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

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D3.7
"Experimental evaluation of techniques to improve the scalability and quality properties of the routing system"

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-D3.7 Title: Experimental evaluation of techniques to improve the scalability and quality properties of the routing system Responsible: D.Papadimitriou (ALB) Editor(s): D.Papadimitriou (ALB) Contributor(s): D.Papadimitirou (ALB), Mickael Hoerdt (UCL), Benoit Donnet (UCL) Dissemination level: Public (PU) **Date of preparation:** *October 31st, 2010* **Version: 1.1**

D3.7 - Experimental evaluation of techniques to improve the scalability and quality properties of the routing system

Executive Summary

The aim of the ECODE experimental research project is to introduce a new Internet architectural component realized by means of a machine learning component. This deliverable, belonging to WP3 (Cognitive network and system experimentation), is part of a series of 3 deliverables – one per technical objectives (TO) as defined in the ECODE project technical annex. It aims at describing the feasibility, benefits and applicability of introducing machine learning techniques to improve the scalability and the quality of the Internet routing system that is based on the Border Gateway Protocol (BGP). More specifically, this deliverable addresses the experimentation performed in the framework of TO3 related to this use case following the design and implementation documented in Deliverable D3.6.

We are interested in finding appropriate machine learning technique for path exploration sequences detection and identification so as to mitigate its effects. By mitigation, we mean here enforcing the suppression of subsequent sequences of BGP routing update messages that are detrimental to the routing system convergence (delays, and transient instabilities). The experimented the machine learning techniques for this use case, focus on the techniques: decision trees, the hidden Markov model (HMM) and Conditional Random Field (CRF). A decision tree is a decision support tools using a tree-like graph of decisions. The HMM model is a statistical model in which the system being modelled is an embedded stochastic process with an underlying Markov process that is not observable (it is hidden) but can only be observed through another set of stochastic processes that produce the sequence of observations. Conditional random fields (CRFs) are probabilistic framework for labeling and segmenting sequential data by defining a conditional distribution (over label sequences given a particular observation sequence), with an associated undirected graphical model. The primary advantage of CRFs over HMM is their conditional nature, resulting in the relaxation of the independence assumptions required by HMMs in order to ensure tractable inference.

Performance evaluation results of the proposed machine learning techniques are detailed. At this stage, evaluation of the proposed machine learning has been performed by means of execution in combination with real BGP engine running on the XORP routing platform. These experiments have been conducted for realistic Autonomous System (AS) topologies, following several properties of today's Internet topology such as hierarchical structure, power-law degree distribution, strong clustering as well as a constant average path length. Additional experiments are ongoing to refine these results and future work will consist in further improving the learning model specified in this document. We further demonstrate the feasibility of running these machine learning algorithms together with a real BGP router.

List of Authors

Affiliation	Author
ALB	Dimitri Papadimitriou
UCL	Mickael Hoerdt
UCL	Benoit Donnet

List of Acronyms

AS	Autonomous System
BGP	Border Gateway Protocol
CRF	Conditional Random Field
CPU	Central Processing Unit
eBGP	External Border Gateway Protocol
FN	False Negative
FP	False Positive
HMM	Hidden Markov Model
eBGP	External Border Gateway Protocol
iBGP	Internal Border Gateway Protocol
IP	Internet Protocol
ISP	Internet Service Provider
RIB	Routing Information Base
TN	True Negative
TP	True Positive
T1	Tier-1
T2	Tier-2
T3	Tier-3
XORP	eXtensible Open Routing Platform

Table of Contents

1. Introduction

The Border Gateway Protocol (BGP) [4] is a path vector routing protocol used to maintain connectivity among the collection of independently administered Autonomous Systems (ASes) in the Internet. Thanks to BGP, Internet Service Providers (ISPs) are able to connect to each other and end-users can connect to more than one ISP. The primary function of a BGP router is to exchange network reachability information with other BGP routers. Among others, this information includes a list of ASes that the reachability information goes across. This list allows routers to prune routing loops while some policies at the AS level can be enforced.

Path exploration is a phenomenon occurring during the convergence process of path vector routing protocols such as BGP. Path exploration slows down the convergence process has shown significant impact on the performance of BGP [5]. In response to path failures or routing policy changes, BGP routers may try several transient paths before selecting a new best path or declaring a destination as unreachable. This phenomenon, called Path exploration, can occur due to topology changes, hardware failure, and new routers or BGP session establishment. Also, session failures due to equipment failures, maintenance or due to congestion on the physical links can lead to routing changes. Finally, it can also be due to changes in routing policies after reconfiguration of preferences or route filters, or when policies in different ASes are conflicting. The final outcome of this convergence will be either a withdraw message or an update message for a given prefix in the network and might potentially take a long time to be produced. In case the final outcome of a convergence is a withdraw message, a long series of re-announcements can take a long time to finally end with the removal of the prefix reachability from the BGP routing table.

In this context, we are interested in designing and experimenting appropriate machine learning technique for path exploration sequences detection and identification so as to mitigate their effects. By mitigation, we mean here enforcing the suppression of subsequent sequences of BGP routing update messages that are detrimental to the routing system convergence (delays, and transient instabilities). The underlying idea is that by using (semi-)supervised machine learning techniques, it would be possible to predict the outcome of a BGP path exploration sequence before it happens in order to save time in the BGP decision process concerning a given prefix.

1.1 Structure of the Document

In this document, we first give in Section 2 an overview of the machine learning techniques implemented to detect and identify Path exploration sequences and mitigate their effects by predicting their occurrence and anticipate the outcome of the BGP decision process. These techniques are based on the C4.5 decision trees, Hidden Markov Model (HMM), and Conditional Random Fields (CRF), the discriminative analog to HMM. Then, in Section 3, we provide an overview of the experimental model developed using the XORP routing platform to conduct our experiments. In Section 4, we describe the experimental setup and the methodology followed to execute our experiments. We present in Section 5 the evaluation of the memory size required to maintain BGP routing information to train our learning algorithm, the performance of our learning algorithms (in terms of metrics such the number of path exploration events detected, the detection time, the number of undetected path exploration events), the gain (in convergence time) from applying the learned model driving our mitigation technique where the route selection process is anticipated upon path exploration event detection.

2. Machine Learning

The goal is to experiment the feasibility and the viability of integrating semi-supervised machine learning techniques within a BGP routing engine. For this purpose, the following machine learning techniques are experimented: C4.5 decision trees, Hidden Markov Model and Conditional Random Field (also referred to as Markov random field).

2.1 Decision Trees

The proposed machine learning technique is based on the C4.5 automatic decision tree building process. To build the decision tree, one needs to inject a set of training samples produced from BGP sequences corresponding to an event that already happened in the network. This step is called the learning phase. We show in this work that the learning phase is an operation that can only be performed off-line for two reasons: firstly over a long period of time there are too few samples for a given prefix (i.e. relatively stable paths), secondly producing samples is a memory intensive operation because it needs to remember all the updates messages related to a given event. A specific sample is produced this way: on a sequence of updates messages received corresponding to a known event, a measurement vector that depends on the n previous BGP messages received is computed. To try to build a decision tree that would be able to predict the end of a sequence depending on this vector, this measurement vector is then associated with the end of the event which will be either the withdrawal of the prefix or the re-announcement of it and then included in the set of training sample. Once the decision tree has been built, it is implemented inside the BGP router as series of conditional if-then instructions on the measurements. This means that the decision to be taken still depends on a measurement vector computed on a limited set of BGP updates messages that arrived at the router.

2.1.1 Tree-based Learning

The problem we are facing is the following: given a sequence of BGP UPDATE messages, we want to predict in advance the outcome. A sequence can either end with a new best path or with a withdrawal. This is thus a binary classification problem. Having information about the outcome of the path exploration process will then allow one to improve the global BGP convergence speed.

The developed model is the entity in charge of predicting the outcome of a path exploration process. This model needs to fulfill two major constraints: efficiency and compliance with real-time constraints. Indeed, routers need to take quick decisions about whether aborting or continue the path exploration process. Also, routers may have limited CPU and memory resources.

Several assumptions must be considered to define such a model precisely:

- A sequence of UPDATE messages related to a single prefix can be isolated.
- There exists a path exploration detector able to detect the beginning and the end of a path exploration process, based on BGP UPDATE messages.

The selected learning model is a tree-based learning. It matches our constraints as the complexity of the tree traversal is logarithmic to the number of nodes in the tree and the size of the tree depends on the number of rules contained in the tree. We chose to base our learning on received BGP UPDATE messages, which means the use of supervised technique to build the tree.

Decision trees are decision support tools using a tree-like graph of decisions [12]. Trees are built iteratively. The best-known decision tree builder algorithms are ID3 [2] and C4.5 [3]. The next section focuses on the C4.5 algorithm.

2.1.2 C4.5 Algorithm

The ID3 algorithm, introduced by Quinlan [2], works as follows: for each node, the best decision attribute is chosen to maximize the expected reduction in entropy after sorting on this attribute, also called the Gain. The C4.5 algorithm [3], also introduced by Quinlan, is an improvement of the ID3 algorithm using the information entropy. Compared to ID3, C4.5 introduces two new goals: the avoidance of overfitting to the data and the incorporation of continuous-valued attributes.

- Overfitting avoidance: C4.5 uses a validation set distinct from the training set used to detect overfitting. When increasing the tree depth, if the classification performance increases on the training set and decreases on the validation set, the hypothesis being built is overfitting the training data set. The growth is stopped when the data split is not statistically significant. Post-pruning is applied on the whole tree to remove sub-trees whose removals improve validation set accuracy. The post-pruning idea is to infer the tree as well as possible, convert the tree to an equivalent set of rules, and then prune each rule by removing any preconditions that result in improving its estimated accuracy. And finally, sort final rules by their estimated accuracy and consider them in this sequence when classifying.
- Continuous-valued attributes: C4.5 selects candidate thresholds midway between instances with distinct classifications.

2.2 Hidden Markov Model

HMM is a statistical model in which the system being modeled is assumed to be embedded in a stochastic Markov process that is not observable (it is hidden) but can only be observed through another set of stochastic processes that produce the sequence of observations (the observation is a probabilistic function of the state). Such model represents stochastic sequences as Markov chains where the states are not directly observed but are associated with a probability density function. That is, the sequence of state can be observed only through the stochastic processes defined into each state, i.e., the parameters of the probability density function of each state must be known before being able to associate a sequence of states $Q = q_1,q_2,..,q_T$ to a sequence of observations $O = O_1,O_2,..,O_T$, where T is the number of observations in the sequence.

The HMM model λ with N hidden states and M distinct observation symbols per state that correspond to the physical output of the system being modeled, is defined by three probability distributions: the (state) transition probability distribution (A), the observation probability distribution (B), and the initial state distribution (Π); and denoted by $\lambda = (A,B,\Pi)$.

2.2.1 Hidden Routing System States as HMM

HMM models the probability of occurrence of observations x (AS_Path sequences that populate the RIB_In) and routing system state y; that is, it is a representation of the joint distribution P(x,y). Our problem consists in classifying observed AS-Path sequences as received by the BGP selection process with the purpose to accelerate the detection of path exploration sequences and to subsequent select (or generate) the adequate AS-Path. Each state of the BGP routing system is modeled as HMM state. Five hidden states are defined for the HMM $\lambda = (A,B,\Pi)$. The output of the HMM triggers the selection of an AS_Path that populates the Loc-RIB. Note that the model can be applied per destination prefixes or set of (spatially contiguous) prefixes.

The HMM proposed to model routing system states per destination prefix or per set of destination prefixes undergoing the same state transition(s), is characterized as follows (see Figure 1):

- N (number of hidden states in the model): 5. These five states that characterize (hidden) routing system states that are not directly observable at the local BGP router are labeled as follows: State_1 (S1): No AS_Path change, State_2 (S2): AS_Path re-initialization, State_3 (S3): AS_Path length increase, State_4 (S4): Path exploration Hit, State_5 (S5): Exploration-less withdrawal. The latter accounts for withdrawals that do not lead to a path exploration hit. Indeed, using the terminology introduced in Deliverable D3.6, the HMM should typically account that only $W(A_0)$ is a trigger for exploration hit, withdraws of intermediate states associated to announcements $A_1,..,A_m$ occurring before $W(A_0)$ shall not be considered part of an exploration sequence that affect downstream neighbors.
- M (number of distinct observation symbols per state): these symbols correspond to the AS sequences received in BGP UPDATE messages, stored in the RIB-In, and processed by the BGP route selection process that populates the Loc-RIB.
- A: state transition probability distribution $a_{ij} = P(y_{j,t+1}|y_{i,t})$ with $1 \leq i,j \leq N$ correspond to the individual state transition of the routing system state.

- B: the observation probability distribution in state j, $b_j(x) = P(x_t|y_j,t)$.
- Π: the initial state distribution

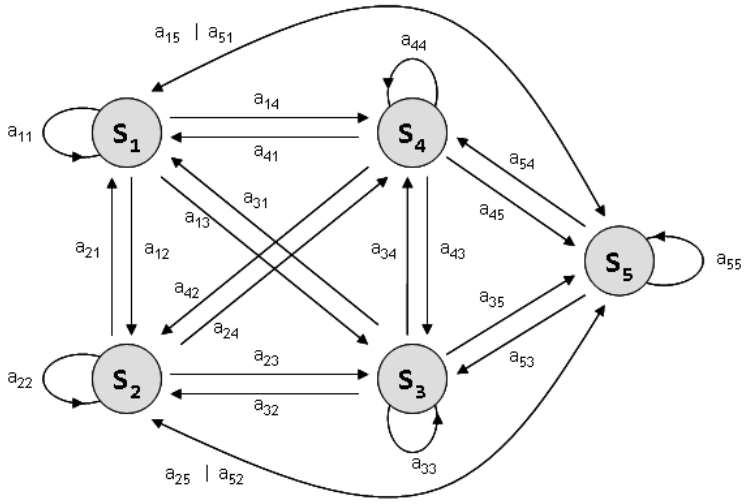


Fig.1: Five-State HMM model

2.2.2 AS-Path Sequence Classification

A classifier is a function h that maps observed AS-path sequence (x) to BGP state event classes. The goal of the learning process is to find a function h that correctly predicts the class h(x) of new AS-path(s). A training example for this learning process is a pair (x,y), where y refers to the state and to its associated label l. The training data set actually consists of sequences of (x,y) pairs. They exhibit sequential correlation which is characteristic of the path exploration phenomenon. For example, before occurrence of a topological change, all y label values will be "no AS-Path change". Afterwards, all q label values will be "AS-Path increase". Such patterns can be exploited to improve the prediction accuracy of the classifier. In the present case, it is possible to explore sequences by looking at the distribution of legitimate sequences and then observe distribution changes when the state of the routing system changes. The goal is to construct a classifier h that can correctly predict a new state label sequence y = h(x) given an input sequence x.

Assume that the cost function $C(y_i,y_j)$ gives the cost of assigning state label value i to an example whose true label is j. The goal is to find a classifier h with minimum expected cost. The cost function assesses the penalties associated to selection of BGP routes that contain (part of) the path exploration sequence: missed path exploration events, false positive detections (the classification declares a path exploration event when in reality there is none; such an error may typically occur when decision is taken too rapidly) and false negative detections (the classification does not declare an event to be a path exploration event when in reality it is; such an error typically occurs when the decision is taken too slowly).

The HMM model is a probabilistic model of the way in which the observed sequence x_i and state label y_i strings are generated: it is a representation of the joint distribution P(x,y). It is defined by two probability distributions: the transition distribution $P(y_t|y_{t-1})$, which tells how adjacent y values are related, and the observation distribution P(x|y), which tells how the observed x values are related to the hidden y values. As the HMM model is a representation of the joint probability distribution P(x,y), it can be applied to compute the probability of any particular y given any particular x: P(y|x). Classification a new observation sequence is performed by selecting the class with the minimum expected cost as provided by the formula (see also problem 2 in Deliverable D3.6, Section 4.2.2) that predicts the optimal value y' given the observation x.

$$y' = \arg\min_{y_i} \sum_{y_j} P(y_j|x)\, C(y_i, y_j)$$

Incorporating the cost function C into the classification task of AS-Path sequences consists thus in predicting the (conditional) joint distribution of all of the (state) labels in the output sequence: $P(y_j|x)$. If this joint distribution can be accurately predicted, then the cost function can be evaluated and the optimal decisions can be chosen for the observation sequence x.

In practice, as the length L of the sequences can be very long, the direct evaluation of this equation requires $O(N^L)$ probability evaluations (where N is the number of labels) which can be impractical. However, when the cost function is only concerned with classifying the entire sequence, this computation can be performed in $O(N^2L)$ time, where L is the length of the observation sequence. In this case, finding the y' with the highest probability consists in computing:

$$y' = \arg\max_{y_j} P(y_j|x)$$

This expression can be computed by means of the Viterbi algorithm which corresponds to the second HMM problem. Computation by application of the Bellman's dynamic programming algorithm consists in assigning for each class and each time step of a time interval [0,t], the probability of the most likely state transition sequence. When the algorithm reaches the end of the sequence, it has computed the most likely path from time 0 to time t and its probability. In other terms, by determining the most probable state sequence given a certain observation sequence x, one can isolate observation sequence corresponding to "path exploration hit" and remove their difference from the BGP route selection process.

As explained in Deliverable D3.6, HMM models suffer from two principal drawbacks: i) the structure of the HMM is often a limited model of the true process producing the data. Part of the problem stems from the Markov property. Any relationship between two separated y values must be communicated via the intervening y's. A first-order Markov model, i.e., where $P(y_t)$ only depends on y_{t-1}, cannot capture these kinds of relationships; and ii) the HMM model generates each x_t only from the corresponding y_t, which makes difficult to use a method based on sliding window of x_t values to predict a single y_t.

2.3 Conditional Random Field

Conditional models have been explored in the scientific literature to overcome the limitations of the HMM model. Compared to the joint probability distribution represented by the HMM model, conditional models do not explain how the observation sequences are generated but instead predict the state label values given the observation sequences: conditional models represent $P(y|x)$ rather than $P(x,y)$. This permits them to use arbitrary features of the observations including global features, features describing non-local interactions, and sliding windows. In particular, Conditional Random Fields (CRF) that overcome the label bias problem, model the relationship among adjacent state label pairs as a Markov Random Field conditioned on the observation sequences (used as input), i.e., the influence between adjacent state label values is determined by the input. The objective in investigating CRF models, the HMM discriminative-equivalent, is to determine if we can reduce the error rate observed with HMMs.

2.3.1 CRF and Chain-Structured CRF

Conditional Random Fields (CRF) models the relationship among adjacent pairs y_{t-1} and y_t as an Markov Random Field conditioned on the x inputs. In other terms, the way in which the adjacent y values influence each other is determined by the input features. Lafferty [19] defines the probability of a particular label sequence y given observation sequence x to be a normalized product of potential functions $M_t(y_{t-1},y_t|x)$ of the form

$$M_t(y_{t-1}, y_t | x) = \left(\sum_\alpha \lambda_\alpha t_\alpha(y_{t-1}, y_t, x) + \sum_\beta \mu_\beta s_\beta(y_t, x) \right)$$

In this formula, $t_\alpha(y_{t-1},y_t,x)$ is a transition function of the entire observation sequence and the labels at positions t and t–1 in the label sequence; $s_\beta(y_t,x)$ is a

state function of the label at position t and the observation sequence; and λ_α and μ_β are weight parameters to be estimated from training data.

We use the simplified notation of this formula by writing $s_\beta(y_t,x) = s_\beta(y_{t-1},y_t,x)$ and $f_a(y_{t-1},y_t,x)$ as being either a state function $s(y_{t-1},y_t,x)$ or a transition function $t(y_{t-1},y_t,x)$. Using this notation, we write the conditional probability $P(y|x)$ of a label sequence y given an observation sequence x as follows, where $Z(x)$ is a normalization factor (needed because the potentials M_t are unnormalized values):

$$P(y|x) = \frac{1}{Z(x)} \exp\left(\sum_\alpha \lambda_\alpha \sum_{t=1}^{L} f_\alpha(y_{t-1}, y_t, x) \right)$$

For a chain-structured CRF in which each label sequence is augmented by start state y_0 and end state y_{L+1}, with labels *start* and *end* respectively, the probability $P(y|x)$ of label sequence y given an observation sequence x may be efficiently computed using matrices. For this purpose, we define a set of matrices $\{M_t(x)|t=1,..,L+1\}$, where each $M_t(x)$ is a transition matrix for each stage t=1,..,L+1, with elements of the form

$$M_t(y_{t-1}, y_t | x) = \left(\sum_\alpha \lambda_\alpha f_\alpha(y_{t-1}, y_t, x) \right)$$

For a given position t (from y_{t-1} -> y_t), the transition matrix M_t is the score of arc y_{t-1} -> y_t at position t in the trellis representation (see Section 2.3.2).

Moreover, the conditional probability of label sequence y given observation sequence x may be written as the product of transition values along the path for a given y= (start,y_1,..,y_L,end)

$$P(y|x) = \frac{1}{Z(x)} \prod_{t=1}^{L+1} M(y_{t-1}, y_t | x) = \frac{1}{Z(x)} \prod_{i=1}^{L+1} \exp\left(\sum_\alpha \lambda_\alpha f_\alpha(y_{t-1}, y_t, x) \right)$$

The normalization factor Z(x) for observation sequence x can be computed from the set of $M_t(x)$ matrices and is given by the (start,end) entry of the product of all L+1 $M_t(x)$ matrices.

$$Z(x) = \left[\prod_{t=1}^{L+1} M_t(x) \right]_{start,end}$$

2.3.2 Determine Routing System States from Observation

Using the graphical treillis representation of CRF, we expand all possible label assignment for each node, and add the dummy nodes start and end. This leads to the model represented in Figure 2, where i=2,3,4. Note that this Figure does not represent the full treillis to keep it readable.

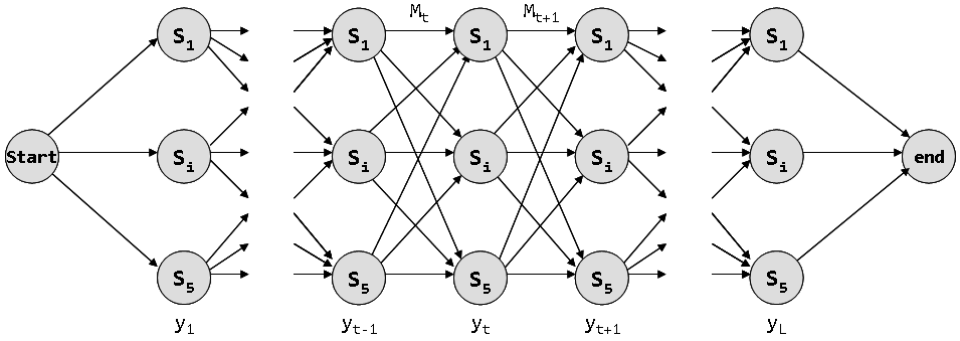


Fig.2: Treillis Representation

When a given AS_Path sequence is processed each AS part of this sequence, the interesting transition are those reaching S_4 (and thus those that cumulate multiple S_4

label along their path even when interleaved by other state labels). To overcome a general limitation of sequence models, whether generatively or discriminatively trained is the Markov assumption among labels: any state label at y_t is independent of all previous labels given its immediate predecessors at $y_{t-k},...,y_{t-1}$. Thus, these models represent dependence only between nearby AS numbers but cannot represent the higher-order dependencies. To relax this assumption, we apply the skip-chain CRF, a conditional model that collectively segments AS sequences into sub-segments and classifies the sub-segments by entity type, while taking into account probabilistic dependencies between distant sub-segments. These dependencies are represented in a skip-chain model by augmenting a chain-structured CRF with factors that depend on the labels assigned to distant but AS numbers with similar properties that occur within a given AS_Path sequence. Formally, the skip-chain CRF is defined as a general CRF with two clique templates: one for the linear-chain portion, and one for the skip edges. In the present case, by skipping chains between two skip edges (leading to label S_4), CRF can learn to label multiple occurrences of AS_Path exploration consistently for the same prefix. Using the notation introduced in Deliverable D3.6, skipping chains CRF enable for receiving routers to detect and identify sequences as well as prevent sending to downstream routers announcement(A)-withdrawal(W) BGP update sequences of the form: $A_0,W(A_0),A_1,\{A_1,...,A_m\}^*,W(A_1),...,W(A_{m-2}),A_{m-1},\{A_{m-1},A_m\}^*,W(A_{m-1}),A_m,W(A_m)$.

Note that training CRF requires a global adjustment of the λ values. This global training is what allows the CRF to overcome the label bias problem by allowing the x_t values to modulate the relationships between adjacent y_{t-1} and y_t values.

3. Experimental Model

3.1 BGP XORP Model

The XORP BGP update processing is implemented as a pipeline of XORP processing stages as illustrated on Figure 3. Each line of the pipeline is connected to a BGP peer on the output path as well as on the input path to be able to send and receive update messages. The information going through the pipe is a BGP route pointer, associated with its list of attributes. This means that once a message arrive on the BGP input path, each route and its attributes will go through the pipeline independently. Prefixes are pushed through with messages from the RIB_In processing stage up to the BGP decision process stage, which will elect the route to be propagated through the next peer and then pass on the decision to the Fanout processing stage. The Fanout processing stage is responsible for queueing the messages in the output path, finishing at the RIB_Out processing stage. The RIB_Out processing stage collects all messages that have the same path attributes and builds and sends a single BGP update message to the next peer.

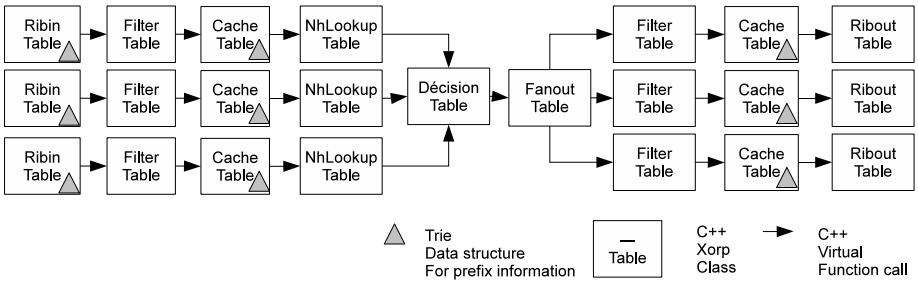


Fig.3: XORP BGP update processing pipeline

The route processing stages are implemented as C++ classes connected via virtual function C++ calls. This allows flexibility in composing the pipeline of processing stages, as it is possible to connect them dynamically via the virtual function calls, enable or disable a processing stage, insert or remove a new processing stage on the fly. It also eases the implementation of a new processing module by starting from a general template (an empty processing stage) and adding new functions to it and writing the associated unit tests.

Route information is stored in a Trie data structure, where the index in the route prefix, and the leaves are pointers to route attributes. Route attributes are shared. As we can see in Figure 3, this basic Trie data structure is present at several locations on the pipeline: at the RIB_In processing stage, the cache input path and the cache output path. This Trie is updated by the sequences of BGP update messages arriving in the pipeline. At the RIB_In processing stage, route announcements create new entries in the Trie or update them if they already exist. Route withdrawals delete entries from the Trie.

3.2 Memory Module

We consider a processing stage that keeps the last n routes announced or withdrawn in memory in order enable their processing by means of semi-supervised machine learning. As the route attributes are shared within the current pipeline design (there is one Trie per peer, and several tries on a line), we decided to implement a new processing stage that keeps track of the history of the attributes for a given route pointer in a Trie data structure. This means that when a withdrawal message is received for a given route, the route is not removed from the Trie, but the withdrawn for this route is recorded.

The placement of the historical processing stage is shown on the Figure 4. As with routes stored in the RIB_In history of routes are kept per peer in a separate Trie and the new processing stage is located in the input path of the pipeline, before the decision process processing stage. Thanks to the modularity of the processing

pipeline implemented via C++ classes and virtual function calls, it is possible to insert or remove the historical processing stage from a peer on the fly.

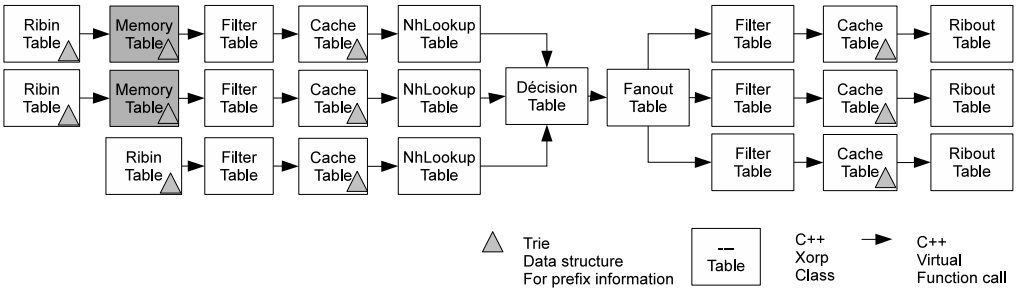


Fig.4: XORP BGP update processing pipeline with the memory module

In this section, we first describe in more details how the implemented Memory processing stage fits within the BGP pipeline design of XORP (Section 3.2.1). We then describe the data structures used to remember the history of attributes extracted from the BGP messages going through the pipeline (Section 3.2.2).

3.2.1 Interaction with the Pipeline

The XORP BGP processing stages are implemented by a C++ class inherited from a base class BGPRouteTable containing the following virtual methods to implement in the child class. The method is first called in the first processing stage of the pipeline (RIB_In in our case). Then, the call is propagated hop by hop from one processing stage to the other.

1. ***add route method*** is called whenever the currently processed route was not present in the Trie of the RIB_In processing stage. This allows us to know that a route is considered new by the RIB_In processing stage. That means that either the route was not in the Trie because it was deleted before or that the route was not in the Trie because it was never seen before. A route that is not in the Trie of the RIB_In processing stage can still be in the Trie of the Memory stage as we keep the history of deleted routes too. If the route is not in the Trie of the Memory stage, we initialize its history, if it's already in the Trie, we simply add a new element in its history of attributes.
2. ***replace route method*** is called if the currently processed route is already present in the Trie of the RIB_In processing stage. This means that the route was already seen before and that the previous BGP message for this route was not a withdraw. When called in the Memory processing stage, this method simply adds a new element in the history of attributes of the route.
3. ***delete route method*** is called when a withdraw message is received. In that situation, the route is removed from the Trie in the RIB_In processing stage. However, in the Memory processing stage it is not removed but the history of attributes for this route is updated to reflect this event. As a withdraw message doesn't contain any attributes, we reflect the fact that the route was removed by using an empty pointer in the attribute history list. It is worth noticing when keeping the history of BGP update sequence in the Memory processing stage that if several BGP withdrawn are received for a given route, the original RIB_In processing stage calls this method only once.

3.2.2 Data Structures in the Memory Processing Stage

The data structure in the RIB_In processing stage is a Trie where by using the IP address of a prefix, one can lookup the corresponding route for this particular prefix. A route data structure is essentially composed of a structure containing the network prefix of the route and a pointer to its path attribute list. The RIB_In Trie is augmented to be able to lookup not only by using the network address but also by using the path attribute list. For this, all leafs of the Trie having the same path attribute are linked together in a doubly linked list. To be able to find the route by path attributes, a dictionary associating path attributes list to the corresponding doubly linked list is present in the C++ class definition.

For the data structure used in the Memory processing stage, we have chosen to use a simple Trie, without the option to be able to lookup a route by its path attribute list. This choice has been made because we are interested in the attribute history of

a route for a given prefix. Once we have this history, we can compute all the measurements required to run the machine learning algorithm. The leafs of the Memory processing stage Trie are not anymore pointers to a route like in the RIB_In processing stage but pointers to an augmented route data structure containing a list of attributes augmented with a timestamp to retrieve the time of reception of a particular attribute in the history. To avoid that the Memory processing stage takes over too much memory resource when tracking the history of attributes of a route, the list of attributes is circular and limited in size.

In addition to the list of attributes pointer kept to access the history of a given route per prefix, we store the current measurements values used to predict whether a path exploration sequence will end up with either a withdrawal or a re-announcement. For the learning phase, we store a vector of the last n measurements corresponding to the last n received BGP messages. Storing this vector enables to select the measurements to extract from the vector during the learning phase as well to retrieve the semi-supervised outcome associated with this measurement.

Data structure	size in bytes
RibIn Route data structure	136
List of Augmented Attribute list pointers	16
Measurements	96
Vector of measurements pointers	24
Timestamp Augmented Attribute list pointer	16
Total	272

Table 1: Size of the augmented route data structures

Table 1 summarizes the size of the data structures used within the augmented route data structure used in the Memory processing stage Trie's leafs. The data structure size is doubled compared to the regular route data structure used in the RIB_In processing stage, but this does not have much influence on the memory consumption as we will see in the Section 5. Most of the memory in the Trie data structure is consumed taken by the list of route attributes list that have a variable size but are hopefully shared by all routes present in the system. This is why the augmented data structure only uses pointers towards the attribute list.

4. Experimental Setup and Methodology

4.1 Experimental Setup

Our experiments are based on a "replay" of BGP monitoring traces from Routeview Project and RIPE BGP traces. This replay is possible by injecting BGP dumped messages into our implementation of Path Exploration within XORP. We do not collect ourselves BGP messages. Instead, as explained below, we download BGP messages from the Routeview Project and RIPE BGP traces. As a consequence, for a base experimental setting, a single machine running our implementation into XORP is enough for performing the tests. The iLab.t experimental platform is thus not strictly necessary for re-playing the BGP traces collected off-line. This explains why we consider use of stand-alone machine(s).

4.1.1 Experimental Hardware Platform

For experiments using the Routeview project date, a single server, with the following configuration:

- Operating System: Linux 64 bits, with kernel 2.6.28
- CPU: Intel Xeon E5430, quad-core CPU, with 6 MB of L2 cache shared by pairs of cores, and 32 kB of L1 cache on each core
- Memory: 4.8 GB of main memory

For experiments using the RIPE BGP data, two servers with the following configuration:

- Operating System: CentOS, with Linux Kernel 2.6
- CPU: Intel Quad-Core Xeon E5405, 2.6 GHz (12MB Cache)
- Memory: 32 GB DDR2-667 registered ECC (16 DIMMs)
- Disk: 4 Seagate 300 GB SAS drive, 15,000 rpm
- NIC: Intel® PRO/1000 PT Ethernet Server Adapter, 2x RJ45, PCI-e

4.1.2 Experimental Data

The experimental data originate from two different sources:

i) ***Routeview Data***: on the month of November 2009 coming from BGP updates recorded from a total of 42 peers at the Oregon routeview monitor [3], the total number of update messages processed was about 89 millions. On the machine we used for experimentations, it took about 5h to process this month of data without the Memory processing stage enabled, about 9h to process it with the Memory processing stage enabled and a history size of 2 attributes, 11h to process it with a history size of 4, 12h with a history size of 5 and 15h with a history size of 10. Note that the learning phase took about 24h to be processed entirely.

ii) ***RIPE[1] (Réseaux IP Européens) BGP Data***: BGP updates are taken from the RIPE Routing Information Service (RIS). RIS is a RIPE NCC project that collects and stores routing data from the Internet, on several locations around the globe. RIS offers tools bringing up this data to the Internet community. Raw data are collected by the Remote Route Collector (RRC) using Quagga routing software, stored in MRT format. This format is described in the IETF document entitled MRT routing information export format draft-ietf-grow-mrt-11.txt). These files can be read using libbgpdump, a library written in C, currently maintained by the RIPE NCC. BGP UPDATE messages are parsed and stored in the Adj-RIB-In, then and then processed by the machine learning algorithm. Note that in usual programs based on MRT feeds, BGP messages are all injected at once. This creates an issue when one wants to evaluate the efficiency of the learning based on measurements that depends on time, such as the message arrival frequency. Fortunately, MRT files contain a timestamp for each message. We add a virtual clock inside the memory processing state. This clock is modified dynamically on the fly according to the timestamps contained in the MRT file. All measurements are then based on that virtual clock.

[1] RIPE (Réseaux IP Européens) is a collaborative forum existing since November 1989 and open to all parties interested in wide area IP networks. Work is carried out by individual volunteers in their own or their organisation's time.

4.2 Experiments

4.2.1 Run-time Memory Cost

Measurements values and computation are stored within a single C++ class (See Table 1). The methods from this class access the history of the augmented route data structure to perform their computation. The timestamp stored in each element of the history is also used for the computations which depend on the timing of the received BGP messages. Currently measurements values are all scalars and stored as attributes of a single Measurement class instance.

Whenever a new BGP update message is processed by the Memory processing stage, those values are updated by calling the methods of the Measurements class, with one method per value to update. The initial values of the measures are initialized within the class constructor.

Only the test program was used for this evaluation, along with a single script monitoring the memory consumption of the program over time. Two scenarios were considered: On the same set of data, the testing program is executed with the Memory processing stage activated with a variable history size, and without. The testing program is also executed in learning phase as described in the Section 3.2.

4.2.2 Filtering of BGP Update Messages

The reasons why we would like to filter the updates stream get from RIPE NCC and RouteViews are two folds. Firstly, RIPE NCC and RouteViews often use software routers (Quagga/Zebra) to collect BGP udpates from remote peers with multi-hop eBGP sessions, which frequently suffer session resets, during which the entire BGP table has to be re-transferred. Secondly, the BGP path explorations are mainly caused by the variation of AS-Path attribute, thus the noise introduced by the changes of MULTI_EXIT_DISCRIMINATOR (MED), COMMUNITY, etc. attributes should be removed.

In our filtering process, all the updates whose AS-Path attributes make no changes to BGP route table will be filtered out. In this way, we can remove updates due to both BGP table transfers and internal dynamics of last-hop AS.

4.2.3 Learning Results

The objective is to experiment the performance of the implemented machine learning techniques as part of a BGP routing engine by using measurement's history provided by the Memory processing stage. The following machine learning techniques are experimented: C4.5 Decision Tree, Hidden Markov Model (HMM) and Conditional Random Field (CRF). The output of the learning algorithm triggers the selection of an alternate AS_Path that populates the Loc-RIB upon anticipated path exploration event detection and identification. The resulting action involves the suppression of the churn generated towards downstream BGP peers by selecting the alternate best AS_Path to be advertised to the downstream BGP peers.

The following metrics are used on the BGP speaker running the HMM and CRF machine learning algorithms specified respectively in Section 2.2 and 2.3:

1. The number of actual path exploration events detected (true positives). The number and rate of false positives (i.e., a sequence of events is labeled as path exploration while it is not the case). The number and rate of false negatives (i.e., a sequence of events is ignored while it should have been labeled as path exploration).
2. The time required for the detection of path exploration event.
3. The correctness of the selected path and the proportion of correctness of selected AS_Paths: for the number of path exploration events detected the number of events for which the next stable sequence is returned in the Loc-RIB and the RIB-Out.
4. The probability of selecting a wrong AS_Path and the impact of selecting a wrong AS_Path. The impact of selecting the wrong AS_Path is the deviation of the wrongly selected AS path from the AS_Path that would be selected after convergence (i.e. after full path exploration phase). From this deviation an estimate can be achieved on the number of AS's that will be affected by that decision.

The proportion of correct decisions should be high enough. A classifier is a function that maps observed AS-paths to BGP state event classes. The goal of the learning process is thus to find a function, i.e. a classifier, that correctly predicts the class of topologically correlated AS-path(s) with the minimum expected cost. The cost function assesses the penalties associated to the selection of BGP routes that contain (part of) the path exploration sequence:

- Missed path exploration events.
- False positive detections: the classification declares a path exploration event when in reality there is none; such an error may typically occur when decision is taken too rapidly.
- False negative detections: the classification does not declare an event to be a path exploration event when in reality it is; such an error typically occurs when the decision is taken too slowly.

The objective is also to experiment the feasibility of implementing machine learning techniques as part of a BGP routing engine by using measurement's history provided by the Memory processing stage. For this purpose, we evaluate the performance of the C4.5 algorithm as follows:

- First, we look at the Beacon messages coming from two different AS (AS3549 and AS852). The machine learning algorithm, in such as case, is applied on a per peer basis.
- Second, we consider the history coming from several peers at the same time. This dataset is obtained by merging the sequences of all BGP updates coming from all the 42 peers available at the Oregon Routeview monitor.

Metrics considered are the learning set error ratio and the test set error ratio.

For this purpose, two sources of input are considered:

1. We consider BGP messages from the Routeview project. Two months of data are used: November and October 2009. The data used to train the C4.5 machine learning algorithm is based on Beacons. A BGP Beacon is an unused prefix which has a well-defined schedule for announcement and withdrawal. The pattern used by the Beacons is very simple: a network prefix is announced at time t to be finally withdrawn at time t+2h. The beacon we consider are those announced by the RIPE NCC consortium.
2. We also consider the BGP messages from the BGP RIS Raw Data of the RIPE NIS project. Four separate months of data are used: April’09, July’09; October’09, January’10 and April 2010 as training set. These data are collected at Amsterdam (AMS-IX), Otemachi, Japan (DIX-IE) and Stockholm, Sweden (NETNOD).

5. Experimental Results

This section details the results obtained for the experiments detailed in Section 4.2. These results were obtained by applying the machine learning technique detailed in Section 2.1 using the experimental BGP routing engine model described in Section 3, executed on the experimental setup depicted in Section 4.1.

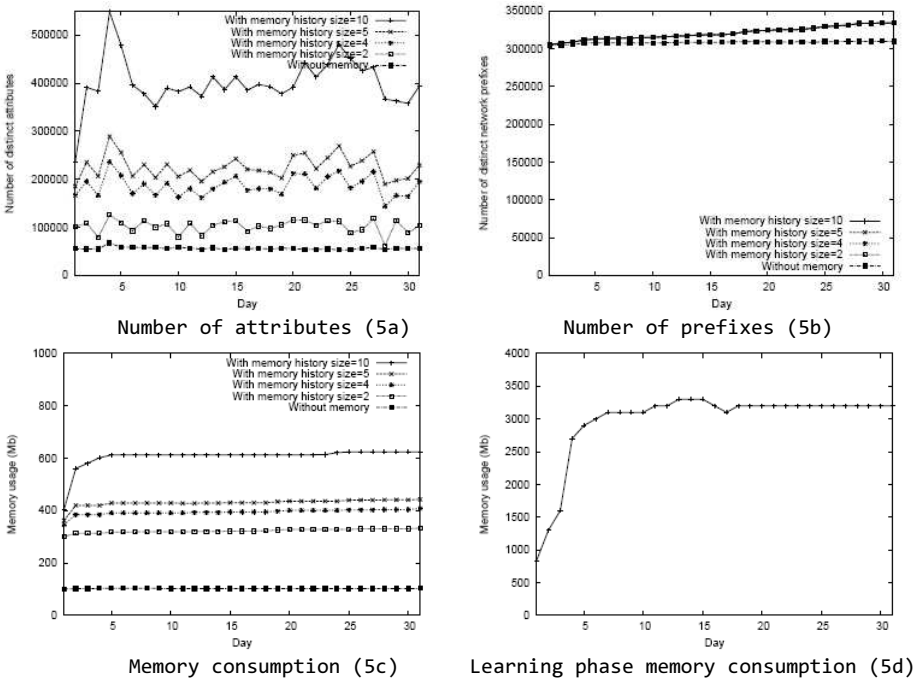
5.1 Memory Cost

5.1.1 Run Time Memory Cost

Results are shown on Figure 5. On the x axis of the graphs we plot the day of the month, on the y axis we plot the memory consumption in MB, the number of attributes in memory and the number of network prefixes kept in memory. On a full month of processed update messages coming from 42 peers the memory consumption reaches a stable state where their number of attributes and/or prefixes kept in memory do not have an impact on the memory cost anymore.

On a month of BGP update processing, Figure 5(b) shows that the number of prefixes kept in memory with the Memory processing stage is about the same as the number of prefixes kept in memory without the Memory processing stage enabled, independently of the history size used. Those prefixes are kept in a separate Trie which would mean that the memory consumption would double in term of network prefixes kept in memory. The situation with the number of attributes is different. The total number of attributes in memory is directly related to the history size parameter. As we can see

on Figure 5(a) without the Memory processing stage activated, the total number of attributes kept in memory is about 50.000, and can reach a total of up to 400.000 with a history of 10.



Number of attributes (5a) Number of prefixes (5b)

Memory consumption (5c) Learning phase memory consumption (5d)

Fig.5: Memory usage of the Memory Processing stage

Despite the fact that attributes are shared, the memory consumption of the Memory processing stage is directly related to the number of attributes kept in history. As we can see on the Figure 5 the number of attributes cost in term of physical memory is reasonable and it is possible to keep the memory footprint of the Memory processing stage less than 500Mb with a reasonable history size. Without history, the cost of keeping the attributes in memory is about 100Mb.

5.1.2 Learning phase memory cost and statistics

Despites the fact that the learning phase memory consumption seems to be stable too, we believe that it is an artifact coming from the physical limitation of the machine. The machine that we used was heavily swapping when we ran the learning phase experiment. Also, out of roughly 4GB of memory available on the machine, about 1GB is allocated to the kernel, which leaves about 3GB of memory for the process that we ran. We can see on the Figure 5(d) that the memory consumption is more than 3GB, which means that the machine is reaching physical memory exhaustion. The figure 4 gives us some information about the distribution of the samples extracted during the learning phase. The learning phase was configured to automatically extract sequences that left a minimum time of 8 min between two update messages. If possible, the sample number 4 was extracted from this sequence and labeled with the end of the sequence event: either an announcement or a withdrawal. On a month period over 7.2M samples were extracted with this method, spawning all the 300k prefix range.

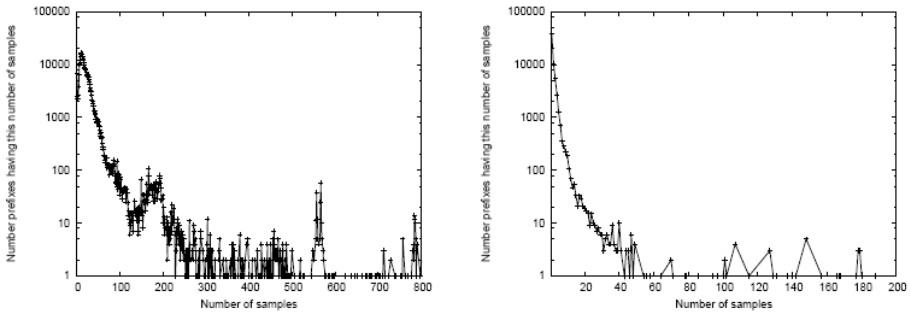


Fig.6: Samples distribution for Announcements (6a) and Withdrawals (6b) over a month of learning

To be able to inject samples in the machine learning algorithm, a minimum amount of samples that are distinct enough is needed to differentiate between (withdrawals/ announcements to be detected. By comparing Figure 6(a) and 6(b), we observe that more samples end with an announcement than with a withdrawal. Their repartition is actually very uneven, with more than 7M of samples ending with an announcement and about 128k samples ending with an withdraw. This could be explained by our extraction method, which proceeds by allowing a maximum amount of time between two updates for a given prefix and requires further investigation. Consequently, the number of samples to be injected in a machine learning algorithm is generally very low, even when extracting those data over a full month. The proportion of prefixes containing enough samples is also very low. As we can observe on Figure 6(b), about 10 prefixes with more than 20 samples are ready to be injected in the C4.5 machine learning algorithm, over a full month of data concerning more than 300.000 prefixes.

5.2 Filtering of BGP Update Messages

The effect of filtering BGP update messages is depicted in Figure 7, where the y-axis is the ratio of removed/total updates while the x-axis is the ID of peers according to the ratio of removed partition in a descendant order. More than 50% of the updates in 13 peers are filtered out and more than 30% of them in 33 peers. It is probable that the removed updates would bias the update frequency and total amount measures.

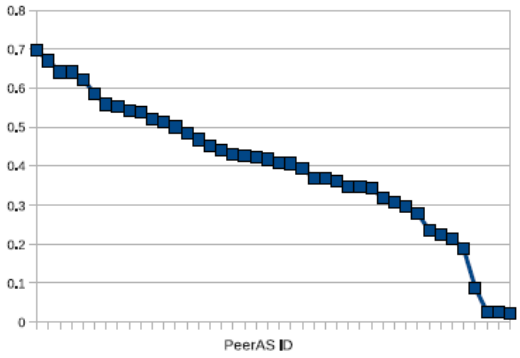
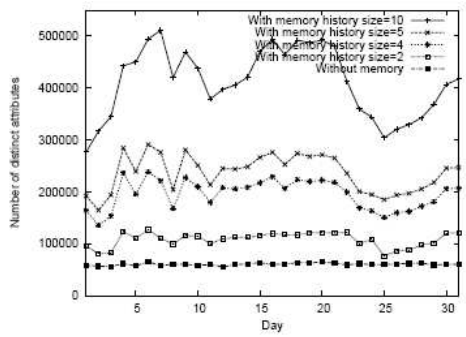
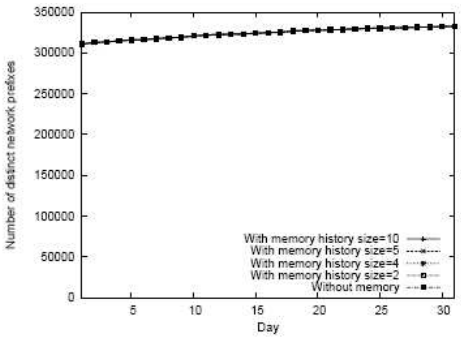


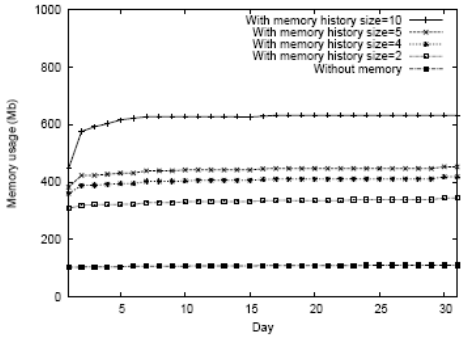
Fig.7: Proportion of BGP updates filtered out



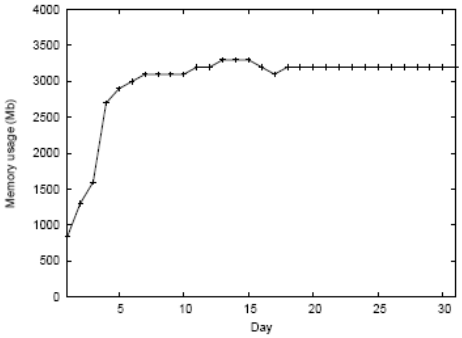
Number of attributes (8a)



Number of prefixes (8b)



Memory consumption (8c)



Learning phase memory consumption (8d)

Figure 8: Memory usage of the Memory processing stage by using filtered updates injection

Then, we repeated the experiments described in Section 5.1.1 and 5.1.2 by injecting the filtered set of BGP update applied to the month of December 2009. One can see on Figure 8 that the memory cost is still the same and that applying filters to the set of udpates doesn't decrease the memory footprint of the Memory processing stage. This observation confirms that our implementation is probably reaching a plateau, where all the history lists are full after a month of monitoring. An interesting difference between those measurements and the previous ones is that the number of prefixes difference is lower with the filtered BGP updates as we can see on Figure 8(b). Note that this difference could come from the use of different month (December 2009 instead of November 2009) in our experiments. This difference still needs to be investigated. The learning phase is still as costly as before. After a month of filtered BGP update messages where injected set contains about 62M of messages, the process is still exhausting all the physical memory available on the machine as we can see on Figure 8(d).

Figure 9 plots the number of samples distribution per prefix extracted from the learning phase. As already observed, the number of ’A’ samples is much higher than the number of ’W’ samples. Over a total of about 9.2M samples, only 1.270 samples are ’W’ samples. This observation is reflected in Figure 9(b). Most of the samples size are limited to one and there are only very few IP prefixes for which there are never more than 10 samples. Considering the month of December, it would be thus quite difficult to extract learning information for the ’W’ case with the proposed method.

5.3 Learning Results

Being able to apply a machine learning algorithm to the history of messages arriving on a BGP router is one of many possible applications of the Memory processing stage. In this section, we demonstrate the feasibility of running a machine learning algorithm together with a BGP router by using the measurement’s history provided by the Memory processing stage. The results are encouraging as first integration trial although not perfect. In this section, we describe the obtained results by means of the experimental software used to demonstrate how the C4.5 algorithm performs on data extracted from a BGP routing engine running on XORP platform.

5.3.1 Training Data Set

The experimental data used to train the C4.5 algorithm in order to automatically build a decision tree from the measurements computed in the Memory processing stage are summarized in Table 2. For this experiment, we rely on two simple BGP beacons provided by the RIPE consortium (at http://www.ripe.net/ris/docs/beaconlist.html) assuming that if the machine learning algorithm is not able to predict deterministic announcements/withdrawals patterns, it will be very difficult to predict non-deterministic ones. The pattern used by those beacons is very simple: A well known prefix is announced at time t to be finally withdrawn at time t+2h. The hours of announcements and withdrawals are well known too and so it is easy to know if a given sequence of messages happening at a well known hours corresponds to a BGP withdraw of the prefix or a BGP announce. As shown in Table 2, we used one network prefix announced from Amsterdam,UK and the other announced from London,UK.

Name	AS number	Month	network prefix	# of A samples	# of W samples
A	3549	11/2009	84.205.64.0/24	19	114
B	3549	11/2009	84.205.65.0/24	92	167
C	852	11/2009	84.205.64.0/24	1	115
D	852	11/2009	84.205.65.0/24	29	167
E	3549	12/2009	84.205.65.0/24	107	176
F	3549	12/2009	84.205.64.0/24	23	140
G	852	12/2009	84.205.64.0/24	1	140
H	852	12/2009	84.205.65.0/24	22	176
X	All	11/2009	84.205.64.0/24	115	124
Y	All	11/2009	84.205.65.0/24	199	167
Z	All	11/2009	192.12.120.0/24	645	704

Table 2: Training data sets

To evaluate the efficiency of the C4.5 machine learning algorithm, we looked at the Beacons messages coming from 2 different AS chosen randomly, on two successive months of the year (November and December 2009). On important parameter of this particular setup is that we are applying the machine learning algorithm as well as the learning phase sample extraction on a per peer basis. Contrary to the setup used in Section 5.1 and 5.2, we do not use a global and single Memory processing stage where all the BGP messages coming from all the peers are stored, we instantiate one Memory processing stage per peer and we compute the history coming from a particular peer. Moreover, to test the applicability of machine learning techniques on history coming from several peers at the same time, we use table a data set extracted from the learning phase experiment described in Section 4 which involves all peers. As indicated on the last three lines of Table 2, this data set is obtained by merging the sequences of all BGP updates coming from all the 42 peers available at Oregon routeview monitor (http://www.routeviews.org/). As expected, the number of extracted 'A' samples is higher but the number of 'W' samples stays about the same as in the other data sets. For this data, we extracted beacons as well as one of the top prefixes as shown from the ranking on Figure 6(a) and 6(b). This prefix is announced from a real autonomous system and seems to be quite active as we can see by the number of extracted 'A' and 'W' samples from it. The last two columns of the tables indicate the number of 'A' samples and 'W' samples extracted from the learning phase with that particular network prefix, AS and month. As we can see the number of 'A' samples is quite low. This is explained by the fact that BGP re- announcements on a single peer do not generally contain a long sequence of messages. For the learning phase, we extract the message number 4 and a low number of samples means that there was less than 4 BGP messages when the prefix was re-announced. Withdrawals trigger a path exploration sequence which contains more messages and thus, allow us to extract more samples to be injected into the C4.5 machine learning algorithm.

5.3.2 Integrated Learning Results

The results described in Table 3 are obtained with the dataset described in Section 5.3.1 injected into the original C4.5 automatic decision tree algorithm implementation from Quilan [17]. Each leafs of the tree is a prediction of the end of a sequence of BGP messages concerning a prefix where there is maximum 8 min of silence between two messages related to this prefix.

The training data set used to train the C4.5 algorithm is indicated on the first column of Table 3. The last two columns indicate the learning set errors and the test set errors. The learning set errors ratio is the proportion of samples that C4.5 was not able to classify simply by using the training set over all the injected samples. This proportion is generally quite small on all the datasets used, which means that C4.5 algorithm is able to successfully extract decision information from the injected data. The Test errors ratio, in the last column, indicates the proportion of errors obtained after classifying samples that were never seen before by the C4.5 algorithm; that is the proportion of injected samples that were erroneously classified. To generate data for the test set, we use 2 methods:

- The first method uses the beacons data set for the training the test. Once a training set is selected, the test is selected depending on one or several differences between the training set. The test set can be generated from a different AS, a different beacon prefix or a different month as it is shown on the first 4 lines of the table. When the training set and the test set are sharing their month, their prefix or their AS it is indicated by a checkmark on the table. For example, in the lines 4 to 8 of the table 5.2, the training set and the test set are all coming from the same month but have a different prefix and a different AS. Those 4 lines where the training set and the test set share the same month constitutes what we call a scenario in our experimentation. On the Table 3 scenarios are separated by a horizontal line. For the training set X and Y, their common information is that they are extracted from the same month, and from all the same peers but they use a different prefix for the test set.
- The second method split the data set into two parts: the first part contains 70% of the total number of samples is used for the training and contains the first 70% of the samples extracted during the learning phase of the experiment. The second part contains 30% of the total number of samples and contains the last 30% of the samples extracted during the learning phase. This

method is used on the data set Z, where we extracted used about 900 samples for the learning phase and about 400 samples for the test set.

Training Set	Same AS	Same Prefix	Same Month	Learning set errors ratio	Test set errors ratio
A				1.5	44.4
B				3.9	9.9
C				0.9	37.8
D				7.1	18.4
A		✓		1.5	7.1
B		✓		3.9	16.2
C		✓		0.9	14.1
D		✓		7.1	43.5
A	✓			1.5	25.8
B	✓			3.9	18.4
C	✓			0.9	11.1
D	✓			7.1	36.9
A	✓	✓		1.5	4.9
B	✓	✓		3.9	11.7
C	✓	✓		0.9	0.7
D	✓	✓		7.1	14.1
X	✓		✓	3.8	13.7
Y	✓		✓	4.4	12.1
Z	✓	✓	✓	12.2	21.7

Table 3: Learning Results

As we can see on Table 3 the proportion of error appearing in the tested sets is quite diverse and it is not easy to conclude anything from them. As we can see, it is possible to find a case with a very low percentage of errors in every scenarios, but as well as cases with a very high percentage of errors in the first 3 scenarios. The last 2 scenarios seem to conserve a low percentage rate of errors with all the training set used. To understand the high percentage of errors in some cases within the same scenario, we looked at the shape of a generated decision tree from a case leading to a high percentage of error compared to a generated decision tree from a case leading to a low percentage of error within the same scenario. For the first case of the first 2 scenarios indicated on Table 3 the respective error rate of the scenarios was 44.4% and 7.1% the built decision tree resulting from using the training set ’A’ was the following one.

```
Decision Tree:
     M2 <= 2 : W
     M2 > 2 : A
```

This very simple decision tree is also leading to a very low percentage of error in the first case of the forth scenario (4.9%) but still achieves quite bad results in the first (44%) and third (25.8%) scenario. We are probably in a case of overfitting here, where the automatic learning process produced a tree that was too specific to the injected learning set. Also the number of ’A’ samples in the data set A was quite low compared to the number of ’W’ samples (19 versus 114), which explains why C4.5 algorithm was not able to produce a decision tree that captured most of the cases but was too specific to a given situation. Some trees can be quite complex but not apply in all the scenarios that we designed, such as the one generated from the training set D:

```
Decision Tree:
     M4 > 0.2 : W
     M4 <= 0.2 :
     | M2 <= 2 :
     | | M2 <= 0.000918 : A
     | | M2 > 0.000918 : W
     | M2 > 2 :
     | | M3 <= 5 :
     | | | M5 <= 6 : A
     | | | M5 > 6 :
     | | | | M1 <= 2.80358 : W
```

```
| | | | M1 > 2.80358 :
| | | | | M3 > 4 : A
| | | | | M3 <= 4 :
| | | | | | M1 <= 2.93859 : W
| | | | | | M1 > 2.93859 : A
| | M3 > 5 :
| | | M3 <= 6 : W
| | | M3 > 6 :
| | | | M4 <= 0.1 : A
| | | | M4 > 0.1 : W
```

This complex tree is leading to up to 43% of errors in the second scenario. The reason of those bad results still needs to be investigated and generally speaking, it is difficult to find out why a decision tree is working in a situation and not in another. Nevertheless the Table 3 shows us some encouraging results. The results coming from the training set X and Y shows us that it is possible to build a decision tree that spans several peers at the same time. This result is encouraging as it would allow saving a lot of memory when implementing a shared decision tree per peer. The most remarkable results are coming from the training set Z, which is a real announced network prefix on the network (it is not a beacon). From these results one would possibly predict the pattern of announcements/ withdrawals of this particular prefix with an accuracy of 80%. This result is very encouraging and deserves some further investigations.

6. Conclusion

The path exploration phenomenon is inherently associated to the path-vector routing protocol of the Internet routing system, i.e., the Border Gateway Protocol (BGP). Henceforth, detecting and identifying path exploration sequences in BGP routing updates composed by announcement/withdrawal of Autonomous System (AS) Paths selected for the same IP address prefix, as well as anticipating the BGP route selection process to mitigate their effects is the fundamental problem addressed by this research study.

In this deliverable, we have thus described the research work performed to realize the task dedicated to the elaboration and experimentation of machine learning techniques to detect and identify BGP path exploration events so as to mitigate their effects by anticipating the decision of the BGP route selection process. First, the technical problem addressed has been formalized as a classification problem of sequential data. Then, the developed machine learning techniques, i.e. C4.5 decision trees, Hidden Markov Model (HMM), Conditional Random Fields (CRF) have been customized to provide appropriate solution to this inter-domain routing system problem. The proposed machine learning-based algorithms are evaluated by experimentation on representative Internet topologies. Our results show that elaborated learning models shall be considered to overcome limits of the base classification techniques for sequential data as observed in BGP routing system.

Performance evaluation results of the proposed machine learning techniques are detailed. Evaluation of the proposed machine learning algorithms has been performed by executing them in combination with a modified BGP routing engine running on the XORP routing platform. These experiments have been conducted using realistic AS topologies, following several properties of today's Internet topology such as smooth hierarchical structure, power-law node degree distribution, strong clustering coefficient as well as a constant average AS_Path length. Our evaluation demonstrates that it is possible to integrate a machine learning technique in a BGP router; however, the benefits of using the C4.5 algorithm to reduce the BGP churn are not proven. Henceforth, additional experiments are ongoing to refine these results using more advanced learning techniques proposed in this document.

References

[1] V.Lefèvre, *Applying Machine Learning Techniques to the BGP Path Exploration Process*, Master Thesis, UCL, June 2009.

[2] J.R.Quinlan, *Induction of Decision Trees*, Machine Learning, vol.1, no.1, pp.81-106. 1986.

[3] J.R.Quinlan. *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1993.

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

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

[6] V.Levenstein, *Binary Codes Capable of Correcting Deletions, Insertions, and Reversals*. In Soviet Physics Doklady, 10(8), pp. 707-710. 1966.

[7] S.Deshpande, and B.Sikdar, *On the Impact of Route Processing and MRAI Timers on BGP Convergence Times*, Proceedings of IEEE Globecom 2004, Dallas (Tx), United States, November 2004.

[8] C.Labovitz, A.Ahuja, A.Bose, and F.Jahanian, Delayed Internet Routing Convergence, IEEE/ACM Transactions on Networking, vol.9, no.1, pp.293-306, June 2001.

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

[10] A.Elmokash, A.Kyalbein and, C.Dovrolis, *On the Scalability of BGP: the Roles of Topology Growth and Update Rate-Limiting*. Proceedings of ACM CoNEXT'08, Madrid, Spain, December 2008.

[11] J.Qiu, *SimBGP: Simple BGP Simulator*, http://www.bgpvista.com/simbgp.php

[12] A.V.Aho, J.E.Hopcroft, and J.D.Ullman, *Data Structures and Algorithms*, Addison Wesley, 1983.

[13] L.E.Baum and, T.Petrie, *Statistical Inference for Probabilistic Functions of Finite State Markov Chains*. In Annals of Mathematical Statistics, vol.37, no.6, pp. 1554-1563, 1966.

[14] A.J.Viterbi, *Error bounds for convolutional codes and an asymptotically optimal decoding algorithm*, IEEE Trans. Information Theory, vol.IT-13, pp. 260-269, April 1967.

[15] G.D.Forney, *The Viterbi algorithm*, Proc. IEEE, vol.l, no.61, pp.268-278, March 1973.

[16] Y.LeCun, L.Bottou, Y.Bengio, and P.Haffner. *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, vol.86, no.11, pp.2278-2324, 1998.

[17] *C45 tutorial and original implementation*. Available at http://www2.cs.uregina.ca/~dbd/cs831/notes/ml/dtrees/c4.5/tutorial.html.

[18] L.R.Rabiner, *A tutorial on Hidden Markov Models and selected applications in speech recognition*. Proceedings of the IEEE, 77(2):257–286, 1989.

[19] J.Lafferty, A.McCallum, and F.Pereira, *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*, Int. Conf. Machine Learning, San Francisco, CA, 2001. Morgan Kaufmann.