its REPLY-type message together with its coordinates (and its estimated error). When the REPLY message arrives to the initial node, we can then update the RTT and the coordinates associated with the remote node, and process the Vivaldi algoritm to compute the new local coordinates and error.

The ASK_CLOSE_PEERS, resp. REPLY_CLOSE_PEERS, permits to ask, resp. to provide, several of the closest peers. The couple formed by ASK_RANDOM_PEERS and REPLY_RANDOM_PEERS acts similarly with random peers from the pool (i.e., the set of peers that one actually knows).

3.3.3.2 About bootstrapping

The bootstrapping problem arises in all Peer-to-Peer (P2P) networks and Internet coordinates systems which adopted this scheme, such as Vivaldi, are no exception. The simplest solution consists in using bootstrapping nodes (or servers) which are statically defined. For boostrapping a Vivaldi module, a command is provided to add the address of another Vivaldi node.

As it is the simplest solution, it is the most commonlt used too. However it is not the more elegant. Indeed, while we deal with a P2P network where peers dynamically go and leave, having to anchor some of the peers somehow is against the underlying principle of such networks. A way to mitigate this is to associate bootstrapping nodes with a domain name rather than a static address. For the time being, this is not implemented in our Vivaldi module but it could be if the need arises.

3.3.3.3 About choosing the peers

From research on how to better choose peers for the Vivaldi Internet coordinate system, it seems that better results are achieved if the pool of peers is partly composed of peers in the close vicinity of the node and partly of more distant random peers[12]. By default, a **PeerManager** instance keeps 64 peers, where half of them are kept in a "close peers" vector and the other half is kept within a "distant random peers" vector.

For the time being, when a ECHO message is sent, we include in the same datagram requests for at least one close peer and for another random peer. Thus, replies to these requests are included within the datagram received from the remote peer. By doing so, we will often receive information about other peers present in the network coordinate system.

When a node receives information about a peer which is closer to itself than other peers present in the "close peers" vector, it will replace the more distant peer known in this vector by this new one. Doing so, we expect to eventually obtain a vector filled with close peers, if not the closest ones.

If the newly-known peer is not eligible to enter within the "close peers" but is precise enough, it rolls a dice to win an entry in the "distant random peers". The odds are small but if the peer is lucky, it replaces the less precise peer[13].

3.3.3.4 About measurement

As we can see from the organization of our ICS module, "echo" messages are sent by **Port** instances and "reply" messages are received through the **XrlVivaldiTarget** instance. So, measurement are made at the control plane of the router. It could be fine if we were not working with XORP, i.e., an asynchronous and distributed architecture. It is hard to say which delay will be added before incoming datagram, and timestamp, will be processed; or worse, which delay will be added if the cognitive module is on another device than the one receiving the probe.

Ideally we want to measure RTTs from an network interface on a node to another. Therefore we cannot let this be done by some high-level module which may not even be on the destination machine for probes. We expect to have access to low-level local Monitoring Points to further improve our module organization and get rid of this drawback. More on this will be discussed within the "Future work" section.

[12] See [23] for more on this subject.

[13] To augment differentiation between close and distant peers, we could also eliminate the closest peer present within the vector. Or mix these approaches.

3.3.4 Evaluation

3.3.4.1 Memory cost

The memory cost of this module is quite small. In addition to a few parameters, data members and temporary values needed within classes, we essentially keep coordinates and peer information. The module keeps these data for each network interface on which Vivaldi is activated. By default, a peer pool size is of 64 peers. If we add the local peer, we have a memory cost of 65 N bytes for each Vivaldi port, where N is the weight of a **Peer** instance and should be at most two or three dozens of bytes. For a reasonable amount of Vivaldi ports – which is limited by the number of network interfaces – the memory footprint is thus really not important.

For more complex coordinates, the memory used could be augmented by some factor (probably one-digit). It should obviously not be a problem.

3.3.4.2 Performance

The burden put by this module on the platform essentially depends on two factors. We can separate the load due to the requests initiated by the local node and the load due to requests initiated by remote nodes.

Locally, the number of requests addressed to peers is dependent on the relative error. Indeed, request interval is inversely proportional to the error, with some arbitrary fixed lowest and highest values which are set, for the time being, to 100ms and 10s.
We expect this error to be big during the first iterations of the algorithm and to decrease to reach a small value during normal operation. Therefore, numerous requests will be initiated at first and will be more spaced as the coordinates permit better prediction of RTTs. The processing of a reply is fast, with the update of remote coordinates and the processing of the Vivaldi algorithm to update the local coordinates.

What is less predictable is the burden put on a node by its peers. Since the “close peers” vector obviously aims to be filled with close peers, it is likely that these peers will keep peering relationship with the local node too. However, we have no idea of how many Vivaldi nodes will pick the local one as peer. For the time being, we impose no limitation but in the future we could include some to avoid overusing local resources.

3.3.5 Future work

We intend to improve and evolve our implementation in two steps.

First, we want to split ICS operations from monitoring, i.e., active probing. The main practical reason is that monitoring will eventually be made by Monitoring Points under the TCI responsibility. By separating these aspects, we will more easily adapt our module to the use of the TCI and integrate the ECODE architecture. Notice that the adoption of the TCI which would be responsible for low-level Monitoring Points present in forwarding plane should solve our problem of reliable measurements.

Second, instead of placing lightweight ICS modules within each router whose coordinates should be computed, we would like to host the ICS algorithm computation remotely for a set of routers, e.g. for all edge routers in an Autonomous System (AS). Doing so, this particular ICS server would be in charge of the coordinates of these routers. This single ICS entity would also be responsible for coordinate exchanges with similar entities from other ASes. Incidentally, coordinates would no longer be known locally, within the router they are associated with.

Several advantages of this approach would be scalability, minimal modification to core routers and easier access to coordinates. The ICS module will be able to communicate with the TCIs located in several routers in its domain and ask them to steer RTT measurements[14] to other routers (internal or external to the domain) equipped with a similar TCI, and then collect the measured delays. This architecture would not require to host an MLE in the routers and modifications on core routers in particular should be minimal. Also, since coordinates would be centralized on a domain basis, it will be easier to get and collect sets of coordinates.

[14]Therefore we expect these routers to have at least one delay monitoring engine and one TCI; the latter being used to communicate with the remote ICS module.

Chapter 4

Minimizing packet loss during re-routing

4.1 Introduction

The goal of this chapter is to improve the quality of the IP router recovery process by proposing i) the formalization of the IP router update process in relation to the network traffic dynamics it undergoes during the process, and ii) the design and evaluation of heuristics to optimize the characterized process in terms of packet loss decrease.

This chapter is organized as follows: Section 4.2 characterizes the problem and the opportunity of traffic-informed recovery and describes the existing research in that context. The next section formalizes the impact of dynamic traffic conditions on the formalized router update process, and Section 4.3.2 describes heuristics to improve the IP router recovery process. In Section 4.4 network traffic dynamics is analyzed for short periods of time and state-of-the-art network models are presented to order and predict these dynamics. The defined paradigms are then combined in a realistic simulation setting described in Section 4.5 where the performance of the defined heuristics is evaluated making use of realistic network traffic models. At last future work and a conclusion is given in Section 4.6

4.2 Problem statement

OSPF and IS-IS are Link State (LS) Interior Gateway Protocols (IGP) commonly used in today's IP networks. These protocols were designed in times when computer communication networks were only used for research purposes and best-effort service was sufficient. IGPs were designed to allow routers to automatically compute and configure routing and forwarding

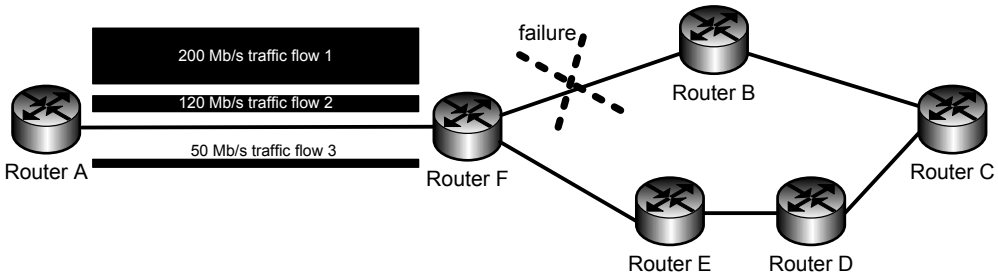


Figure 4.1: IP backbone router about to update entries for three traffic flows

tables without consuming too much CPU time during network instabilities. Whereas LS convergence times in the early times could take tens of seconds, nowadays, sub-3-second and sub-second convergence times are usual in large IP backbone networks ([24]).

Whereas research and development efforts have been undertaken to further reduce the reconvergence time, few studies take into account the specific characteristics of the network traffic that is received and forwarded in order to improve the recovery process. Figure 4.1 illustrates the heavily simplified situation of IP backbone router F needing to forward 3 traffic flows from Router A towards Router C (from where the traffic flows will split towards their exact destinations). Whereas the shortest forwarding path for all these traffic flows is via Router B, a failure has occurred breaking connectivity between the backbone router and Router B. As will be handled in more detail in the coming sections, the default IP re-rerouting process randomly updates the routing entries corresponding to the three traffic flows (e.g. updating the flows in order (2,3,1)). However packet loss occurs as long as the entries are not updated (traffic gets black holed). This means that in case the routing entry corresponding to the traffic flow of 200 Mb/s is only updated as last flow after one second, that 200 Mb has been lost for this traffic flow during the recovery period, while clearly it would have been better if the update order would have been (1,2,3).

While in this example only 3 routing table entries need to be updated, it is clear that in a realistic scenario of an IP backbone with 10 Gb links where routers need to update 5000 entries upon failure detection, packet loss minimization can make a big difference. However, a few aspects make the problem more challenging than is represented above: i) can the low-level system design of the router help in minimizing the resulting packet loss, ii) network traffic flows evolve dynamically over time, such that the router needs to have an up-to-date view on the bitrate of monitored network traffic flows, iii) for the same reason, the relative bitrate ratio's of these flows change over time, even during the recovery process network traffic can change.

4.3 Formalization of the RUP under changing traffic conditions

The RUP optimization heuristics described in D34, assumed that the network traffic bitrate remains constant during the IP router update process. While this is a convenient assumption for experimentation, it is more realistic to model the process under dynamic traffic conditions, meaning that the bitrate of a flow can change over time during the RUP. Therefore, the *associated traffic flow rate* is now referred by $br(f_i, t) : F_n \times T \rightarrow R$ of the affected flows in byte.

4.3.1 Packet loss

While the recovery time as formalized in D34 remains the same under either static or dynamic traffic assumptions during the RUP, the formula for packet loss slightly changes for a flow f_i, having a continuous bitrate described by a time series function $br(f_i, t)$, the loss is the following::

$$loss(f_i) = \int_0^{r(f_i)} br(f_i, t)$$

Upon failure occurrence, as long as an affected traffic flow is not recovered, packet loss occurs. The loss is proportional to the bitrate of the traffic flow during the event of the update. Figure 4.2 shows the loss as experienced by 2 concurrent flows during their recovery. For the entire entire set of flows F_n, this results into a total loss of:

$$loss(F_n) = \sum_i^n \int_0^{r(f_i)} br(f_i, t)$$

4.3.2 Heuristics for minimizing packet loss during the RUP

Assuming that we know the time series $br(f_i, t)$ of all flows (using prediction techniques), in this section we formulate heuristics or optimization functions having as goal to minimize the packet loss during the routing update process. Minimization of packet losses can be achieved by i) updating the IGP prefixes associated to a (set of) flows in a different order, and/or ii) bundling these prefixes in well-specified consecutive batches. As such, an optimization function is defined as a function which maps a given traffic model (set of well-defined $br(f_i, t)$-functions) at a certain time t to a tuple $(flowordering, batching)$. The first part of the tuple denotes a permutation of

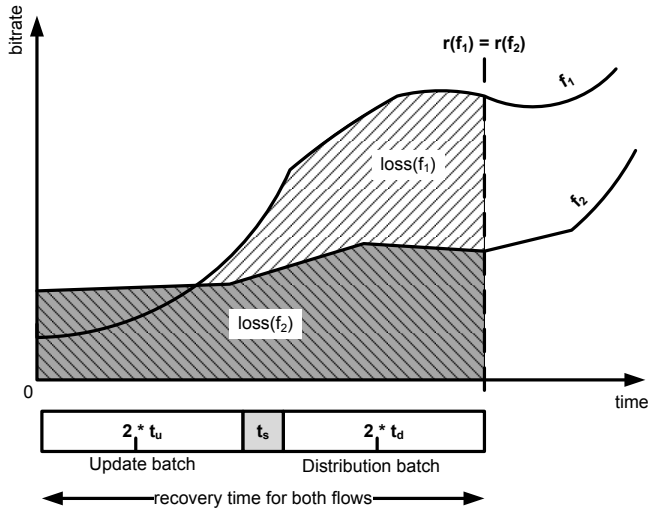


Figure 4.2: Packet loss under dynamic traffic conditions

the flows, the second part refers to a decomposition of the flows into ordered batches.

4.3.3 Fixed batch size

Given a fixed batch size n, techniques can be found to minimize the resulting packet loss under these conditions. For this purpose, let's define the cumulative loss function per flow:

$$cumloss(f_i, T) = \int_0^T br(f_i, t)dt$$

The following algorithm can now be used for fixed batch size Y: Calculate the total number of batches needed, and handle every consecutive batch starting with the first, starting with a working set of all flows:

1. Calculate the recovery time RT for the current batch
2. For every un-updated flow f_j, calculate $CL_j = cumloss(f_j, RT)$
3. Sort all CL_j's
4. Select the Y flows with the highest CL_j and finish the current batch for the selected flows
5. Remove the already batched flows from the working set and continue with the next batch

4.3.4 Variable batch size

It is clear that by using fixed batch sizes, the previous optimization algorithm minimizes packet losses by means of flow ordering only. In other terms, the algorithm relies on ordering the IGP prefix updates associated to a (set of) flow(s).

The earliest recovery time possible for a flow f_i (i referring to the queue-position of the flow) at time t is defined as follows:

$$ERT(f_i, t) = t + i(t_u + t_d) + t_s$$

Building further on the heuristic from D34, we can formulate a heuristic that takes into account both flow ordering and the batch size so as to minimize the packet losses. For this purpose, consider the following scenario. We can observe that in the middle of the ordered process of updating the IGP prefixes associated to the affected flows F_n, with our current batch containing a set of flows to be updated $b_{current} = (f_i, \ldots, f_{i+s})$[1], 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 can compare the additional cost of extension vs. the additional cost of finishing the update-distribution batch to guide us into the decision above. By defining $t_{bcurrent}$ as the starting time of the current batch $b_{current}$, the extension cost can be formulated as follows:

$$ec(b_{current}) = \sum_{j=i}^{i+s} \int_{ERT(f_j, t_{bcurrent})}^{ERT(f_{i+k}, t_{bcurrent})+(i+s-j+1)(t_u+t_d)} br(f_j, t)dt$$

This formula 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 to the minimal delay it can experience (compared to the earliest recovery time[2]), given the position of the flow. This additional delay, when multiplied with the associated bandwidth, allows deducing 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

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

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

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, has also 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:

$$fin_{b_{current}} = \sum_{f_j \notin b_{current}} \int_{ERT(f_j, t_{bcurrent})}^{ERT(f_j, t_{bcurrent})+2t_s.} br(f_j, t)dt$$

Our configuration strategy now consists in identifying the action with the least associated cost. The overall algorithm can now be expressed as follows:

1. Add all flows to the working set
2. Sort all flows in the working set in decreasing order of their current cumulative loss $cumloss(f_j, t_{current})$
3. Compute both extension and splitting cost
   - If no current batch exists (first flow), then create a batch and add the first flow into this newly created batch.
   - Otherwise
     - If the extension cost is smaller than the splitting cost, then add the first flow in the sorted list to the current batch;
     - Otherwise, create a new batch and add the first flow into this newly created batch.
4. Remove the added flow from the working set
5. Repeat the procedure from step 2 until the working set is empty

4.4 Modeling traffic dynamics

An accurate network traffic model is expected to capture the prominent traffic properties, e.g. short- and long- range dependence, self-similarity in large-time scale and multi-fractality in short-time scale. On the other hand, it has been observed that Internet traffic also exhibits non-stationary and non-linear properties. Therefore, in order to benchmark the given RUP-heuristics, one needs adequate traffic models fit and predict realistic IP backbone network traffic. This section summarizes the state-of-the-art of

the network traffic models as will be used for experimental evaluation in Section 4.5.

Network traffic analysis studies in the last decades have uncovered the subtle pattern of *self-similarity* in network traffic time series. Stochastic self-similarity describes a statistical property of the traffic and manifests itself in several equivalent fashions: slowly decaying variance, long range dependence (LRD), non-degenerate autocorrelations, and Hurst effect. Intuitively, a process is said to be self-similar if its statistical behavior is independent of time-scale. Formally, a time series $Y = \{Y_t | t \in T\}$ is self-similar if its autocorrelation function decays only hyperbolically instead of exponentially (see [25]). For self-similar processes, the autocorrelation function drops hyperbolically (not exponentially) toward zero but may never reach zero (non-summable auto-correlation function). The 'degree' of self-similarity is often denoted by the Hurst parameter, which is a measure of the persistence of a statistical phenomenon, denoting the length of the *long-range dependence* of a stochastic process.

After the seminal work reported in [26] confirmed mathematically in [25], it has been commonly accepted that Internet traffic behaves statistically self-similarly ([27, 28]) and that aggregating streams of such traffic typically intensifies the self-similarity ("burstiness") instead of smoothing it.

4.4.1 AutoRegressive Moving Average models

The root of most time series analysis and prediction models are based on the AutoRegressive Moving Average (ARMA) model. The $ARMA(p, q)$ model, linearly models a time series as follows:

$$y_t = \sum_{i=1}^{p} \alpha_i y_{t-i} + \sum_{i=1}^{q} \beta w_{t-i} + w_t$$

This formula combines two techniques i) auto-regression (AR(p) model, where p is the order of the autoregressive part of the model), which reflects the fact that a prediction is based on the signal itself (using p previous values) reflected by the second term, and ii) moving averages (MA(q) model, where q is the order of the moving average part of the model), reflected by the white noise series w_t (having $E(w_t) = 0$ and $var(W_t) = \sigma^2$) which is put through a linear non-recursive filter determined by the coefficients α_i (weighted average). The auto-regressive part directly reflects the Short Range Dependence (SRD) of a time series.

4.4.2 (Fractionally) Integrated models

Because ARMA models time series as stationary (the variance of the white noise process being constant over time), which is often not the case, several improvements have been made to the base ARMA model.

Some non-stationary time series can be made stationary by one or more levels of differencing[3]. Ones the resulting differenced time series is stationary, an ARMA-model can subsequently be fit and predictions can be made by integrating the predictions back. The resulting model of this approach is called the AutoRegressive Integrated Moving Average-model (ARIMA). The resulting model including the lag operator L[4] for ARIMA(p, d, q) is as follows.

$$(1 - \sum_{i=1}^{p} \alpha_i L^i)(1 - L)^d y_t = (1 + \sum_{i=1}^{q} \beta_i L^i) w_t$$

As shown in [26], Internet traffic exhibits a high degree of long-range dependence (LRD) properties in addition to short-range dependence (SRD); hence, the Fractional Auto Regressive Integrated Moving Average (FARIMA) process has been proposed that can capture both SRD and LRD to model and predict traffic. A FARIMA process describes both short- and long-range dependence simultaneously by generalizing the ARIMA process: the fractional parameter $-1/2 < d < 1/2$ determines the strength of the long-range behavior (the Hurst parameter $H = d + 0,5$) whereas parameters p, and q (and the corresponding coefficients) allow for modeling of short-range properties of the traffic. The fractional difference operator $(1 - L)^d$ is defined in terms of the Binomial Theorem as the series:

$$(1 - L)^d = \sum_{j=0}^{\infty} \binom{d}{j} (-1)^j L^j = 1 - dL + \frac{d(d-1)}{2!} L^2 - \frac{d(d-1)(d-2)}{3!} L^3 + \dots$$

If $d = 0$, the FARIMA(p, d, q) process coincides with the usual ARMA(p, q) process, whereas if d has an integer value it reflects an ARIMA process. In practice, the series[5] denoted by the binomial expansion of $(1-L)^d$ is curtailed at some suitably large L. This way, the FARIMA-model handles LRD using a parsimonious model notation.

FARIMA models were successfully used in [29] to predict video, and Internet traffic on a timescale of one second or larger. The self-similar character of network traffic was shown to be adequately captured using a FARIMA

[3] A differenced time series y_t generates a new time series of differences $z_t = y_t - y_{t-1}$

[4] applying lag operator L on y_t generates y_{t-1}, and a power i of L defines a similar recursive process of the order i

[5] The coefficients of the series can be written in terms of the Gamma-function: $\binom{d}{j}(-1)^j = \frac{\Gamma(d+1)(-1)^j}{\Gamma(d-j+1)\Gamma(j+1)}$.

model. However, at smaller time scales, depending on the specific traffic trace, it was shown that the signal-to-noise (SNR) was worse compared to larger time scales.

4.5 Experimental validation

4.5.1 Platform choice

In order to motivate the choice of experimentation platform, we recapitulate the phases of the packet minimization procedure for the router update process:

1. Monitor traffic going through an IP router (per prefix of fixed length aggregate all packet sizes within a predefined bin size)
2. Fit a network traffic model to the monitored traffic in order to make predictions on future prefix bitrates
3. Apply optimization heuristics on the predicted network traffic models in order to return a suggested prefix ranking and prefix batch composition
   (a) The prefix ranking determines the order in which prefix updates need to be processed
   (b) The prefix batch composition determines which prefixes need to be processed in a group
4. Executing the update of the affected prefixes according to the output of the previous step. This is a two-phased process (see deliverable D34):
   (a) Updating the RIB and FIB entries towards the central FIB (step 2 in Figure 1)
   (b) Distributing the updated entries towards the LFIBS (step 3 in Figure 1)

Evaluating such a multi-phased process in an emulation platform generates the following challenges:

1. Fitting network traffic models in real time to monitored network traffic data (phase b) is slow. This means that it takes up to 30 seconds per prefix. Given a working set of 5000 prefixes, it is not obvious to execute this in a feasible way on a standard Linux-based desktop machine, as parallelized machinery is most probably needed for this.

2. The value of optimization techniques in the use case becomes significant when:

   (a) Large throughputs (larger than 1 Gbps) and a high number of prefixes are affected, which is the case in an IP backbone router

   (b) Batch optimization techniques at the lowest processing level (milli- and microsecond level) can be executed

3. Typical commodity hardware and OS's such as Linux is not able to cope with real-time process quantum configuration to optimize the update/distribution process of prefixes as can be done on a real-time OS on a high-end commercial backbone router (typical throughput levels of the hard- and software is at most 1 Gbps).

4. The order in which prefixes can be updated is typically not a configurable parameter in XORP, and therefore this process should be redesigned

5. Even if some optimization can be done using XORP-based emulation platforms, the resulting gain will be minimal and not viewable in a demo-setting

Given the above observations, the experimentation of the optimization of the IP router update process is done in a custom simulation framework built around modules in C++, Python and R and not in a XORP-based emulation platform.

An interface was developed such that PCAP-trace files from the MAWI project [30] could be read by the system used to evaluate the efficiency of the modeled RUP with respect to a realistic network traffic setting. The PCAP-traces were then processed by a set of time series modeling tools to fit the traces to the state-of-the-art time series models as presented in Section 4.4. RPY2 was selected as the interface between Python and R [31]. The R-packages fARma, forecast, fGarch, and fracdiff were used for fitting the network traffic traces. All of the experiments ran on a regular desktop computer equipped with an AMD Athlon 3000 CPU and 4Gb RAM.

The experimental validation consists of several parts i) collecting, analyzing, and preprocessing network traces to transform them into a set of parallel time series (one per prefix), iii) modeling and fitting the resulting time series into a network traffic model, and iv) using the resulting network traffic model to minimize the packet loss during the RUP. The global process is shown in Figure 4.3.

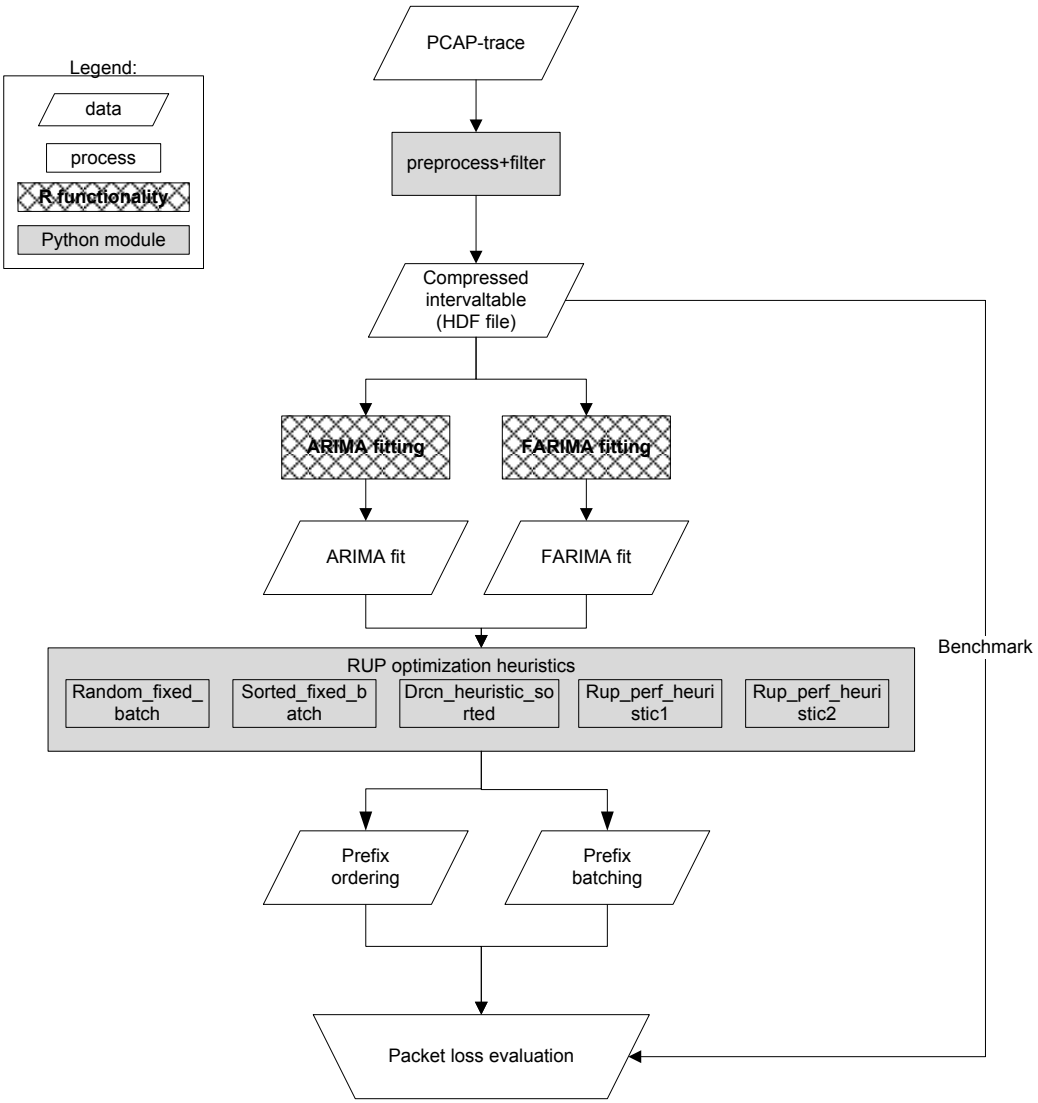


Figure 4.3: Global evaluation process

4.5.2 Network traffic analysis and preprocessing

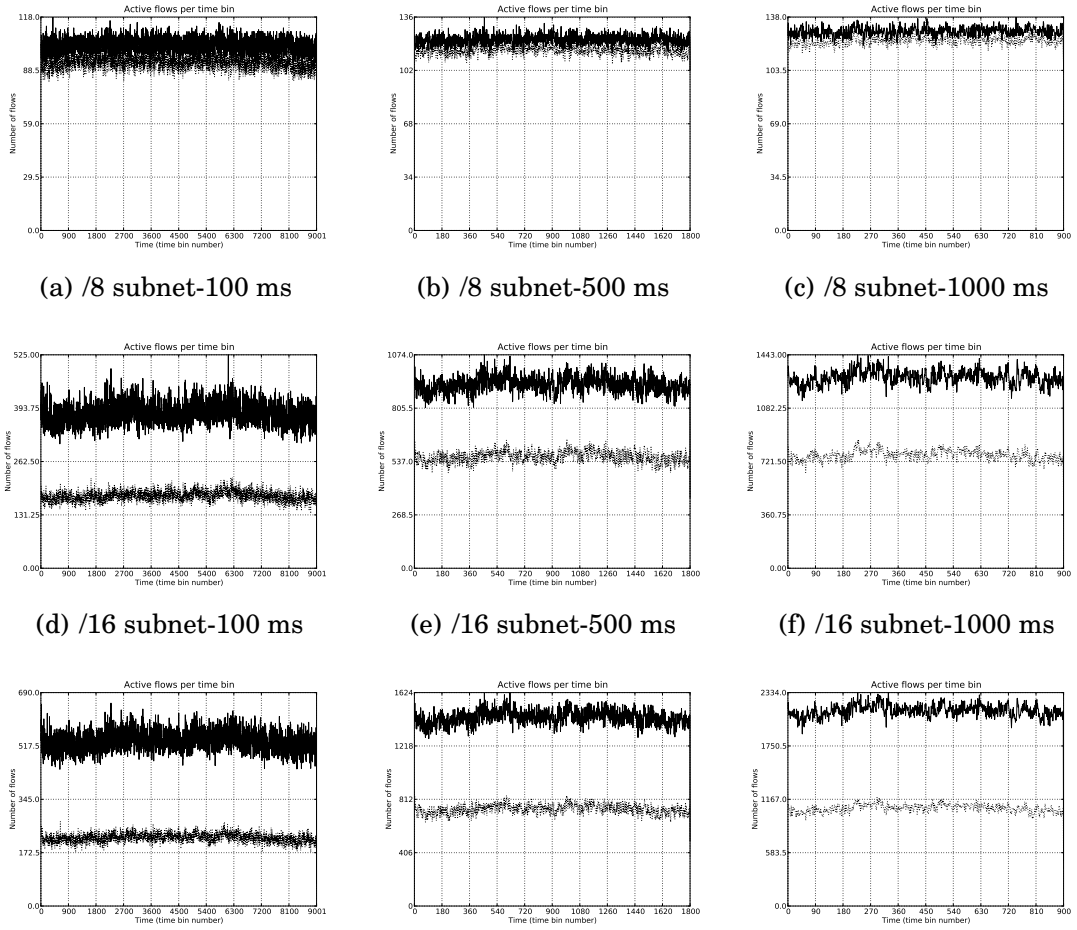
A random subset of tracefiles of Samplepoint-F of the WIDE backbone network (related to the MAWI-project) was analyzed. Samplepoint-F is a monitoring point on a trans-Pacific line (150Mbps link) in operation since 2006/07/01. Tracefiles are analyzed and modeled at several aggregation levels. The following aggregation levels were considered: spatial aggregation using /8, /16 or /24 IPv4 subnet prefixes and temporal aggregation (binning) using time intervals of 100 ms, 500 ms and 1000 ms. These 9 investigation subsets are generated by a preprocessing module which groups all accumulated packet sizes per subnet for each of the above time interval, generating a 2-dimensional *intervaltable*. This preprocessing provides on average 140K /24 subnets, 14K /16 subnets, and 0.21K /8 subnets for a 15 minute trace containing on average 30M packets.

Before applying network traffic models to a given trace, it is important to analyze the *activity* and *persistence* of prefixes at different aggregation levels. This analysis indicates if a subnet prefix (either /8, /16 or /24) has active packets being sent at some time interval and if this activity it still persistent for the same subnet prefix during the next time interval. Figure 4.4 shows for every activity level and the persistence for 9 combined aggregation levels. For all aggregation levels, both the activity and persistence level are approximately constant over time. At coarser aggregation levels (using for example /8 subnets), the persistence-level is close to the activity-level. However, the persistence level at /16 or /24 is significantly lower (only about 50 to 60 percent of the active flows stays active the next time interval). This high churn rate gives an indication on the high variability of time series corresponding to a subnet.

One of the main outcomes of the proposed RUP-optimization techniques is about reordering the update of prefixes. The idea behind is that, if high bitrate prefixes are updated earlier, then the resulting packet loss will decrease. In this context it is interesting to investigate the *spread* of bitrate among several prefixes at several points in time at several aggregation levels. This is shown in Figure 4.5 for 24-bit subnets. The figure makes clear that although the majority of the prefixes is contained in a smallest bitrate histogram part, that half of the bandwidth is taken by other prefixes. This is the main motivation for optimizing the RUP.

4.5.3 Fitting traffic models

The experiments we performed restrict each investigation set to at most 5000 prefixes. Based on [24], this number of prefixes provides a realistic assumption on the number of prefixes affected by a failure. Whereas existing network traffic studies mostly analyze and model global network traffic aggregates at some point in time, resulting into one smoother time series to

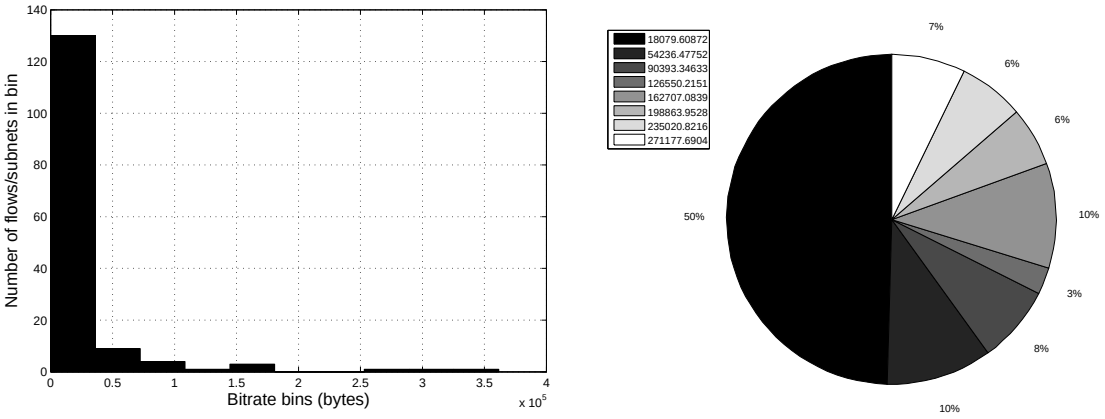


(a) /8 subnet-100 ms (b) /8 subnet-500 ms (c) /8 subnet-1000 ms

(d) /16 subnet-100 ms (e) /16 subnet-500 ms (f) /16 subnet-1000 ms

(g) /24 subnet-100 ms (h) /24 subnet-500 ms (i) /24 subnet-1000 ms

Figure 4.4: Flow activity and persistence per aggregation level



(a) Histogram of bitrate sizes of subnets (b) Relative part of the bandwidth of different subnet bins

Figure 4.5: Composition of bandwidth into several aggregated traffic flows (24-bit subnets)

be fit, our study tries to fit several (i.e., 5000) parallel time series at a time. This approach asks for a unified and optimized fitting strategy.

The following models were evaluated for fitting traffic: ARIMA(p,d,q) and FARIMA(p,d,q) models. Because our task consisted of fitting many parallel time series, we did not manually inspect the Auto-Correlation Function (ACF) and Partial ACF (PACF) of many of these series as it is usually the case in time series modeling. For ARIMA(p,d,q)-model fitting, we relied on the automated procedure as specified in [32] to find the optimal values for p, d and q restricted to maximum values of 5. Once these values are determined, the Maximum Likelihood Estimator (MLE) was used to find the best coefficients corresponding to these models. For FARIMA(0,d,0)-model fitting, we used the MLE as proposed in [33].

Table 4.5.3 summarizes the average fitting performance of the above models to the subsets of time series of 5000 prefixes. To estimate their respective performance, we use two metrics: normalized mean-square error (NMSE), and the Pearson correlation (R), as defined by the following formula's.

$$NMSE = \frac{\sum_n (\widehat{y}(x) - y(x))^2}{\sum_n (y(x) - \mu_T)^2}$$

$$R_{X,Y} = \frac{cov(X,Y)}{var(X)var(Y)}$$

The NMSE gives an indication of how much of the variance of a signal can be accommodated by a model, the smaller the NMSE, the better the fit. The correlation coefficient indicates the degree of linear relationship between two variables. This is a variable between 0 and 1. Higher means more correlation. From Table , it is clear that the FARIMA-models provide a better fit to the network traffic than the ARIMA-models. This observation further indicates, that even on the level of prefixes, a long-range dependence can be observed in the network trace data. However, if we compare these numbers to network traffic studies using the same methods on the global network traffic bitrate, these numbers are considerably lower. This result is not surprising, given the high churn rate of active prefixes as shown in Section 4.5.2. Figure 4.6 shows a traffic time series corresponding to /8 prefix and its fit to an ARIMA(2,1,2)-model, compared to the fit to an FARIMA(0,d,0)-model.

4.5.4 Packet-loss minimization

Given the best fitting traffic model and their expected predictive value as detailed in Section 4.3.2, we quantified the gain one can achieve by using the heuristics defined in Section 5. As described in the next subsections,

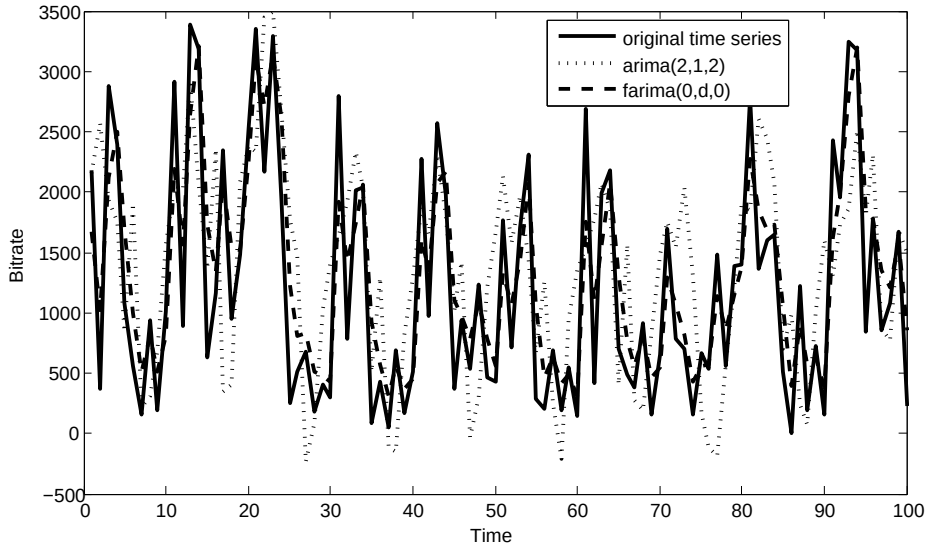


Figure 4.6: Time series fitting of ARIMA vs. FARIMA using /8 subnet and .1 s binsize

Table 4.1: Model fitting error

technique	subnet	binsize	MNSE	R
arima	8	0.1	2.84E-05	0.17
arima	8	0.5	2.64E-05	0.38
arima	8	1	2.49E-05	0.41
arima	16	0.1	1.01E-05	0.06
arima	16	0.5	8.89E-06	0.13
arima	16	1	7.98E-06	0.14
arima	24	0.1	4.82E-05	0.06
arima	24	0.5	4.51E-05	0.11
arima	24	1	4.24E-05	0.13
farima(0,0)	8	0.1	8.67E-07	0.97
farima(0,0)	8	0.5	6.25E-06	0.90
farima(0,0)	8	1	9.40E-06	0.84
farima(0,0)	16	0.1	1.27E-06	0.97
farima(0,0)	16	0.5	2.71E-06	0.91
farima(0,0)	16	1	3.04E-06	0.86
farima(0,0)	24	0.1	2.10E-05	0.92
farima(0,0)	24	0.5	2.62E-05	0.87
farima(0,0)	24	1	2.82E-05	0.82

several parameters (such as the batch size and swapping time) were evaluated against their respective effect on the resulting packet loss in the given dynamic traffic situation. Five routing update strategies have been evaluated:

1. *random_fixed_batch*: the default RUP-algorithm that most IP routers use today. It consists of using fixed batch sizes and randomly (with respect to the order) updating the prefixes upon failure occurence.
2. *sorted_fixed_batch*: similar to random_fixed_batch, but the prefixes are updated in decreasing order of bitrate as they were measured by the router in the last time instance (this strategy assumes persistence in bitrate and activity).
3. *rup_perf_heuristic1*: the strategy described in Section 4.3.3
4. *drcn_heuristic_sorted*: the strategy described in D34, this algorithm also updates prefixes in decreasing order of bitrate as they were measured, and in addition tries to optimize the batching configuration by trading off extension vs. splitting cost on persistent traffic assumptions
5. *rup_perf_heuristic2*: the strategy described in Section 4.3.4

These strategies were evaluated against several parameters of the modeled update process. The followings aspects were parameterized in the process: batch size of the RUP, bin size of the traffic aggregation, subnet of traffic aggregation, used traffic model for prediction and process swapping time (t_s). In order to obtain representative results, each specific parameter combination was repeated 100 times over different time slices of the traces (simulating each failure in time).

4.5.4.1 Strategy vs. packet loss/recovery time

The major interest of this paper lies in the online minimization of packet loss upon failure detection in an IP router by the use of optimization techniques. As can be observed in Figure 4.7, conform earlier studies ([34]), the default strategy of randomly updating prefixes using fixed batch sizes, performs the worst of all strategies in all settings. More surprising is the order and the 'spread' of other RUP strategies. Real optimization techniques quickly gain about 10 percent average decrease in packet loss compared to the random strategy. However, whereas the gain from dynamic optimization strategies such as *rup_perf_heuristic1* en *rup_perf_heuristic2* is clear, it is smaller than one would expect. Most probably the reason for this can be found in the highly variable character of short-term network traffic. Another aspect that becomes clear from the figure, is that optimization in terms of packet loss, does necessarily lead to smaller global recovery times.

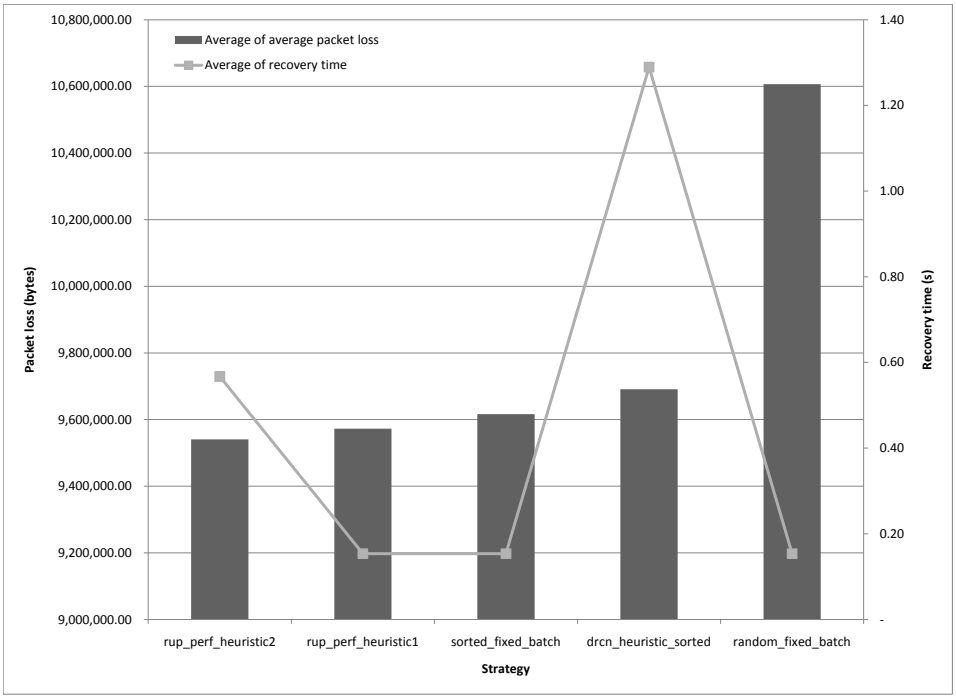


Figure 4.7: RUP strategy vs. loss

This must be understood from the fact that updating and distributing an high bitrate prefix can result into additional waiting time (because of the process swap) for other prefixes. If this process repeats itself, the global recovery time can increase considerably, as can be seen for the strategies *rup_perf_heuristic2* and *drcn_heuristic_sorted*.

4.5.4.2 Batch size vs. packet loss/recovery time

The first three RUP-strategies assume fixed batch sizes in their processing. Figure 4.8 evaluates the effect of using either large or small batch sizes in these strategies. The figure indicates that, except for the random strategy, no significant difference in packet loss or batch size is induced by the batch size.

4.5.4.3 Traffic model vs. packet loss

In Section 4.5.3 the fitting performance of the FARIMA-models was better compared to those of ARIMA-models, when applying them the set of considered MAWI-traces. Figure 4.9 shows that the first heuristic *rup_perf_heuristic1* is considerably less sensitive to the fitting performance of the underlying traffic model than the second *rup_perf_heuristic2*. This can be explained from the fact that the second heuristic makes decisions about the batching and prefix ordering strategy very often (per prefix) compared to the first heuristic (per batch). For *rup_perf_heuristic2*, every bad prediction can result into a wrong batch split and extension, which cannot be undone any-

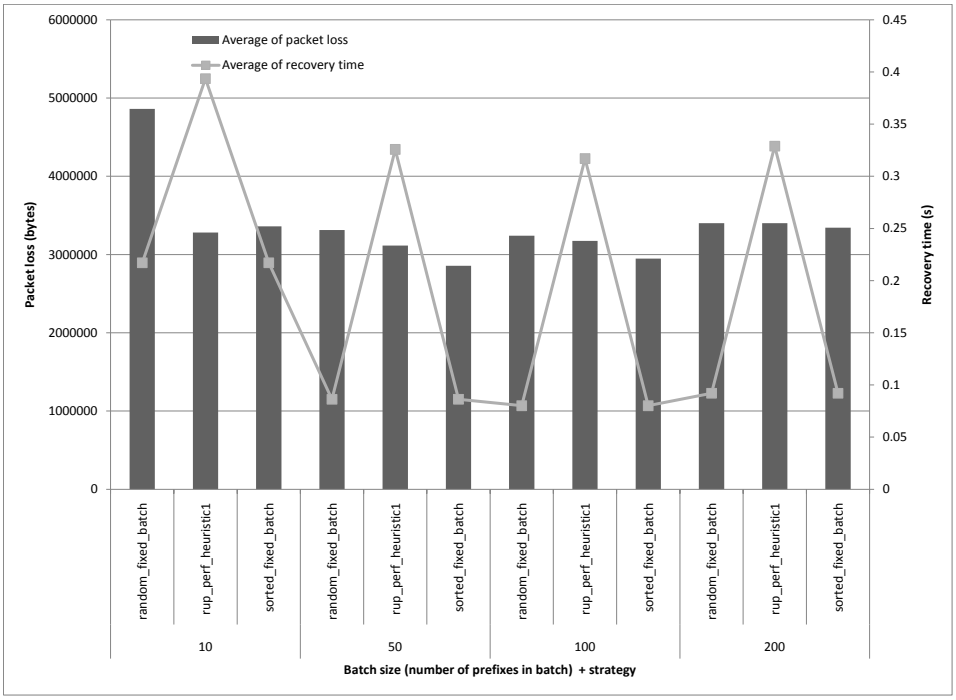


Figure 4.8: Batch size vs. packet loss

more. However, if predictions are fine, the packet loss performance is the best that was obtained.

4.5.4.4 Swapping time vs. packet loss

From Figure 4.10 can be seen that increasing processing swapping times do not have a severe impact on most RUP strategies. However a small packet loss increasing trend can be observed in general for larger values for t_s. A similar trend can be observed for the corresponding recovery times. Again the *drcn_heuristic_sorted*-strategy has the worst sensitivity to the swapping times. This can be understood from the fact that wrong batch-split or batch-extend decisions are penalized higher for larger swapping times.

4.6 Conclusion

The process of an IP router updating its routing entries when a failure is detected taking into account the dynamic setting network traffic was formalized. State-of-the-art ARIMA- and FARIMA-models were studied and applied to fit IP backbone network traces at very short-time scale at different aggregation levels. By the design and evaluation of several heuristics, we have shown that about 10 percent decrease in packet loss can be obtained in realistic, dynamic contexts of the IP router update process.

The described process can be integrated in the ECODE architecture as

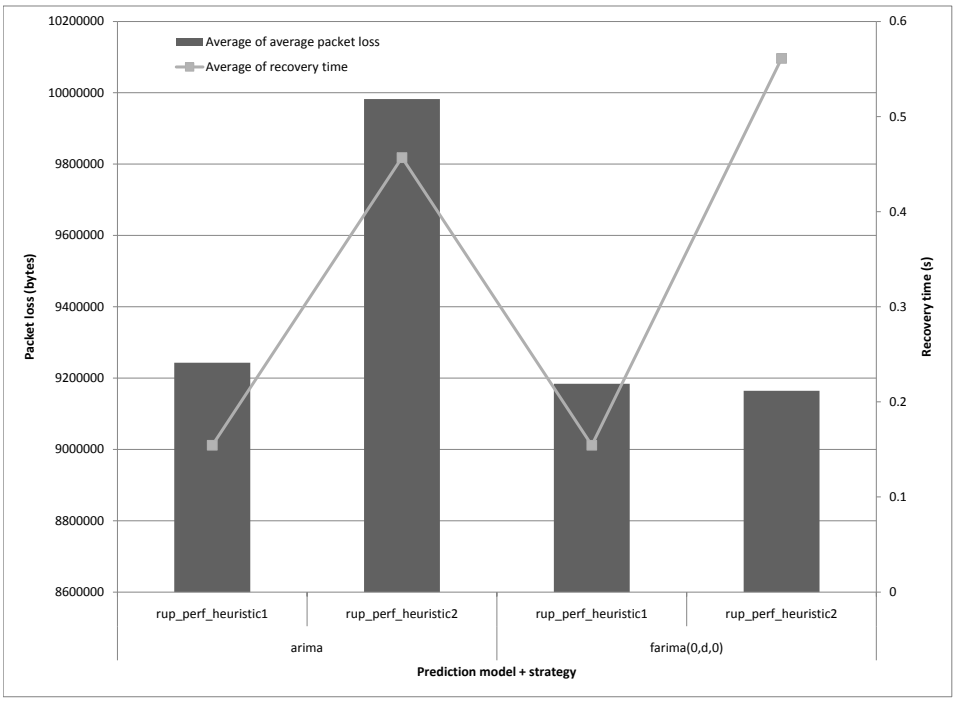


Figure 4.9: Traffic model vs. packet loss

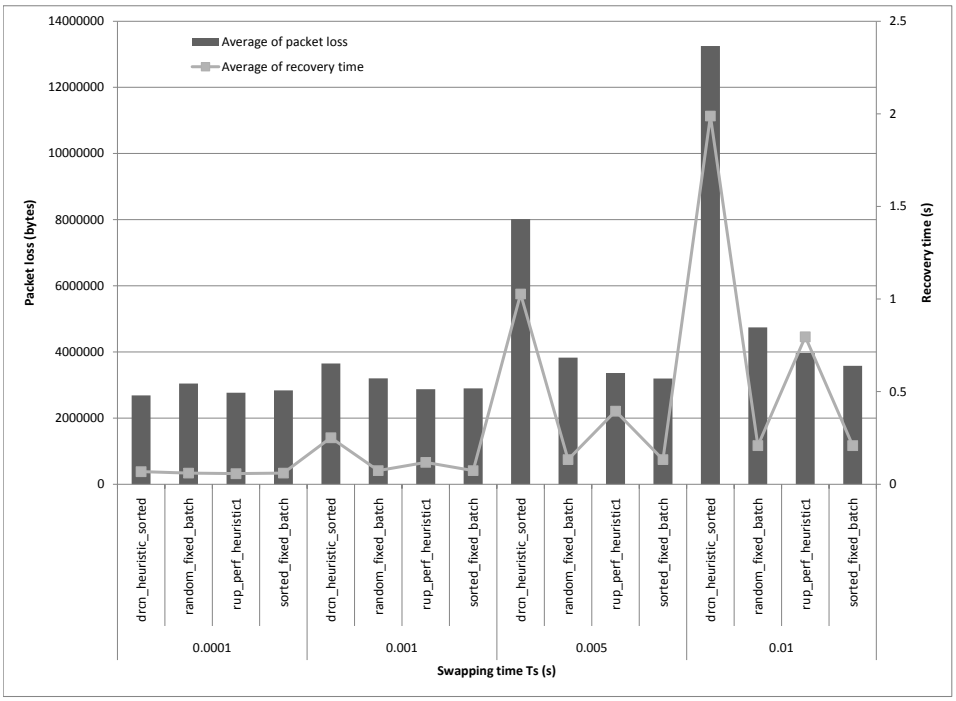


Figure 4.10: Swapping time vs. packet loss

follows: the Monitoring Engine (ME) is responsible for keeping track of the bitrates per prefix at a given bin size and subnet-length. This data is sent to the Machine Learning Engine (MLE) in order to build traffic fitting models (ie. ARIMA and FARIMA models) in background. When a failure occurs, the Routing Engine (RE) forwards a optimization request to the MLE which replies with the optimal prefix ordering and prefix batching to be executed by the low level RE and FE.

The current study focused on the minimization of packet loss without taking into account the computational cost of the underlying model. A future study could consist of quantifying and optimizing the computational steps needed in order to achieve the performance of the strategies presented in this paper. Another topic for future study could be to improve the *rup_perf_heuristic2* strategy, by i) improving its robustness to errors in the prediction model, or ii) by evaluating other prediction models such as ARIMA-GARCH-models ([35]).

Chapter 5

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

5.1 Formalization of the technical problem

In the previous deliverable ECODE deliverable 3.3 we have introduced a novel state space based machine learning technique to identify shared resource link group (SRLG). In this deliverable we further extend and generalize that method and produce results using the developed algorithm that proves the effectiveness of our proposed SRLG identification scheme. Before describing the algorithm in formal details, we first provide an example network that will help us to better understand the proposed algorithm. Fig. 5.1 shows a typical network topology with 6 nodes (routers) and 9 links. Fig. 5.2 shows a typical link state update sequence for the same network with links D, B and H failing. With multiple link failures, the arrival time sequences of the received LSUs are grouped together as depicted in Fig. 5.2. To identify SRLG, it is then required to find correlation pattern among LSU time sequences. We start our algorithm with crudely grouping time sequences and declaring them as SRLGs. For an example, links (D, B), (H, D, B), (H, D) and (B) forms SRLGs from Fig. 5.2 at the beginning. We next form a probabilistic model to represent SRLG in a more realistic fashion.

In the next three subsections, we describe the three phases of our proposed SRLG detection and identification algorithm and illustrate it with representative examples.

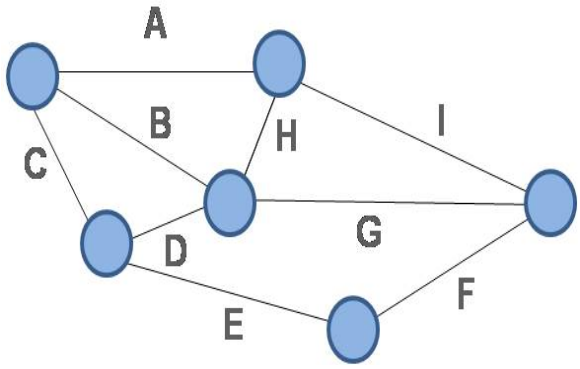


Figure 5.1: Example network topology

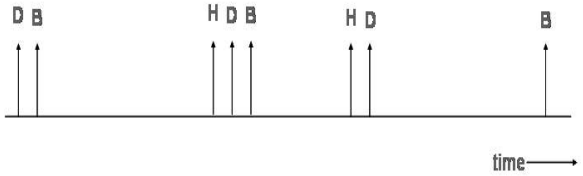


Figure 5.2: Link state update sequence for different link failure

5.2 Learning phase

In the learning phase, we construct a new Bayesian network based state space model. The state space model looks like one shown in Fig. 5.3. Each node of the model represents a set of links that forms a particular SRLG. The bottommost tier signifies isolated links forming their own SRLG. As we move higher through the state space, the number of nodes per SRLG increases. Each of the nodes is connected to one or more upper tier node. These connections specify the transition possibilities which mean that an observed SRLG in a particular state space node has certain finite probability of having an additional link included in the same SRLG.

In Fig. 5.3, p_B, p_D, p_H represents the probability of a link (B, D and H respectively) being an isolated link forming their own SRLG. Whereas, state (B, D) and state (D, H) represents two SRLGs with two member links each. In that context, p_{BD} and p_{DH} represents the probability of (B, D) or (D, H) forming SRLG without the possibility of including any further links in their respective SRLGs. The transition probability from state B to state (B, D), $p_{B \rightarrow BD}$, represents the probability of finding an SRLG with member links B and D, when an isolated link failure B is initially observed. Each of the transitions in the state space model is associated with a number that indicates the frequency of the observed phenomenon during the learning phase. The first part of the learning phase consists of grouping together LSUs from the LSU time sequence. This can be performed in two different ways:

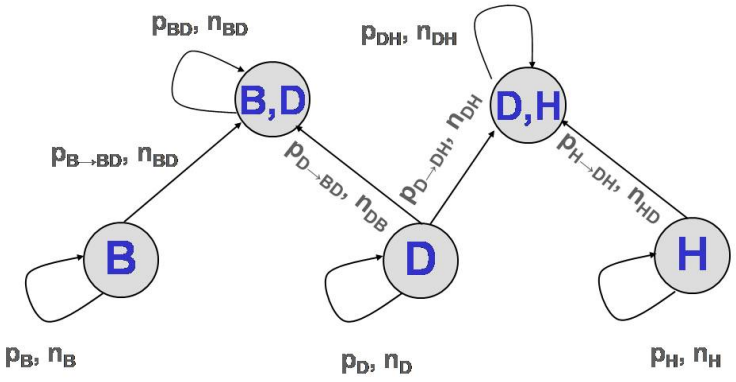


Figure 5.3: State space model formulation example

1. We define a time threshold and the LSUs that are within the elapsed time interval are grouped together. The time threshold is calculated as the addition of the maximum synchronization mismatch between routers, maximum propagation delay and queuing delays. These delays may vary from network to network and has to be manually entered while installing the machine learning component.

2. We initially start with a time threshold which is referred as window threshold. The start time can be any reasonable estimated guess (5 sec. for our case), as it has very little impact on the adaptive algorithm. The window threshold is then adaptively modified as follows. We define T_{min} as the minimum value of timing threshold. We start the algorithm with threshold time (T_{th}) equals to T_{min}. As a new LSA arrives the grouping algorithm starts and set a timer to T_{min}. There can be two possible cases:

a) No more LSA arrives within T_{min}: The algorithm stops further grouping and keeps $T_{th} = T_{min}$.

b) N number of LSA arrives within T_{min}: The algorithm increases the threshold time as follows: $T_{th} = T_{min} + f(N) \sum_{j=2}^{N} T_{interarrival(j,j+1)}$, where $f(N)$ denotes the function of N and denotes the inter arrival time between j^{th} and $(j+1)^{th}$ arrival. Under condition (b) (i.e., when multiple arrivals of LSAs are observed) once the new T_{th} has been calculated and when the algorithm waits for the added threshold time for further LSA arrivals, there can be two different scenarios:

I. No further LSA arrives as T_{th} elapses: If $f(N) \sum_{j=2}^{N} T_{interarrival(j,j+1)} < T_{min}$, then T_{th} is set to T_{min}; otherwise and the algorithm stops grouping LSAs. The current T_{th} is stored for the next phase of grouping.

II. N number of LSA arrives within the extended T_{th}: T_{th} is further extended following the process (b) described above (i.e., N number of LSA arrives within T_{min}) and the algorithm further waits for new LSA arrivals for this extended period.

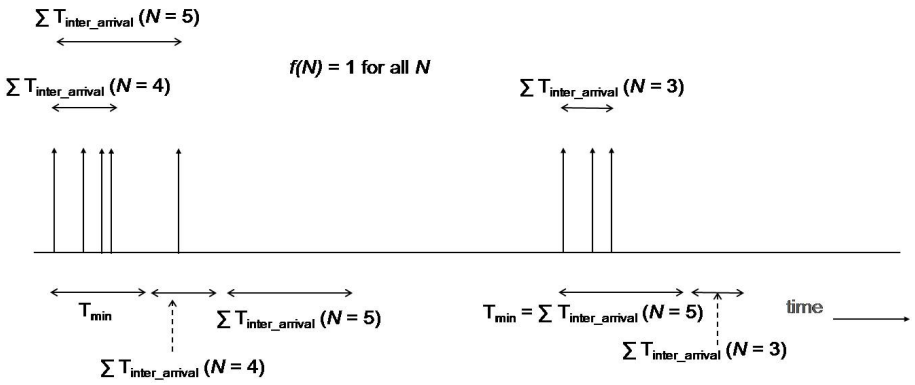


Figure 5.4: Time sequence of LSA grouping algorithm

The algorithm continues till there are no new LSA arrivals within the extended T_{th}. The step I in subsequent states provides a mean where the algorithm adaptively reduces the T_{th} value when there are no arrivals within an extended T_{th} period. Whereas step b and step II ensures the algorithm can adaptively increases T_{th} if more number of arrivals is registered during the extended T_{th} period. The algorithm is illustrated by an example in Fig. 5.4, where we have assumed that $f(N) = 1$ for all values of N.

After the LSU grouping is performed, the state space creation algorithm takes over. In the state space creation algorithm, each of the states or nodes maintains a number of relations with its upper tier. Each time a relation is registered for a particular state, an identifier is assigned and we refer to this identifier as relation Id. Relation Ids are created from the state space name and are shorted appropriately to store so that the overall search time to find the existence of a relation is minimized. Each of the transition to its upper tier state is associated with four variables namely, (a) the transition identification, (b) the frequency of the observed phenomenon during the learning phase, (c) the transition pass, that indicates the number of similar probabilistic transition a node can have to its higher tier, and (d) the associated probability for that particular transition. Every time a transition is observed, the algorithm searches among the existing relation Ids assigned to already registered relations for that particular state. If no match is found, a new relation is created; otherwise, the frequency counter for that relation is increased by one. The pass number indicates how deep the state is from its starting state for a particular group. Suppose for instance that the group (B, D, H) is detected from the LSU grouping. Initially, a state of (B, D, H) is created. Then at the next lower tier, three groups of (B, D), (D, H), (B, H) are created. Here, the pass number automatically becomes 1 for each of the transition from these three groups to the higher state (B, D, H). However, we can form a next tier of (B), (D) and (H) state, whose pass becomes 2 as they are two layers deeper into the SRLG tree for current observation. The important point is that for a particular relation a separate counter has to be maintained for each pass value. This particular counter counts how many times this relation has been observed. We can also observe that the value

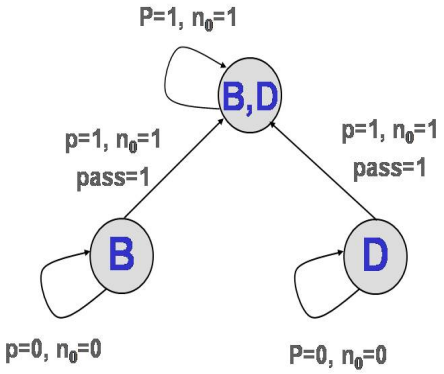


Figure 5.5: State space model for the first group (D, B) of Fig. 5.2

of the transition pass corresponds to the total number of transition from the concerned state to all possible next tier state.

The probability for a particular transition is defined by the following formula, where $pass(k)$ is the pass number associated with k^{th} pass for a particular relation and $\eta_{pass(k)}$ is the number of time the pass has been observed by the learning phase. The value of k can take any values from 1 to $n-1$.

$$P_{transition} = \frac{\eta_{pass(k)}/pass(k)}{\sum \eta_{pass(k)}/pass(k)} \tag{5.1}$$

We provide the following example to illustrate the above algorithm. Here we present the evolution of the state space model due to the arrival of the first two groups as shown in Fig. 5.2.

1. State space formation due to the arrival of the failure notification for the group (D, B) as shown in Fig. 5.5.

2. State space modification due to the arrival of the failure notification for the group (H, D, B) as shown in Fig. 5.5.

Initially, as depicted in Fig. 5.5, the transition pass is 1 for both the transition and the self transition probability for state (B,D) is 1. However, the self transition probability for states (B) and (D) remains zero. For the second LSU group (H, D, B), the value of the self transition probability changes to 1. The pass number for transitions from second tier ((B, D), etc.) to upper tier (B, D, H) is 1. However, a new transition with a new pass number 2 has to be created between first tier states ((B), (D), et.) and the second tier states (B, D), etc.). Therefore, the associated probability values for each of the transition are modified according to the equation (5.1). Note that, the transitions are going only in the upward direction and the total sum of the transition probabilities going out of a node always sum up to be 1, which verifies the validity and correctness of the algorithm.

Now to provide a general description of the above described algorithm,

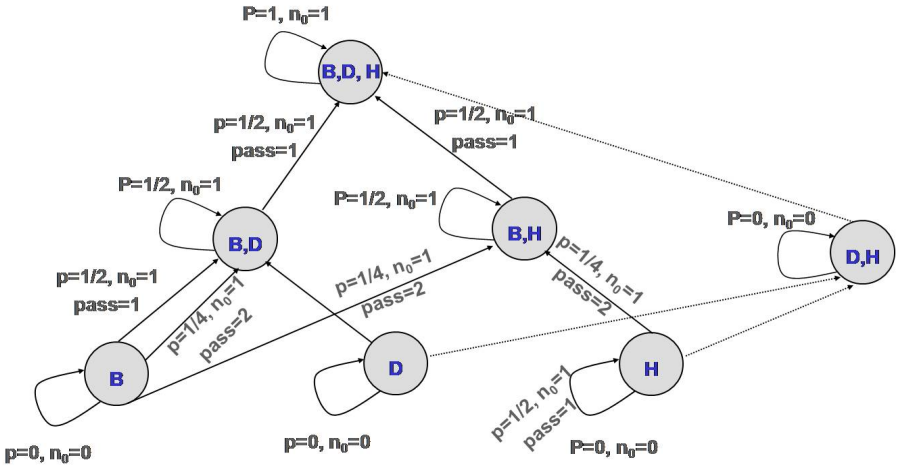


Figure 5.6: State space model for the second group (H, D, B) of Fig. 5.2

we suppose that there are n links denoted by l_1, l_2, ... , l_n where l_1, l_2, l_3, ... can represent link A, B, C, ... of our example scenario respectively. We denote the SRLG set as $S_{p,q}$ where p specifies the number of member link within that SRLG and the q specifies the SRLG id for individual SRLGs having p specified links chosen from the pool of n links as group member. Any member of the $S_{p,q}$ set can represent any state of p^{th} tier in our state diagram like (B, D) or (B, H) for our example state space model.

As there are a total of n links, therefore, the SRLG with the highest number of links that can be formed is denoted by $S_{n,1}$, where all the links are the member of that SRLG and the number 1 specifies the uniqueness of that particular SRLG. The next hierarchy of SRLG (in terms of number of links as member) can be represented as $S_{n-1,q}$ where q can vary from 1 to ${}^nC_{n-1}$, i.e., the all possible number of combinations with which $n-1$ links can be chosen from n links. The further hierarchy and their properties are specified in the table 5.1:

For our algorithm each of the $S_{n-r-1,q}$ maintains one state variable and a structure of relationships with the hierarchy member having hierarchy number $n-r$. The structure provides the relationship of $S_{n-r-1,q}$ with the immediate hierarchy i.e., $S_{n-r,x'}$ and to itself. For each of the relationship there can be multiple passes, depending upon the starting point of the SRLG group (e.g., $S_{n-r,x''}$) as shown in the examples (e.g., group (B, D, H) or group (B,D)). This is denoted by $pass(k)$ in our case. The number i specifies how many tiers below the concerned state (e.g.,$S_{n-r-1,x}$) exist from the starting SRLG (e.g., $S_{n-R,x''}$) in the state space model. Therefore, $i = (n-r-1) - (n-R) = R-r-1$ for a relationship between $S_{n-r-1,x}$ and $S_{n-r,x'}$ where the starting SRLG for this learning phase is $S_{n-R,x''}$. Each of the structure under the same pass keeps a counter ($\eta_{pass(k)}$) that is incremented after every occurrence of such relationship. Every relationship is also associated with a transition probability value that is updated after every iteration of learning

Hierarchy number	Symbol	Possible value of q	Member links per SRLG	Subset Relationship
0	$S_{n,1}$	1	n	
1	$S_{n-1,q}$	1, 2, ..., ${}^{n}C_{n-1}$	$n-1$	$S_{n-1,q} \in S_{n,1}$
...	...	...	...	...
R	$S_{n-R,x''}$	1, 2, ..., ${}^{n}C_{n-R}$	$n-R$	$S_{n-R,x''} \in S_{n-R,\hat{x}''}$ where $hatx''$ can take any of ${}^{R+1}C_{R}$ values for which SRLG $S_{n-R,\hat{x}''}$ contains all the members of the group $S_{n-R,x''}$
...	...	...	...	...
r	$S_{n-r,x'}$	1, 2, ..., ${}^{n}C_{n-r}$	$n-r$	Similar as Hierarchy R
$r+1$	$S_{n-r-1,x}$	1, 2, ..., ${}^{n}C_{n-r-1}$	$n-r-1$	Similar as Hierarchy R
...	...	...	...	...
$n-1$	$S_{1,q}$	1, 2, ..., ${}^{n}C_{n-1}$	1	Similar as Hierarchy R

Table 5.1: SRLG state hierarchy and their specification

phase according to equation 5.1.

This concludes the learning phase of our algorithm. In the learning phase, we create a state space representation of the SRLG map where the transition probabilities of various state transitions provide a statistical estimator of whether a group should be considered as SRLG or not. The learning can be offline, fed with the previous log file stored regarding network failures. The machine learning model can be programmed to filter out data from the log file to create LSU update sequence and hence, initial LSU groups which the algorithm can take as input data. In addition, the learning phase may continue functioning online while link failures occur. This dynamically modifies the structure of the SRLG state space and the associated transition probability values. Irrespective of the mode the learning phase is executed, the algorithm for SRLG detection needs the next phase of deciding and declaring SRLGs from the probability values provided by the learning phase.

5.3 Decision making phase

In the decision making phase, we use the state space based model and its transition probabilities to construct a basic control scheme regarding the existence of SRLGs in the entire network. The decision making follows the following recursion. Whenever a new LSU arrives mentioning a link failure, it triggers the following probability computation: The algorithm checks how far along the state space model can proceed. The algorithm detects all the transition connectivity between the state representing the failed link and all other states in the state space model. We need to keep in mind here that the transitions are only possible from a lower tier state to a higher tier state. The algorithm eventually calculates the overall probability of reaching all the possible upper tier states that is connected to the concerned state with a definite transition probability. All the probability values are then compared to a pre-defined threshold. The highest tier state that has probability higher than the threshold is then declared as the representative of SRLG. With our example state space model in Fig. 5.6 (using formula 5.1):

$p(BD|B) = p(B \rightarrow BD)$(for pass 1) $+p(B \rightarrow BD)$(for pass 2)
$= 1/2 + 1/4 = 0.75$

$p(BH|B) = 1/4 = 0.25$

$p(BDH|B) = p(B \rightarrow BD)p(BD \rightarrow BDH) + p(B \rightarrow BH)p(BH \rightarrow BDH)$
$= 0.75 * 0.5 + 1/4 * 1/2 = 0.5$

Using this example, we can observe that if the threshold probability is less than 0.75 and greater than 0.5 the SRLG include link B and D, whereas if the threshold probability is below 0.5 it chooses B, D and H as the mem-

bers of SRLG. If more than one state in the same tier has equal probability and are the highest state having probability above threshold, the algorithm defers decision and waits for additional inputs. In a more general term the transition probability calculation can be summarized through equation . Here denotes the overall probability of having an SRLG represented by $S_{n-R,x''}$ given a link failure notification related to l_α arrives where l_α belongs to $S_{n-R,x''}$ set.

$$pr(S_{n-R,x''}/l_\alpha) = \sum \prod_{j=1}^{n-R-2} pr\left(S_{j,i} \rightarrow S_{j+1,i'}\right) \tag{5.2}$$

where the summession terms run for $\prod_{m=1}^{n-R-2} \begin{pmatrix} m+2 \\ m \end{pmatrix}$ combination of transition for different i, i' where $S_{j,i} \subset S_{n-R,x''}$ and $S_{j+1,i'} \subset S_{n-R,x''}$ for all $j < n - R$. Here $pr\left(S_{j,i} \rightarrow S_{j+1,i'}\right)$ is calculated as follows:

$$pr\left(S_{j,i} \rightarrow S_{j+1,i'}\right) = \sum_k pr\left(S_{j,i} \rightarrow S_{j+1,i'} | pass(k)\right) \tag{5.3}$$

Equation 5.2 states that if we need to calculate the probability of one particular SRLG represented by $S_{n-R,x''}$ from a particular link l_α represented by the state $S_{1,\alpha}$ depends on all possible transition possible through the state space model from $S_{1,\alpha}$ to $S_{n-R,x''}$. The summation term in equation 5.3 considers all possible ways to connect state $S_{1,\alpha}$ to $S_{n-R,x''}$. It is a trivial exercise to show that there exist $\prod_{m=1}^{n-R-2} \begin{pmatrix} m+2 \\ m \end{pmatrix}$ ways to connect (track down from) state $S_{n-R,x''}$ to state $S_{1,\alpha}$. It is evident that the selection of the decision probability threshold is crucial for the overall performance of the proposed algorithm. We propose to run the same algorithm offline on the log file of LSU sequence of some known SRLG groups for varying values of probability threshold to find out the optimized value of the probability threshold. This concludes the description of the decision making phase. In the next subsection, we describe the use of the proposed SRLG identification mechanism to reduce recovery time.

5.4 Protection time reduction phase

In this phase, we use the outcome of the SRLG identification phase to reduce recovery time upon failure occurrence. As discussed earlier, identification of SRLG may trigger simultaneous routing and forwarding table updates for multiple links failure scenario with the arrival of single link failure notification. Modification of routing and forwarding table might take several milliseconds depending on the router entry. With any of the current link

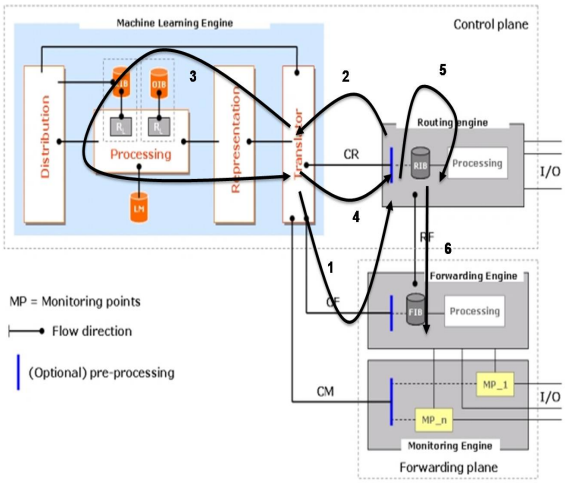


Figure 5.7: Router architecture along with SRLG decision flow diagram

state protocol (such as OSPF), common link failure resulting from an "SRLG failure" may trigger multiple LSDB update and subsequently RIB entries re-computation, one to address each of the link failure. Failing to prune the set of links involved by the SRLG failure at RIB entries re-computation leads to higher magnitude of packet losses compared to the situation where the set of links (associated to this SRLG failure) result in a single re-computation step. It has to be noticed that, independently of the actual RIB entries re-computation time, failing to take into account the set of links affected by the SRLG failure leads to traffic losses until all failed links have been pruned from the LSDB used as input for RIB entries re-computation.

We here outline the integration of the proposed technique into the ECODE architecture that enhances alegacy router with a machine learning (ML) component. As shown in Fig. 5.7, the ML component or engine interacts with the existing routers engines. Such router comprises four basic modules: the machine learning engine (MLE) and the monitoring engine (ME) in addition to the ordinary forwarding engine (FE) and routing engine (RE). The monitoring engine is not involved in the present application and thus not further considered in this study. The SRLG identification algorithm runs in the MLE. The typical data flow diagram is shown in Fig. 5.7 with numbered arrows.

The functionality of each of this data flows are described as follows:

1. MLE initiate request for link state updates from the RE.

2. RE starts sending update to MLE as they are available. RE continues to do so as the algorithm runs.

3. MLE learns and detects SRLG with Bayesian Network based state space model.

4. MLE sends SRLG prediction to RE.

5. RE re-calculates routing table for SRLG failure.

6. RE updates forwarding information base (FIB) entries for SRLG failure.

Therefore, the described architecture with the SRLG identification algorithm present in the machine learning engine helps router to reduce protection switching time.

5.5 Result and Discussion

Our experimental setup is based on an example network with predefined topology and SRLGs. We create an approximate analytical framework to represent the time sequence of LSUs for such a network. We use Matlab to generate these time sequences and then we run our algorithm on these time sequences to validate our model. The XORP implementation of the above mentioned machine learning algorithm will be dealt in the deliverables under Work Package 4. This deliverable mainly provides the proof of concept for the state space based Bayesian network model to be used for the use case of SRLG identification.

In this section we first describe how we formulate our input LSU time sequence for the experimentation of our algorithm.

To generate LSU time sequence used as input to our state space based model. As obtaining failure data set is difficult (due to the fact that failures are relatively sparse events), we created a model to generate input data set that closely mimics the real scenario. For this purpose, we assume that the link failure or any SRLG failure follows Weibull distribution [36]. The Weibull distribution is known for long for its special property to represent failure scenario [36]. The pdf of Weibull distribution is shown in equation 5.4:

$$f(x,\lambda,k) = \begin{cases} (k/\lambda)\left(\frac{x}{\lambda}\right)^{k-1} e^{-\left(\frac{x}{\lambda}\right)^{k}} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{5.4}$$

Where, λ is the scale parameter and k is the shape parameter. When we take k>1, the failure rate increases with time, which resemble the realistic network scenario. For our purpose we take k =1.5 and λ a very higher value to counter the fact that network failure events are rare in reality.

We have taken a network of 25 links among which links form multiple SRLGs of different link numbers. For our purpose, we have taken one SRLG each with 5 and 4 member links, two SRLGs with 3 and 2 member links and rest forming no SRLGs. We distinguish two cases depending on the

dominant delay factor in the origination and propagation of LSUs after the occurrence of correlated failure:

Case 1: Hello message and RouterDead detection Interval dominates. Suppose that 2 distinct LSUs for 2 different links, associated to the same SRLG, are issued by two different routers. The maximum delay they can suffer at the origin is due to the desynchronized Hello message intervals. As the Hello message intervals are in the order of seconds (typical values 1 to 10 sec.), at the higher inter arrival delay between two LSUs of the same SRLG failure, the Hello interval de-synchronization dominates the propagation or queuing delay. So for higher delay, we can assume a uniform distribution for the LSU inter-arrival time as desynchonization of issuing Hello message between two routers is completely random and independent

Case 2: Queuing, Transmission, and/or Propagation dominates. When the desynchronization due to Hello message is negligible, propagation and queuing delay dictates the cause for delay variance. Due to the accumulation of multiple mutually independent random queuing delays, we can assume without the loss of generality, that the interarrival time between LSUs are memory less and demonstrate exponential distribution. We have tested different fat tailed distribution that have power law decay to check if any significant changes occur in the final results. However, different interarrival time distribution shows minimal effect on the final result because failures are rare events that are usually separated by a large amount of time. Generally, inter-failure occurrence time is much larger than the inter arrival time between two LSUs for same SRLG failure to create any impact whatsoever.

With this specifically generated input data files, we run the LSU grouping algorithm as described in Section IIA. We use both grouping techniques described in Section IIA. However, very little difference in terms of final outcome was observed. Once the LSUs are grouped, we run the proposed state space based learning as well as the decision making algorithm to predict the existence of SRLG. We compare our results from the prior knowledge of SRLGs from our assumed network map and compute the amount of false positive and false negative the algorithm generates.

Fig. 5.8 provides the results for percentage of false positive and false negative as the number of failures per SRLG increases. This percentage value is the overall average among multiple random experiments performed after a certain number of failure data from each SRLGs are used to learn the state space. This experiment was carried over a set of disjoint SRLGs where the members of each SRLG are not part of any other SRLGs.

We next follow the same experiment for SRLGs with common members. Fig. 5.9 provides the results for multiple SRLGs having one common member. For this example we have included a common member between SRLG group having 5 and 4 members. Therefore, the previous SRLG with 4 mem-

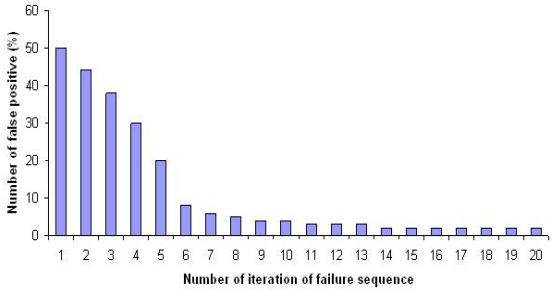


Figure 5.8: Percentage of false positive and negative with number of failure iteration (disjoint SRLGs)

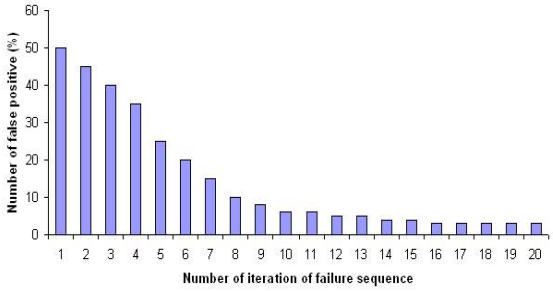


Figure 5.9: Percentage of false positive and false negative with number of failure iteration (SRLGs with one common node)

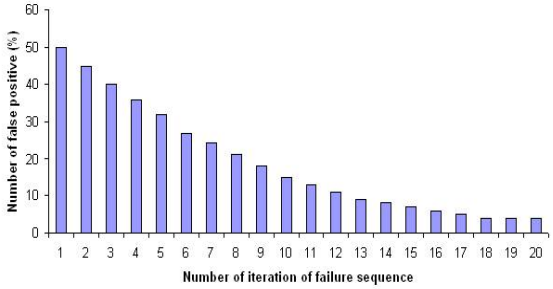


Figure 5.10: Percentage of false positive and false negative with number of failure iteration (SRLGs with two common nodes)

bers has now 5 member links. We can easily see that the performance degrades due to the interconnection between SRLGs. It is observed that the algorithm produces both false positive or false negative more often when the SRLG failures with common member link are happening. However, the percentage of false positive or negative decreases as the number of failure per SRLG increases. This is because the algorithm gathers more statistical input regarding different SRLG failures and hence learns to predict better. We further increase the number of common members. We add one more common link to the SRLG with 5 members so that the new SRLG with 6 and 5 links have two common member links. In Fig. 5.10 we observe further degradation due to two common members between SRLGs.

Finally, we investigate the gain of our SRLG detection method in terms of reducing the protection switching time and eventually reducing the amount of packet loss during failure. We assume the router update process to follow the quantum update procedure as described in [34] with update time = 100 μs, distribution time = 100 μs, swapping time = 1000 μs, prefixes per batch = 500 and total number of flows through the concerned router = 5000 having flow rate varying from 1 Kbps to 100 Mbps. For our example network with 20 failure cycles the total amount of data loss in the concerned router becomes 16 GBytes without the SRLG detection procedure. Whereas, SRLG detection and reduce it to 7.6 GBytes ($>$50 percent). This is significant for networks where failures are more frequent like wireless sensor networks or networks with old equipments.

Chapter 6

Profile-based accountability

In an attempt to achieve network fairness and avoid congestion, network operators try to use forms of subscriber accountability. Subscriber accountability deals with holding subscribers accountable for the traffic (or more specific: the network congestion) they introduce into the network. Although it has been on requirement lists of next generation Internet architectures for many years (e.g. [37]), the deployed accountability techniques are often static and very rigid.

In some Internet access plans, subscribers are given a maximum amount of bandwidth they can introduce during a fixed period (typically a month). Other Internet access plans advertise an unlimited amount of bandwidth but in fact use a fair use policy, where the introduced traffic may not diverge too much from the average generated traffic. There a number of problems with these accountability approaches. First, they only provide network fairness on a large time frame (e.g. a month). Subscribers have a certain amount of traffic that they can introduce into the network during this time frame and network operators can only assume that they introduce the traffic gradually. A subscriber who introduces a large peak in the generated traffic on the first day of the month and avoid peaks for the rest of the month cannot be held accountable as he does not violate the fair use policy. Second, current accountability approaches do not take the state of the network into account. Typically, a subscriber who introduces a large amount of traffic when the network is already heavy loaded can be considered more harmful than a subscriber who introduces a large amount of traffic when the network is far from being congested. The inclusion of this additional dimension is not possible in current accountability techniques.

The accountability technique discussed in this chapter, which we label profile-based accountability, tries to overcome these problems by characterizing the subscriber behaviour in real-time. By investigating subscriber parameters such as the generated traffic with respect to the actual network load an accurate view on the current subscriber behaviour can be determined, which allows for an automated and timely response to abuse by

subscribers (e.g. a subscriber who constantly generates more traffic than allowed). In this approach, we focus on locally obtained information about the subscriber; as such, the profile-based accountability technique can be deployed on each node and each node can perform actions to hold a subscriber accountable.

6.1 Formalization of the problem

The goal of the profile-based accountability technique is to characterize the momentary subscriber behaviour, and use this information to fairly allocate resources. In this context, fairness means that every subscriber receives the resources he paid for. Hence, there is a clear link with the subscription plan a subscriber has: subscribers that deviate too much from their expected behaviour, characterized by their subscription, in a way that it has a negative impact on the network can be punished by receiving less resources than they asked for. We call this expected behaviour the subscribed profile: although this subscribed profile is linked with the customer's subscription it is not fixed over time. The subscribed profile can vary depending on the overall behaviour of other subscribers having the same subscription and therefore the same subscribed profile.

Profile based accountability features three major tasks. First, the different classes of behaviour, which we call profiles, need to be determined during the profile learning stage. During this stage, the different profiles are characterized and defined. The subscriber information (e.g. the introduced traffic into the network) is monitored and used as input to find different classes of behaviour. The output of this stage is a list of subscriber profiles that are present in the network together with a mechanism to map the monitored monitor information to the subscriber profiles (e.g. through a decision tree). From a machine learning point of view, this phase can be constructed off-line, although incremental techniques can improve the scalability of the approach.

These generated subscriber profiles are used for the second task, the profile prediction stage. At runtime, the monitored subscriber data is continuously mapped to one of the **learned profiles**. This mapped profile provides a behavioural description of the subscriber at a given time. During this phase, the learned profile is continuously compared to the subscribed profile and its deviation from this subscribed profile is calculated. This deviation indicates how much the actual behaviour of the subscriber differs from the expected behaviour (i.e. the subscribed profile) in such a way that it has detrimental effects on the network condition.

The calculated deviation serves as input for the third and final task, the resource allocation stage. At this stage, actions are taken to ensure a fairness in resources between subscribers based on their current learned be-

haviour. Subscribers that do not deviate from their subscribed profile will be favoured in the allocation of resources. As such, subscribers that feature a large deviation will only receive sufficient resources when the network is not congested (i.e. there are more resources available that there is demand). These fairness measures can be implemented by changing the routing and forwarding plane of the routing engine (e.g. by implementing an adaptive Active Queue Management Weighted Fair Scheduling router).

The remainder of this chapter is structured as follows. Section 6.2 discusses the taken approach for the profile learning and prediction stage. The algorithmic options that are being considered are detailed in Sections 6.3 and 6.4. These algorithmic options have been implemented on the IBBT iLab, their details are discussed in Section 6.5. Initial results obtained from this implementation are discussed in Section 6.6, while future work is identified in Section 6.7.

6.2 Approach taken

The profile learning and prediction stages, described in Section 6.1, require the mapping of information about the subscribers behaviour, obtained through monitoring to a profile. For this, the profile needs first to be learned so that future mappings can occur by following some guidelines (e.g. stated through a decision tree or rules).

The problem can be modeled analytically as follows. The profile learning stage needs to determine the function map, where:

$$S_i(t) = map(M_i(t)) \tag{6.1}$$

Here, $S_i(t)$ denotes the subscriber behaviour (i.e. the learned profile) of a given subscriber i at a given time t. $M_i(t)$ denotes a vector of subscriber data that was monitored at specific time intervals of which t is the latest measurement. In other words, $M_i(t)$ contains the latest n measurements at time t:

$$M_i(t) = (Mon_i(t), Mon_i(t-1), Mon_i(t-2), ..., Mon_i(t-n+1)) \tag{6.2}$$

, where $Mon_{i,t}$ denotes the measurements taken for subscriber i at time t. Currently, $Mon_{i,t}$ is a vector with two elements where the first element denotes the traffic introduced by subscriber and the second element represents the overall network load, calculated by measuring the traffic introduced by all subscribers. Note that the second element will be the same for each subscriber i at the same time t.

Similar to these action profiles, the subscribed profiles need also to be learned and determined. These subscribed profiles define the overall behaviour of subscribers having the same subscription and are purely deter-

mined by monitoring the behaviour. As the overall, not one individual, behaviour evolves, the subscribed behaviour can evolve as well. However, this evolution will happen at a much slower pace (e.g. in terms of days) than that of the action profiles.

The determination of the mapping function map is part of the profile learning stage. Once determined, it can be used during the profile prediction stage to find the deviation of the mapped profile (output of map) with the subscriber i's subscribed profile SP_i at a time t:

$$D_i(t) = diff(SP_i, map(M_i(t)) \tag{6.3}$$

Here, $D(t) \in [0, 1]$ is a normalized value defining the deviation of subscriber i at a given time t with his subscribed profile SP_i.

As the subscriber behaviour can change over time (e.g. during the course of a day, a subscriber can access different applications each having different characteristics with respect to the traffic they introduce into the network) we split the determination of the mapping function map into two sub-tasks. First, we will map each $Mon_i(t)$ value to an action profile $A_i(t)$ describing the momentary behaviour of a subscriber during a given time frame:

$$A_i(t) = mapAction(Mon_i(t)) \tag{6.4}$$

When the time frame is chosen small enough, we can assume that the subscriber behaviour will not change during that time frame, making the action profile semi time independent during that time frame. By applying the $mapAction$ function to each $Mon_i(t)$ value we obtain a series of action profiles, each describing the subscriber behaviour at a given time slot. These action profiles can already contain information about the severity of the introduced traffic with respect to the overall network load. By taking this series of action profiles into consideration we can determine the deviation from the subscribed profile, through the $diffAction$ function:

$$D_i(t) = diffAction(SP_i, \vec{A_i}(t)) \tag{6.5}$$

where $\vec{A_i}(t)$ is a vector of the latest n action profiles:

$$\vec{A_i}(t) = (A_i(t), A_i(t-1), A_i(t-2), ..., A_i(t-n+1)) \tag{6.6}$$

6.3 Determination of action profiles

The $mapAction$ function needs to map real-time monitor values, containing behavioural information of subscribers, to an action profile. We have used a clustering and classification algorithm to implement this function. In the profile learning phase, the clustering and classification algorithms will be applied on an off-line generated training set. During this phase, the goal is to derive rules that allow to classify real-time monitor values into action profiles during the profile prediction phase.

6.3.1 K-means

K-means ([38]) is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. In other words centroids do not move any more. Finally, this algorithm aims at minimizing an objective function, in this case a squared error function. The objective function

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \parallel x_i^{(j)} - c_j \parallel^2$$

where $\parallel x_i^{(j)} - c_j \parallel^2$ is a chosen distance measure between a data point $x_i^{(j)}$ and the cluster center c_j, is an indicator of the distance of the n data points from their respective cluster centers.

The algorithm is composed of the following steps:

1. Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the K centroids.
4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centers. The k-means algorithm can be run multiple times to reduce this effect.

6.3.2 C4.5 Decision Tree Classifcation

The Simple K-Means algorithm is a great way to cluster data, however, in order to efficiently classify the real-time data during the profile prediction phase, a decision tree is needed. For this we will use the C4.5 Decision Tree Classification algorithm[39].

C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set $S = s_1, s_2, ...$ of already classified samples. Each sample $s_i = x_1, x_2, ...$ is a vector where $x_1, x_2, ...$ represent attributes or features of the sample. The training data is augmented with a vector $C = c_1, c_2, ...$ where $c_1, c_2, ...$ represent the class to which each sample belongs.

At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. Its criterion is the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurses on the smaller sublists.

This algorithm has a few base cases:

- All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
- None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
- Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

In pseudocode the algorithm is:

1. Check for base cases
2. For each attribute a
   Find the normalized information gain from splitting on a
3. Let a_{best} be the attribute with the highest normalized gain
4. Create a decision $node$ that splits on a_{best}
5. Recurse on the sublists obtained by splitting on a_{best}, and add those nodes a children of $node$

We used an open source Java implementation of the C4.5 algorithm in the Weka data mining tool[40], i.e. J48.

6.3.3 Employed attributes

To obtain the decision tree, generated by the C4.5 algorithm, as part of the $mapAction$ function we apply the C4.5 on a dataset that characterizes the behaviour of each subscriber. Currently, the following attributes are used to perform the clustering:

- The flow rate generated by each subscriber averaged over a timeframe
- The flow size generated by each subscriber (characterized by the cumulated size of packets) in the last timeframe
- The overall link load, characterized by the sum of all individually generated traffic. This link load is calculated by summing up all bytes and averaging over a timeframe
- The number of active connections
- The number of congested packets generated by each subscriber, as a metric for the congestion, reported during the last timeframe
- The total number of packets generated by each subscriber, reported during the last timeframe

6.4 Deviation of subscribed profile

Once the subscribed profile and action profiles are determined, based on the algorithmic options detailed in the previous section, determining the deviation is straightforward. This is illustrated in Figure 6.1, which shows a visualization of the action profile and subscribed profile clustering and the evolution of a single subscriber over time (characterized by the arrow).

As the subscribed profile will be determined at a slower pace, a wider variety of behaviours will be taken into account and the cluster will naturally be much larger than that of an action profile which only takes a small timeframe into account. Hence, the subscribed profile will define the boundary of what action profiles an individual subscriber may 'visit'. Subscribers are allowed to visit different action profiles, as long as they fall within those boundaries. This is illustrated in Figure 6.1, where the arrows represent the evolution to other action profiles. In this case, the evolution from action profile 4 to action profile 8 causes the subscriber to be out of his subscribed profile and hence out of profile. The deviation can be easily calculated using a distance function, calculated from the subscribed profile.

6.5 Implementation

6.5.1 Network model

The network is assumed to be a simple tree model. There is a single root node, and on each level there are a fixed number of branches. Links on the same level, have the same bandwidth. The servers are all located on the root node, while the clients are located on the leaf nodes. At any given moment, there is only 1 user active on each "client" leaf node.

This simplicity of this model still allows for enough realism, and is helpful in development and result processing. The models constraints could be removed without consequences for most of the code.

6.5.2 Traffic generation

As the individual subscriber behaviour is analyzed and made dependent on the type of applications a subscriber accesses, it is important to have realistic traffic traces. All tests were carried out on the iLab.t Virtual Wall with a distributed traffic generation tool, specifically design for the ECODE project, that is able to generate traffic by instructing real applications. To generate traffic using the Virtual wall, a software tool has been developed which can setup everything needed for the simulation, using a description of the scenario to emulate. There are many different tasks which this tool must solve, and so the code is split into parts accordingly. Each task is described in a subsection below.

6.5.2.1 Scenario description input

The scenario to emulate is described in a simple XML document. An example is given in Figure 6.3. A GUI frontend for generating this XML file has been created. It is shown in Figure 6.2

The XML description is processed by a ruby script, which creates all files needed to run the experiment on the virtual wall.

6.5.2.2 Converting a requested emulated topology into a wall topology

The scenario contains a description of the topology to emulate, using physical nodes on the virtual wall. Due to limitations of the virtual wall, this physical topology can be different than the emulated topology. The most important limitations are a maximum number of physical interfaces in each

wall node, and a maximum bandwidth of 1 GB over each link. A wall node can also only simulate a certain number of clients (this limit needs to be determined experimentally). Once a working topology has been determined, an NS description of this topology is written. This file is used by the virtual wall managment software to allocate the nodes for the simulation.

6.5.2.3 Assigning IPs and setting up routing.

The simulated network requires IPs and networks to be assigned in the simulated network. Currently, a minimum of networks, as needed by the wall topology, is used. It is also possible to use networks corresponding to the emulated topology, but this creates bigger routing tables. A ruby script determines and assigns networks. It then creates bash scripts that, in the pre-simulation setup of the simulation, are called in order to set up routing. These scripts use the Linux "ip" command. Because multiple emulated nodes can be aggrated on a single physical node, multiple IP addresses can be assigned to the same node. This creates a problem, as client applications, used for simulating clients cannot always use a user-specifiable IP. In order to force these applications to use a certain client IP a trick has to be used. Our trick is to assign multiple server addresses, and specify the routing on the physical node so that each server address corresponds with a client address. This way, all client applications CAN specify their source IP, by selecting a matching server IP.

6.5.2.4 Setting up bandwidth limitations

Bandwidth limitations are set up using the Linux traffic shaper "TC". TC supports various sorts of complex egress queues, and filters that determine which packet goes to which queue. By shaping both directions of traffic a link (thus shaping the egress on all connected nodes), the bandwidth limitation on that link is emulated.

TC uses a cryptic and confusing syntax. To minimize problems implementing the bandwidth limitations, "TCNG" (TC Next Generation) is used. This is a sort of compiler, which translates a higher level, much "friendlier" language into TC rules. The TCNG files are generated by a ruby script.

6.5.2.5 Simulating servers

Servers are emulated using actual server software such as apache and xstreamer. For youtube emulation, the apache server uses a plugin, to limit the outgoing rate per connection to 1Mbit/s, which is the same as youtube.

6.5.2.6 Client simulation

Clients are simulated using a ruby script which calls other scripts to simulate behavior. The ruby script takes an XML file specifying which behavior occurs between which hours as input. At the correct times, scripts for simulating behavior are started and stopped.

6.5.2.7 Network monitoring

The network is monitored using a custom tool, written in C++. This tool uses libpcap for monitoring network packets. Several monitoring intervals can be specified, and for each such interval a logfile is specified. When the requested time has passed, statistics about the data received during that time interval are written to the corresponding log file. Multiple interfaces can be monitored at once.

6.5.2.8 Starting and stopping simulations, and moving results

When the simulation starts, bash scripts are started on each node, which configure the simulation, and then start it. The configuration step includes installing the needed software, configuring it if needed, setting up ip addresses, configuring routing, and bandwidth shaping. When the simulation is complete, the scripts move the results from a temporary location, to the appropriate final location.

6.6 Experimental results

We have emulated 2 initial scenarios, and using the monitoring data we obtained, we examined the clustering results. The investigated network topology consisted of a server node, with 5 links of 25 Mbit/s to the routers, and from each router links of 5 Mbit/s to the clients. These scenarios feature 3 type of subscriber behaviours:

- Day web subscribers: Surf the web all day, but not in the evening. 75% of subscribers.
- Evening youtube subscribers: Watch youtube all evening. 10% of subscribers.
- Evening Downloaders: Download large files in the evening. 15% of subscribers.

In total, there are 200 of these subscribers in scenario A. The bandwidth usage is shown in Figure 6.4.

Figure 6.5 shows the dataset for this scenario after classification using the J48 algorithm in Weka. Based on the total bandwidth and the user's share of this total bandwidth, the algorithm tries to predict the corresponding action profile through classification. A square point indicates that the point was wrongly classified, while crosses mark a correct classification. In this case the J48 algorithm classified 99.3% of the measurements correctly.

Figure 6.6 shows the corresponding decision tree for this classification. As can be seen, a simple decision tree can be constructed that allows for a straightforward implementation of the $mapAction$ function. This decision tree can then be used in the profile prediction phase, which is part of future work, as discussed in the next section.

6.7 Future work

The obtained results discussed in Section 6.6 provide an initial indication of the accuracy of the classification algorithms. Future work will be carried out in WP4 and is divided into three main areas: First, the evaluation will be extended to investigate other scenarios and other network configurations. This will allow to obtain more realistic results of a wide bouquet of configurations. Second, as the evaluation is currently only limited to the accuracy of the profile learning stage, the deviation of the subscribed profile will be taken into account as well, following the described algorithmic options in Section 6.4. Third, the effect on the overall network performance will be investigated as well during the third and final phase: the resource allocation phase.

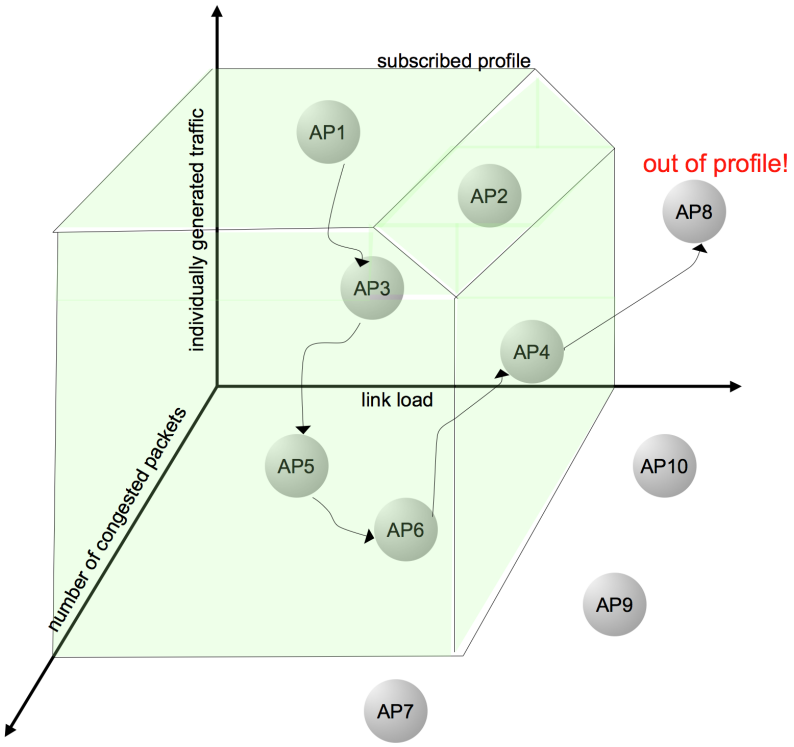


Figure 6.1: Determination of the deviation of a profile

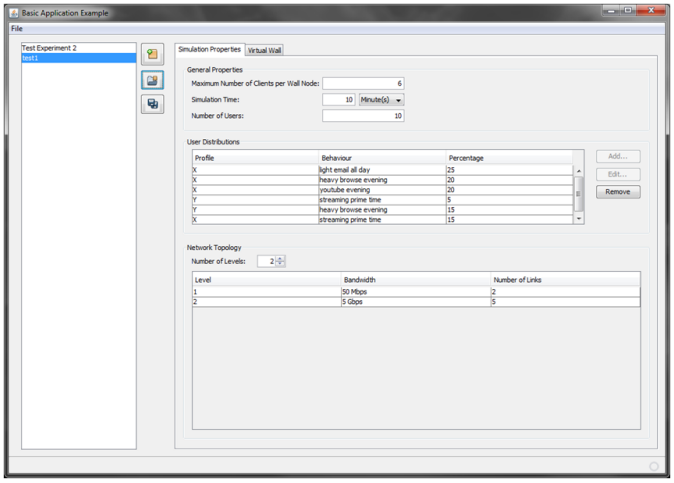


Figure 6.2: Example of Scenario Description GUI

```
<?xml version="1.0" encoding='UTF-8'?>
<experiment name="profile-config-1.1a">
  <wall_aggregation_limits>
    <clients>100</clients>
  </wall_aggregation_limits>

  <simulationtime>120 minutes</simulationtime>
  <usercount>50</usercount>

  <userdistribution>
    <usergroup profile="1" behaviour="web" percent="50"/>
    <usergroup profile="1" behaviour="youtube" percent="50"/>
  </userdistribution>

  <topology>
    <level nr="1" bw="25 Mbps" links="5" />
    <level nr="2" bw="5 Mbps" links="10" />
  </topology>

  <monitoring>
    <simulation_interval>
      30 seconds
    </simulation_interval>
    <emulated_interval>
      1 hour
    </emulated_interval>
  </monitoring>
</experiment>
```

Figure 6.3: Example of XML Scenario Description

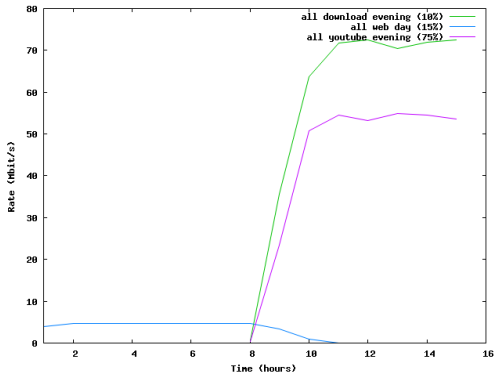


Figure 6.4: Bandwidth usage of scenario A

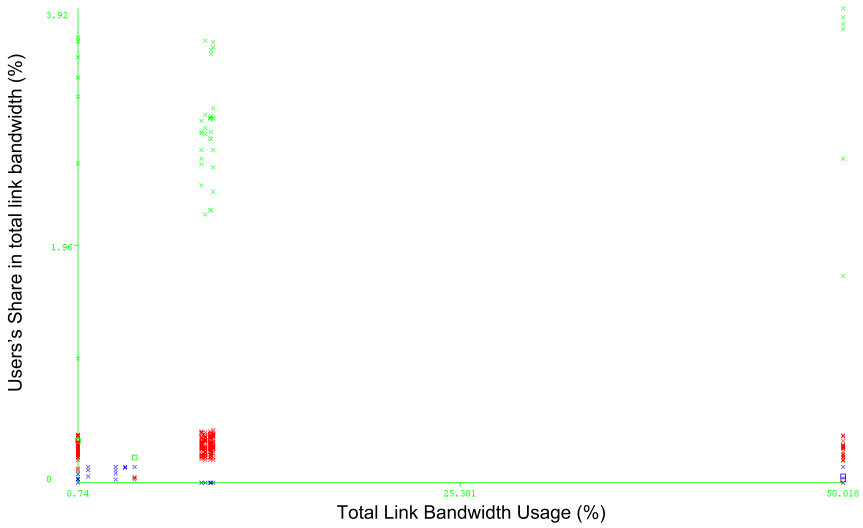


Figure 6.5: Classification done by means of the J48 algorithm

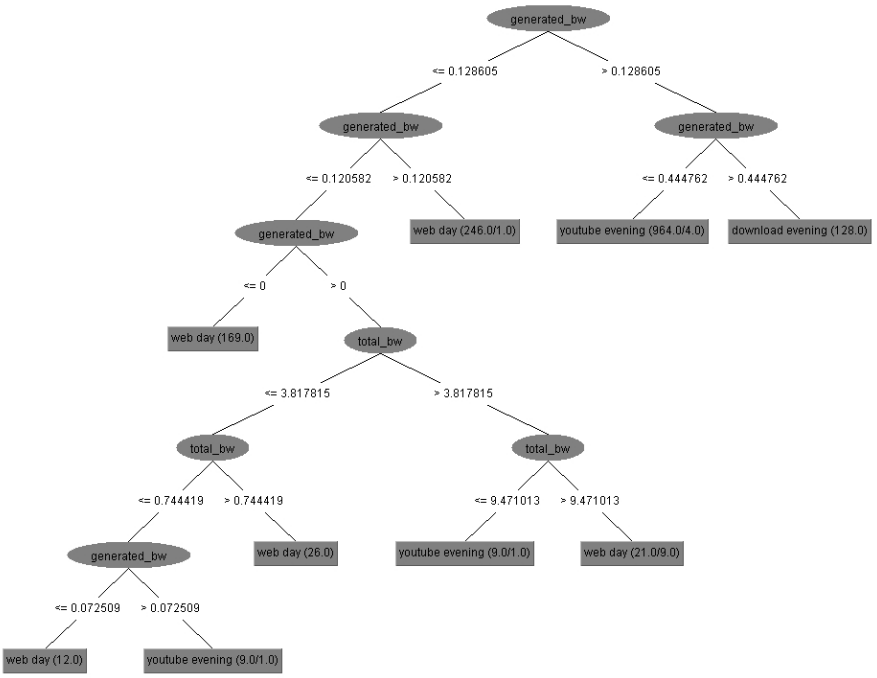


Figure 6.6: Decision Tree generated by means of the J48 algorithm

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 (chapters 4 and 5), and profile-based accountability (chapter 6).

In each chapter, the problem addressed by the use case is formalized and the relevant machine learning (ML) techniques are used to provide appropriate solutions. The proposed ML-based algorithms have been evaluated by simulations, which can be seen as first high-level prototypes. For use-case b1, first implementations in the XORP environment have already been developed and locally tested.

The main contributions can be summarized as follows:

- We explain how IDIPS (our path ranking service) has been implemented within XORP, an extensible open source routing platform. Our implementation is based on three modules: Ranking, Prediction, and Measurement. The Ranking module is in charge of dispatching requests from Clients to other modules. The Measurement module is used to measure path performance metrics, while the Prediction module uses a Machine Learning technique (Normalized Least Squares) to predict path performance metrics, so that we reduce the amount of traffic injected in the network. In addition, we provide a first description on how we plan to evaluate IDIPS on the iLab platform.
- Internet Coordinate Systems (ICS) are promising techniques to predict unknown network distances (typically delays) from a limited number of measurements. Most ICS algorithms are based on metric space embedding and suffer from the inability to represent distance asymmetries and Triangle Inequality Violations (TIVs). To overcome these drawbacks, we formulate the problem of network distance prediction as a machine-learning problem, namely guessing the missing elements of a

distance matrix, and solve it by matrix factorization. A distinct feature of our approach [1], called Decentralized Matrix Factorization (DMF), is that it is fully decentralized. The factorization of the incomplete distance matrix is collaboratively and iteratively done at all nodes with each node retrieving only a small number of distance measurements. There are no special nodes such as landmarks nor a central node where the distance measurements are collected and stored. We compare DMF with two popular ICS algorithms: Vivaldi and IDES. The former is based on metric space embedding, while the latter is also based on matrix factorization but uses landmarks. Experimental results show that DMF achieves competitive accuracy with the double advantage of having no landmarks and of being able to represent distance asymmetries and TIVs. The knowledge of estimated delays between nodes can also be useful to select better paths for real-time applications. We had proposed some methods that rely on the nodes running an ICS to detect routing shortcuts in networks. We have now evaluated more precisely the quality of the results provided by these methods. Finally we explain a first implementation of an ICS in the XORP environment and discuss some relevant aspects of the current implementation. We also analyze the memory and performance cost of our module. Finally we explain how to improve and evolve this module for better integration and tight interaction with the ECODE architecture.

- An accurate understanding or characterization of network traffic dynamics can improve network efficiency. Long-term traffic characterization enables network operators to dimension their networks accordingly; short-term network traffic trends enable dynamic rerouting techniques to efficiently use alternative paths in a network. We use state-of-the-art network traffic models to model network traffic in a very short timeframe (sub-second) as observed by an IP router during the process of updating routing entries after failure detection. We study the highly sensitive interaction of network traffic with the IP router update process in detail and present a mathematical model to characterize this process. The goal of this model is to optimize the process of updating routing entries in an IP router with minimal packet loss. For this, two optimization heuristics are formulated and are evaluated together with state-of-the-art alternatives in a realistic simulation environment. The trade-off of several parameters in the model is characterized, and we show that a gain in performance (decrease in packet loss) can be achieved.

- In the OSPF data mining use case for shared risk link groups (SRLG) identification, we use machine learning technique at the routers to study the link state protocol (e.g., OSPF) data to predict the existence of SRLG in the network. In particular, we use the correlation between different link state updates (LSUs) issued by different network nodes (routers) upon failure. The concerned network router then runs a novel Bayesian network-based statistical learning process to construct

a state space model for representing and learning about the possible existence of SRLGs. The decision of this online learning is transferred to the routing information base (RIB) so that it can accordingly modify the routing table for the entire SRLG upon failure detection of one of the candidate link of that particular SRLG and hence reduce the protection switching time.

- With respect to the profile based accountability use case, this document presents an analytical model of the problem. This model identifies the different functions that need to be implemented through machine learning solutions. Different algorithmic options for implementing these functions are described and the way they can be applied to the specific problem is detailed. Furthermore, we discuss the experimental set-up that has been built and that includes a traffic generator, specifically built in the context of the ECODE project, that allows generating traffic traces on the iLab.t Virtual Wall based on the behaviour of actual applications. The results of initial tests that apply the machine learning algorithms on the obtained traffic traces are presented.

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.
- Prototype codes in XORP will be developed for use cases b2 and b3, and improved for use case b1. All of them will have to be fully tested over the iLab.t Virtual Wall tested and possibly PlanetLab. Then, in WP4 they will be integrated into the common ECODE architecture defined in WP2. When these real implementations are integrated, a second stage of tests and validations will be carried out to check the integration and refine the ECODE architecture, if needed.

Bibliography

[1] Y. Liao, P. Geurts, and Leduc G. Network distance prediction based on decentralized matrix factorization. In IFIP Networking, LNCS, Chennai, India, May 2010.

[2] G. Leduc. Design and implementation of technical objective 2. Deliverable FP7-ICT-2007-2-1.6-223936-D3.4, ECODE, October 2009.

[3] M. Handley, O. Hodson, and E. Kholer. XORP goals and architecture. In Proc. ACM SIGCOMM Hot Topics in Networking (HotNets), October 2002.

[4] 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.

[5] L. Mathy. Cognitive engine experimental low-level design. Deliverable D2.2, ECODE, May 2010.

[6] David Andersen Hari, David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. pages 131–145, 2001.

[7] Azureus Bittorrent. `http://azureus.sourceforge.net`.

[8] Benoit Donnet, Bamba Gueye, and Mohamed Ali Kaafar. A survey on network coordinates systems, design, and security. To appear in IEEE Communication Surveys and Tutorial, Dec 2010.

[9] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In Proc. IEEE INFOCOM, New York, NY, USA, June 2002.

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

[11] H. Zheng, E. K. Lua, M. Pias, and T. Griffin. Internet Routing Policies and Round-Trip-Times. In Proc. the PAM Conference, Boston, MA, USA, April 2005.

[12] S. Lee, Z. Zhang, S. Sahu, and D. Saha. On suitability of euclidean embedding of internet hosts. SIGMETRICS, 34(1):157–168, 2006.

[13] 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, oct 2007.

[14] Suman Banerjee, Timothy G. Griffin, and Marcelo Pias. The interdomain connectivity of PlanetLab nodes. In Proc. of the Passive and Active Measurement Workshop – PAM'2004, Lecture Notes in Computer Science (LNCS) 3015, Antibes Juan-les-Pins, France, April 2004.

[15] Yun Mao, Lawrence Saul, and Jonathan M. Smith. Ides: An internet distance estimation service for large networks. IEEE Journal On Selected Areas in Communications (JSAC), Special Issue on Sampling the Internet, Techniques and Applications, 24(12):2273–2284, Dec 2006.

[16] Yang Chen, Xiao Wang, Xiaoxiao Song, Eng Keong Lua, Cong Shi, Xiaohan Zhao, Beixing Deng, and Xing Li. Phoenix: Towards an accurate, practical and decentralized network coordinate system. In Proc. IFIP Networking Conference, Aachen, Germany, May 2009.

[17] Gene H. Golub and Charles F. Van Loan. Matrix computations (3rd ed.). Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[18] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In NIPS, pages 556–562. MIT Press, 2001.

[19] A simulator for peer-to-peer protocols. `http://www.pdos.lcs.mit.edu/p2psim/index.html`.

[20] B. Wong, A. Slivkins, and E. Sirer. Meridian: A lightweight network location service without virtual coordinates. In Proc. the ACM SIGCOMM, aug 2005.

[21] Liying Tang and Mark Crovella. Geometric exploration of the landmark selection problem. In Proc. of the Passive and Active Measurement Workshop – PAM'2004, Lecture Notes in Computer Science (LNCS) 3015, Antibes Juan-les-Pins, France, April 2004.

[22] Barry Raveendran Greene and Philip Smith. Cisco ISP Essentials. Cisco Press, 2002.

[23] Bamba Gueye and Guy Leduc. Resolving the noxious effect of churn on internet coordinate systems. In Lectures Notes in Computer Science 5918, pages 162–173, Zurich, Switzerland, December 2009.

[24] 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.

[25] Murad S. Taqqu, Walter Willinger, and Robert Sherman. Proof of a fundamental result in self-similar traffic modeling. SIGCOMM Comput. Commun. Rev., 27(2):5–23, 1997.

[26] Will E. Leland, Walter Willinger, Murad S. Taqqu, and Daniel V. Wilson. On the self-similar nature of ethernet traffic. SIGCOMM Comput. Commun. Rev., 25(1):202–213, 1995.

[27] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. Networking, IEEE/ACM Transactions on, 3(3):226–244, Jun 1995.

[28] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. IEEE/ACM Trans. Netw., 5(6):835–846, 1997.

[29] N. Sadek, A. Khotanzad, and T. Chen. Atm dynamic bandwidth allocation using f-arima prediction model. In Computer Communications and Networks, 2003. ICCCN 2003. Proceedings. The 12th International Conference on, pages 359–363, Oct. 2003.

[30] 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.

[31] R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[32] Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for r. Monash Econometrics Working Papers 6/07, Monash University, Department of Econometrics and Business Statistics, June 2007.

[33] Hasslett and Ratery. Maximum likelihood estimation of the parameters of a fractionally differenced arima(p,d,q) model. Applied Statistics, 1989.

[34] W. Tavernier, D. Papadimitriou, D. Colle, M. Pickavet, and Demeester P. Optimizing the ip router update process with traffic-driven updates. In Design of Reliable Communication Networks (DRCN), pages 25–28, Washington D.C., USA, October 2009.

[35] B. Zhou, D. He, and Z. Sun. Network traffic modeling and prediction with arima/garch, 2005.

[36] A. Papoulis and S.U. Pillai. Probability, Random Variables, and Stochastic Processes - 4th edition. McGraw-Hill, 2001.

[37] R. Braden, D. Clark, S. Shenker, and J. Wrowclawski. Developing a next-generation Internet architecture. In White paper, DARPA, July 2000.

[38] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, 1967.

[39] J. Quinlan. C4.5: Programs fir Machine Learning. Morgan Kaufmann, 1992.

[40] M. Hall, F. Eibe, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. SIGKDD Explorations, 11(1), 2009.