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

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D4.3

Experimental evaluation of the machine learning engine

Project description
Project acronym: **ECODE** Project full title: **Experimental Cognitive Distributed Engine** Grant Agreement no.: **223936**
Document Properties
Title: **Experimental evaluation of the machine learning engine** Responsible: **IBBT** Editor(s): **Wouter Tavernier (IBBT)** Dissemination level: **Public (PU)** Date of preparation: **Nov. 2011** **Version: 0.1**

List of authors

ALB	Dimitri Papadimitriou
IBBT	Bart Puype, Steven Latre, Wim Van de Meerssche, Dirk Deschrijver, Wouter Tavernier
INRIA	Chadi Barakat and Amir Krifa
LAAS	Pedro Casas Hernandez, Johan Mazel, Philippe Owezarski
UCL	Damien Saucez, Benoit Donnet, Olivier Bonaventure
ULg	Guy Leduc

Executive summary

This deliverable (D4.3) reports on the results obtained from the experimental evaluation of the designs specified in deliverable D2.3 from WP2. The structure of this document is split into several use cases (7), resulting in a specific chapter for every case. These cases are representative of i) critical functions (intrusions/anomalies detection, accountability, etc.) that are either not easily achievable or not achievable at all by classical means or ii) new functionality that would otherwise not be realizable.

The first use case (a1) on adaptive sampling tackles the scalability issue of traffic measurement due to high speed networks, rapidly changing network conditions and the problem of missing meaningful information in the related network traffic. In this context, an adaptive system is evaluated combining different existing sampling primitives in order to support a large spectrum of monitoring tasks while providing the best possible accuracy. The system coordinates responsibilities between the different monitors and shares resources between the different sampling primitives. Experimental Results prove the ability of the system to keep the resulting overhead around a target value. Compared to application-specific solutions, the system shows its advantages in providing more accurate results.

The detection of distributed intrusions, attacks, and anomalies has been investigated in the context of the use case (a3). As documented in deliverable D2.3, an unsupervised machine-learning based system has been designed. The resulting NEWNADA system is composed of three different modules: (i) a monitoring Multi-Resolution Change-Detection module, (ii) an Unsupervised Machine-Learning based Analysis module, and (iii) a Characterization module. The designed system is evaluated using real traffic traces containing different types and patterns of network attacks, including DDoS, worms, and buffer-overflow attacks. By comparison against three previously used approaches for unsupervised detection of network attacks and anomalies (namely, DBSCAN-based, k-means-based, and PCA-based outliers detection), the evaluation results obtained show strong evidences that the learning method underlying the NEWNADA system is able to achieve high accuracy, even for attacks with unknown signatures, at a reasonable computational cost.

A path availability and performance service (IDIPS) is proposed in use case (b1). Given a list of source and destination addresses, and a ranking criterion, the IDIPS server determines all the possible <source, destination> pairs and computes the cost of each path according to the specified ranking criterion requested by the client. The corresponding chapter in this deliverable evaluates the impact of the IDIPS service in the context of a XORP implementation for its accuracy, stability and scalability properties.

The second part of use case (b1) investigates the idea of using a Network Coordinate System (NCS) to reduce the number of measurements needed between a given number of nodes to estimate valuable network metrics (e.g., delays or available bandwidths) between all pairs of nodes. The proposed DMFSGD-scheme uses Machine Learning, namely Stochastic Gradient

Descent (SGD), to find a low-rank approximation of the *n*x*n* distance matrix in a distributed way. DMFSGD is tested on 3 real (measured) round-trip-times datasets and on one dynamic dataset of available bandwidths. Its precision outperforms Vivaldi's, which is the reference NCS in the field. Moreover it has no coordinate drift problem, and can be used to estimate both delay and bandwidth values (regression), as well as to estimate delay or bandwidth classes (classification). In addition a novel method to find one-hop routing shortcuts between given source-destination paths is evaluated. Results show that this method only needs a limited number of additional measurements for returning candidate relay nodes returning significant delay gains in the resulting paths.

In use case (b2), the link failure history is used to identify Shared Risk Groups (SRGs) from network element. Through clustering and data-mining of failure occurrences, a predictive model is built, allowing inference of SRGs upon the detection of a first network element failure. The improved link failure detection times resulting from the algorithm are evaluated through measurements and compared to analytical estimations. The detection time decreases as SRGs become larger, meaning that SRGs having a larger impact on the network performance have a better chance at seeing (greater) improvement from the inference procedure. In addition, scalability results for SRG table size and computational complexity of the algorithm are characterized.

The performance of using an automated way of configuring Loop-Free Alternate (LFA) entries in a routing table is evaluated in the context of the second component of use case related to network resiliency (b2). The use of Loop-Free Alternates avoids that large recovery times are needed for routing tables before network traffic affected by an adjacent failure is recovered. Performance evaluations show that the proposed technique, named ALFA, is able to cover almost 100 percent of potential link failures, while providing an automated way of configuring these entries.

The use case (b3) on profile-based accountability involves machine learning techniques in order to detect (and the possibly penalize): i) TCP receivers who deliberately ignore TCP congestion feedback to willingly achieve an advantage in throughput (misbehaving receivers), and ii) TCP sender who react differently to TCP congestion feedback than is generally expected (unresponsive senders). Both the accuracy of the detection algorithm and the achieved fairness gain of the overall accountability mechanism are evaluated. Several approaches are evaluated to find a good attribute set that (i) is able to distinguish between different TCP stacks known to have different levels of aggressiveness and thus lead to significant throughput differences (for example, an Additive Increase Multiplicative Decrease (AIMD) TCP stack such as CUBIC is designed to be less aggressive in increasing the throughput than a Multiplicative Increase Multiplicative Decrease (MIMD) TCP stack such as Scalable TCP), and (ii) does not distinguish between these stacks when there is not a difference in throughput or if the throughput difference is not due to the difference in aggressiveness but due to other factors such as the existence of a bottleneck. While tens of attribute sets were considered, this deliverable focuses on the best 4 candidates found so far: a mapping on a statistical distribution, a distribution of increments, the auto-covariance and a translation to the frequency domain. It is illustrated that these attribute sets show promising results in achieving the desired accuracy of detection and resulting fairness in throughput. However, at the same time the techniques are still prone to parameter fluctuations.

Overall, the experimentation of the learning modules by means of the machine learning engine has been successfully conducted by means of the ECODE Unified Architecture (EUA)

proposed in deliverable D2.2. Indeed, the experimental results obtained show that such a platform can host different learning modules performing different learning tasks.

Contents

Acronyms

ADC Approximation Detection Criterion

AIMD Additive Increase, Multiplicative Decrease

ALFA Automated Loop-Free Alternates

AS Autonomous System

AQM Active Queue Management

CDF Cumulative Distribution Function

CWR Congestion Window Reduced

DDoS Distributed Denial-of-Service

DMFSG Decentralized Matrix Factorization-based on Stochastic Gradient Descent

DoS Denial-of-Service

ECE ECN Echo

ECN Early Congestion Notification

EDC Estimation Detection Criterion

EUA ECODE Unified Architecture

FIB Forwarding Information Base

FRR Fast-ReRoute

LFN Loop-Free Node

HDC Hybrid Detection Criterion

IDIPS ISP-Driven Informed Path Selection

LFA Loop-Free Alternate

LOF Local Outlier Factor

LSA Link State Advertisement

MDS MultiDimensional Scaling

MIMD Multiplicative Increase, Multiplicative Decrease

NADA Network Anomaly Detection Algorithm

NCS Network Coordinate System

OSPF Open Shortest Path First

PCA Principal Component Analysis

RED Random Early Detection

RIB Routing Information Base

ROC Receiver Operating Characteristic

SGD Stochastic Gradient Descent

SRG Shared Risk Group

SVD Singular Value Decomposition

TIV Triangle Inequality Violation

XORP eXtensible Open Routing Platform

XRL XORP Resource Locator

List of Figures

List of Tables

Chapter 1

Introduction

The overall goal of the ECODE project is the application of machine learning techniques in different network contexts leading to the design and experimentation of a distributed cognitive engine. WP2 took in charge the specification of the cognitive network and system architecture as well as the design of the cognitive engine. WP3 has detailed the experimental uses cases and related network contexts on which the project focuses together with corresponding machine learning methods and techniques they involve. The task of WP4 is to experiment the resulting design for the identified use cases. This document (D4.3) reports on the experimental evaluation of the machine learning engine by means of representative use cases and execution scenarios.

- (a1) Adaptive sampling
- (a3) Cooperative intrusion and attack / anomaly detection
- (b1) Path availability
  - ISP-Driven Informed Path Selection (IDIPS)
  - Path availability and coordinate systems
- (b2) Network resilience
  - OSPF SRG inference
  - Automated Learning of Loop-Free Alternate paths for fast-rerouting
- (b3) Profile-based accountability

The low-level design specification of machine learning functionality in order to successfully handle the above objectives has been documented in deliverable D2.3. The resulting architecture consists of the specification of interfaces and components that would allow implementation of interoperable parts by third-party developers. The driver among the resulting specifications was the XORP open source routing platform. XORP provides a fully featured control plane platform that implements routing protocols and a unified platform to configure them. XORP's modular architecture allows rapid introduction of new protocols, features and functionality. The developed prototypes are implemented on top of the ECODE Unified Architecture (EUA) whose design is specified in deliverable D2.2. The latter is a distributed XORP extension.

The rest of this document devotes a chapter to the evaluation of every mentioned use case. At the end of the document some the resulting expertise coming out of these evaluations is shortly described in the concluding chapter 9.

Chapter 2

Adaptive sampling

2.1 Introduction

In Deliverable D2.3, we present the architecture of a cognitive centralized system able to achieve concurrent monitoring tasks while offering the best possible accuracy at limited monitoring overhead. The proposed system deals with multiple monitoring objectives (e.g., flow size estimation, heavy hitter detection, flow counting) and determine how to combine independent measurements collected across the network using different sampling primitives and different monitoring tools. The system also coordinates the responsibilities across the different monitors and shares efficiently the resources between the different sampling primitives. The objective function of the system is to maximize the global measurement accuracy under resource constraints on the sampling rate and the volume of traffic collected, i.e., measurement overhead, from routers. A major property of the system is to automatically adapt to traffic variations and network conditions.

An adaptive sampling system is scalable by definition in terms of traffic growth. Our system will ensure the sampling rates inside routers are optimally chosen, both spatially and temporally, so that the overhead of measurements is limited. Our system is also scalable in terms of number of monitoring tasks and network topology. A new measurement task is simply a new function to calculate over the traffic and to add to the objective function to optimize (linear increase of complexity). As for the scalability in terms of topology, it is granted given the distributed nature of the measurements. Only the processing is centralized, each router is instructed about the sampling rate to use, and it then performs measurements and aggregation in flows, before sending reports about them to the central unit. To determine the scalability of the system, we thus observe the accuracy of the sampled measurements. We also determine how accurately the system reacts to network and traffic changes (i.e., the timing).

2.2 Methodology

In order to evaluate the performance of our system, we developed an associated experimental platform Monlab [66]. The main features of this platform are: *(i)* it is fed by real traffic captured on a transit link then spread and played over an emulated network topology, *(ii)* it includes real NetFlow-like tool for traffic monitoring on all router interfaces of the emulated topology, and *(iii)* it implements the central processing unit.

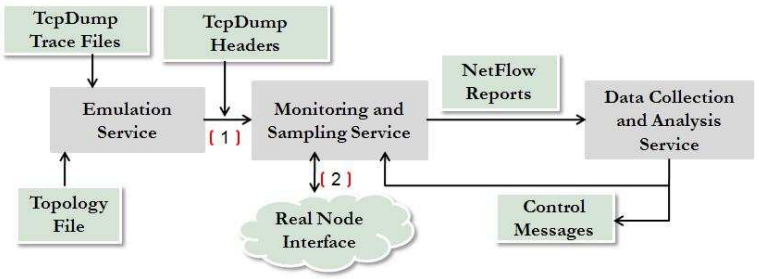


Figure 2.1: Experimental platform

As shown in Figure 2.1, our experimental platform comprises three services: *(i)* the traffic emulation service responsible for the generation of the emulated traffic between network routers (that can be either virtual nodes connected by virtual links, or real routers connected by real links), *(ii)* the traffic monitoring and sampling service, which implements packet sampling and flow monitoring a la NetFlow on each router interface, and *(iii)* the data collection and analysis service, a centralized service that collects NetFlow records, correlates them to better estimate network traffic, and then runs our adaptive algorithm to decide on which sampling rates to update. The third service mainly relies on the Flowd tool for the SoftFlowd package [8], an open source free software capable of NetFlow measurements in high speed networks.

2.2.1 Experimental Platform

SoftFlowd requires network traffic in the TcpDump format. Unfortunately, obtaining real traffic data from an entire backbone network is a hard issue. To cope with, we proceed in the following way. We first seek unsampled packet traces collected on high speed transit links. We consider for this study the ones coming from the Japanese MAWI project [4]. We parse the traces for the IP prefixes, and we dispatch them over the Autonomous Systems (AS) connected to the edge routers of the emulated topology. The dispatching is performed randomly according to some predefined weights that determine the importance of each stub AS. Our system allows to define the length of the prefix as a function of the granularity of the dispatching we want to achieve. For this deliverable, we consider the /16 prefix as the unit for IP address assignment to ASes. Once addresses are allocated, the packets in the TcpDump trace are split accordingly between the different ASes connected to the emulated topology. Shortest routes are calculated. Then packets in the TcpDump trace are associated to the different monitors over their respective paths across the network with the correct timestamps derived from the trace. SoftFlowd samples then packets, form flows and send them back to the central collector. This sampling and monitoring is done in parallel on all network router interfaces.

2.2.2 Scenario Description

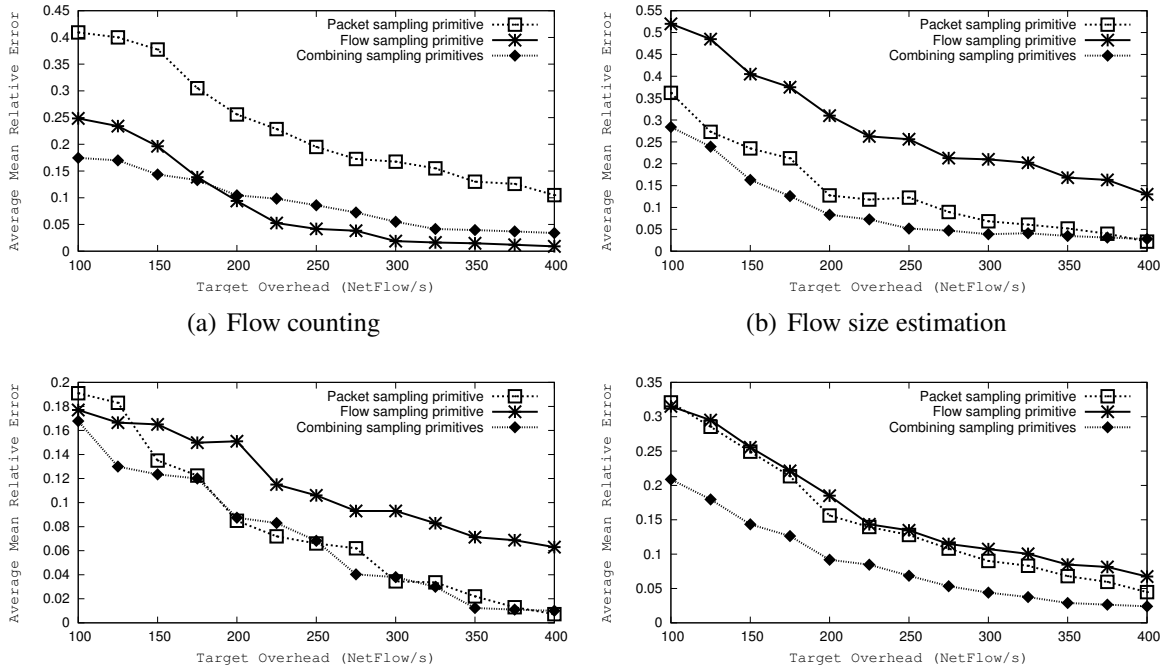
Our platform requires the definition of a network topology over which it dispatches and replays real traffic. We conduct our experiments on topologies similar to Geant [1]. We set the weights of AS's needed for traffic dispatching according to the size of stub AS's in Geant and we make sure these weights sum to 1. An AS weighted with value w is assigned $100.w\%$ of the prefixes available in the trace and will see its traffic (incoming or outgoing) being around $100.w\%$ of the

total traffic, both at the flow and packet levels (random prefix allocation). Once topology and weights are set, TcpDump traces coming from [4] are replayed. Each platform service runs on a dedicated machine. In order to initialize the different estimators, we run a first computation without sampling on target applications: flow counting, flow size estimation and heavy hitter detection. The objective is to minimize the weighted normalized estimation error.

2.3 Scalability

The concept of adaptive sampling is scalable by definition as the system automatically adapts itself to the resource constraints. As stated before, the system is also scalable in terms of number of monitoring tasks and network topology. The fundamental question is thus not to determine the scalability itself but rather the accuracy with which the system can adapt the configuration of routers monitors to the predefined resource limitations. For this purpose, we show in this section the practical benefits of deploying our system by comparing it to application-specific sampling systems. Second, we evaluate the efficiency of our adaptive solution in limiting the overhead and minimizing the estimation error for the considered monitoring applications. We refer the reader to deliverable D2.3 for technical details about the system architecture and its parameters.

For this experiment, we set the timer d for updating sampling rates to 5 minutes, the time scale β to 3600s, the minimum possible sampling rate SR_{min} to 0.0005 and the maximum possible one SR_{max} to 1. The $\mathscr{TO}$ is set to 200 NetFlow-records/s. We compare the performance of the Flow counting (FC), Flow Size (FS), and Heavy Hitter (HH) monitoring applications obtained with the proposed joint method and compare its performance with the two application-specific primitives when used separately, i.e., packet sampling and flow sampling primitive. In Figure 2.2, we plot the average mean relative error for three specific monitoring applications. We plot in Figure 2.2(d) the global accuracy defined as the global weighted accuracy. This figure shows that our solution provides the best global accuracy for the different monitoring applications. It combines measurements of the different monitoring primitives to improve the accuracy of the global estimator. As we can see in this figure, in order to reach similar global accuracy to our joint method using application-specific primitive, one has to use a $\mathscr{TO}$ value larger than 150% of the value used by the method we propose. In fact, each primitive focuses on achieving a specific application. Therefore, while the use of a single primitive allows improving results for its specific monitoring application, it provides inaccurate results for the other monitoring applications. These results are illustrated in figures 2.2(a),2.2(b) and 2.2(c). We observe that each primitive provides accurate measurements for its specific application and less accurate results for the other applications. For instance, packet sampling primitive improves accuracy of FS (Flow Size) and HH (Heavy Hitter) applications while it provides a large estimation error for FC (Flow Counting) application. On the other hand, flow sampling primitive improves results of FC application and provides less accurate results for FS and HH applications. We notice that our solution based on combining results of the different monitoring primitives improves accuracy of the different applications especially when the value of $\mathscr{TO}$ is small. In this case, sampling rate values are low and each single primitive provides inaccurate results. We can thus combine their measurements in order to improve the global accuracy. Hence, using our solution based on combining different sampling primitive measurements, allows not only optimizing resource consumption but also improving the global accuracy as well as the accuracy of each



(a) Flow counting

(b) Flow size estimation

(c) Heavy hitter detection

(d) Global accuracy

Figure 2.2: Average mean relative error vs. Target overhead ($\mathscr{TO}$) for three applications: Our approach vs. two application-specific approaches.

specific monitoring application. Furthermore, these figures also show the large impact of the monitoring constraint ($\mathscr{TO}$) on measurement accuracies. We observe in particular the clear reduction of estimation errors when increasing the value of $\mathscr{TO}$.

2.4 Timing

In this section, we evaluate the impact of the weights used in the global utility function on the behavior of our system. The parameters of experiments are set as in Sec. 2.3. We run three scenarios while changing each time the assigned weight value to each monitoring application. We measure periodically the mean relative error. Then, we consider the average over all these values measured during the experiment. Table 2.1 and 2.2 present a summary of experimental results for a selection of scenarios. We notice that assigned weights have an impact on the behavior of our system. We observe from Table 2.1 that increasing the weight value assigned to a given monitoring application allows reducing the average mean relative error (AMRE) of its measurement.

Consider the following experiments: in a first scenario (SC1), the operator assigns equal weights to the different tasks (0.33). In order to validate our system, we run again the same experiment in SC2 while increasing the weight assigned to the FC application from 0.33 to 0.5. From Table 2.1, we see that the error of FC application is reduced from 0.1045 to 0.0738. In fact, by increasing the weight assigned to FC application the system finds automatically the new best configuration that minimizes the global error while meeting the monitoring constraints ($\mathscr{TO}$). In Table 2.2 we observe that the new average sampling rate value for the flow sampling

Table 2.1: Summary of assigned weights and experimental results for a selection of scenarios

Scenario	Flow counting		Flow size estimation		Heavy hitter detection		Global accuracy
	assigned weight	AMRE	assigned weight	AMRE	assigned weight	AMRE	
SC1	0.333	0.1045	0.333	0.0834	0.333	0.06745	0.08511
SC2	0.5	0.0738	0.25	0.164	0.25	0.0865	0.09952
SC3	0.5	0.0889	0.5	0.114	-	-	0.1014

Table 2.2: Average sampling rate values and reported NetFlow records for a selection of scenarios

Scenario	Flow Sampling		Packet Sampling	
	Average sampling rate value	Percentage of reported NetFlow records	Average sampling rate value	Percentage of reported NetFlow records
SC1	0.016	39.65%	0.314	60.35%
SC2	0.0287	62.84%	0.197	37.16%
SC3	0.03613	58.23%	0.023	41.77%

primitive increases from 0.016 to 0.0287 together with an increase of the sampled flows percentage (reported Netflow record for flow sampling primitive) from $39,65\%$ to $62,84\%$. Hence, for a *set of applications with their corresponding weights*, our system finds the best monitors configuration that optimizes the global accuracy while meeting monitoring constraints. This result is illustrated in SC3. We assign the same weight (0.5) to FC application and we use only two applications instead of three. We observe in Table 2.1 that the system finds a new optimal configuration for this new set of applications and their assigned weights.

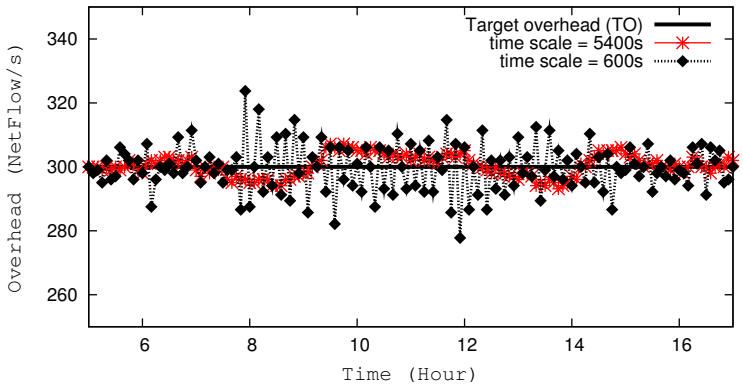


Figure 2.3: Resulting overhead vs. time using two time scales to track traffic variations

In order to evaluate the performance of the overhead prediction method, we plot in figure 2.3 the evolution of the measured overhead (exported NetFlow records) over time. We observe that the system maintains the overhead around $\mathscr{TO}$ for the two time scale values. In fact, the system tries to profit from the available resources in order to provide the best possible accuracy. However, using a small time scale ($\tau = 600s$) leads to an oscillating behavior of the overhead since the system tracks more details and traffic variations. Instead, tracking changes on a large scale ($\tau = 5400s$) leads to a stable behavior of the overhead as the system avoids some details and traffic variations specific to one observation period.

2.5 Conclusion

We have presented an adaptive system that combines different existing sampling primitives in order to support a large spectrum of monitoring tasks while providing the best possible accuracy. Our system coordinates responsibilities between the different monitors and shares resources between the different sampling primitives. Experimental Results proved the ability of our system to keep the resulting overhead around a target value. Compared to application-specific solutions, our system has shown its advantages in providing more accurate results especially for low values of $\mathscr{TO}$. Our system is practical and provides a flexible optimization method based on overhead prediction that reconfigures monitors according to monitoring applications requirements and network conditions.

Chapter 3

Cooperative intrusion and attack / anomaly detection

3.1 Introduction

The *Unsupervised Network Anomaly Detection Algorithm* (NEWNADA) is an unsupervised machine-learning based system conceived to meet the objective of automatic detection and characterization of intrusions and attacks/anomalies within ECODE. NEWNADA comprises three different modules: (i) a monitoring Multi-Resolution Change-Detection module, (ii) an Unsupervised Machine-Learning based Analysis module, and (iii) a Characterization module. Deliverable D2.3 describes the architecture and the interaction between these three modules as well as the implementation of these modules within the ECODE Unified Architecture (EUA). This chapter provides some evaluations of NEWNADA with real traffic containing different types of network attacks, including DDoS, worms, and buffer-overflow attacks.

NEWNADA is a traffic analysis system that permits to identify previously unknown anomalous traffic behaviors without relying on signatures or calibration. The system permits to rank the degree of abnormality of a set of traffic flows going through a monitored network link. In addition, NEWNADA provides a summary of the most relevant traffic descriptors that characterize the top-ranked flows in the form of anomalous traffic signatures. Such signatures permit to automatically separate the interesting traffic events from the normal-operation traffic, thus significantly simplifying network monitoring tasks. The information provided by NEWNADA permits not only to detect and identify anomalous traffic flows, but also to rapidly understand the nature of the anomaly and therefore to rapidly apply accurate and adapted counter-measures.

In this Chapter, we evaluate the performance of NEWNADA's implementation, using both real and artificial network traffic. The methodology and the testbed used for the evaluation are described in Section 3.2. Section 3.3 evaluates NEWNADA by replaying real traffic traces from the WIDE network [29]. The WIDE operational network provides interconnection between different research institutions in Japan, as well as connection to different commercial ISPs and universities in the US. Traffic consists of 15 minutes-long raw packet traces daily collected for the last ten years. The traces we shall work with consist of traffic from one of the transpacific links between Japan and the United States. WIDE traces are not labeled, but some previous work on anomaly detection has been performed using them [35, 41]. In particular, [41] detects network attacks using a signature-based approach while [35] detects both attacks and anomalous flows using non-Gaussian modeling. We shall therefore refer to the combination of results

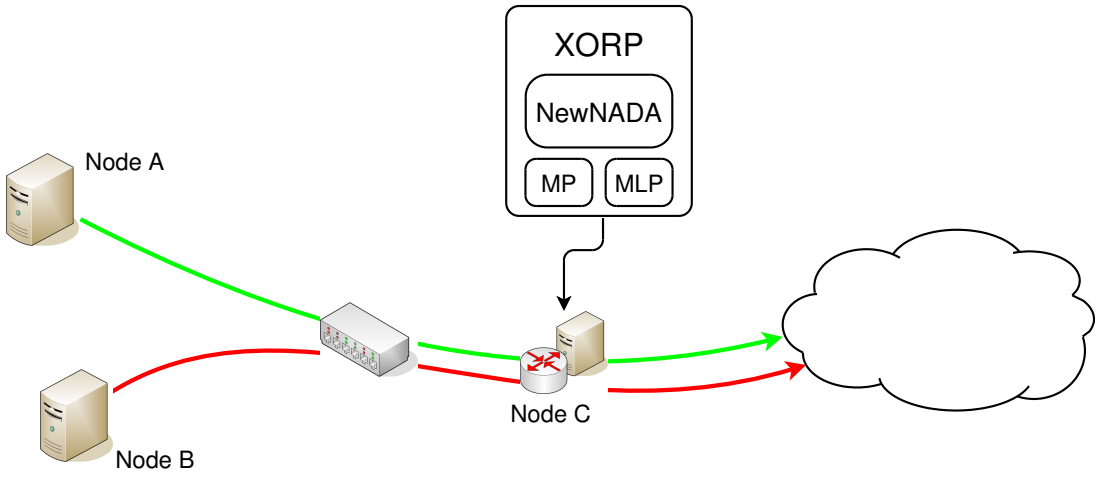


Figure 3.1: Testbed for evaluation of NEWNADA in XORP.

obtained in both papers as our *ground truth* for WIDE traffic. In this section, we also evaluate the NEWNADA's performance for the detection of flooding attacks in replayed traffic traces obtained from the METROSEC project [5]. These traces consist of real traffic collected on the French RENATER network, containing simulated attacks performed with well-known DDoS attack tools. Traces were collected between year 2004 and 2006, and contain DDoS attacks that range from very low intensity (i.e., less than 4% of the overall traffic volume) to massive attacks (i.e., more than 80% of the overall traffic volume). In addition, we compare the performance of NEWNADA against some previous methods for unsupervised anomaly detection. In Section 3.4 we evaluate the ability of NEWNADA to detect network attacks in the well-known and widely used KDD99 network attacks dataset [7]. In this case, the analysis is performed off-line because the main interest of this evaluation is to show that NEWNADA can detect other kinds of attacks when considering other types of traffic descriptors. In this section, we also compare NEWNADA performance with those obtained by an extensively investigated signatures-based NIDS based on decision trees. Decision trees permit to construct comprehensive signatures for network attacks in the form of multiple filtering-rules, using a graph structure. This comparison permits to prove the advantage of using Unsupervised Analysis techniques like those used in NEWNADA when discovering unknown attacks. Finally, Section 3.5 evaluates the computational time of NEWNADA by taking into account the possibility of parallelization of the Unsupervised Analysis module in a future implementation.

3.2 Methodology

In order to evaluate the implementation of NEWNADA in XORP, we replay real traffic traces from WIDE and METROSEC in the simple testbed depicted in Figure 3.1, using TCPReplay [6]. TCPReplay is a suite of libpcap tools that permit among others to transmit previously captured traffic traces. In this topology, host labeled as `Node A` replays traffic traces free of anomalies as background traffic, sending packets to host labeled as `Node C` with the original IP addresses of the trace. In order to replay these attacks, host labeled as `Node B` replays a trace built from only those traffic packets composing the real attacks, previously extracted by hand from the original traces.

Type of Attack	Class	Resolution	Generated Signature
SYN NetScan	1-to-N	IPdst/24	$(\text{nSrcs} == 1) \wedge (\text{nDsts} > 100) \wedge (\text{nSYN/nPkts} > 0.75)$
SYN DDoS	N-to-1	IPsrc/32	$(\text{nDsts} == 1) \wedge (\text{nSYN/nPkts} > 0.7) \wedge (\text{nPkts/sec} > 10) \wedge (\text{nSrcs} > 20)$
ICMP DoS	1-to-1	IPdst/32	$(\text{nICMP/nPkts} > 0.65) \wedge (\text{nPkts/sec} > 50)$

Table 3.1: Signatures generated by NEWNADA in the detection of a SYN Network SCAN attack, a SYN DDoS attack, and an ICMP flooding attack.

The NEWNADA modules relies on the Monitoring Point (MP) and the Machine Learning Process (MLP) that are part of the EUA XORP platform. These processes run in a host labeled as `Node C` as indicated in Figure 3.1, which acts as a gateway for all the incoming traffic directed towards the original IP addresses of the traces.

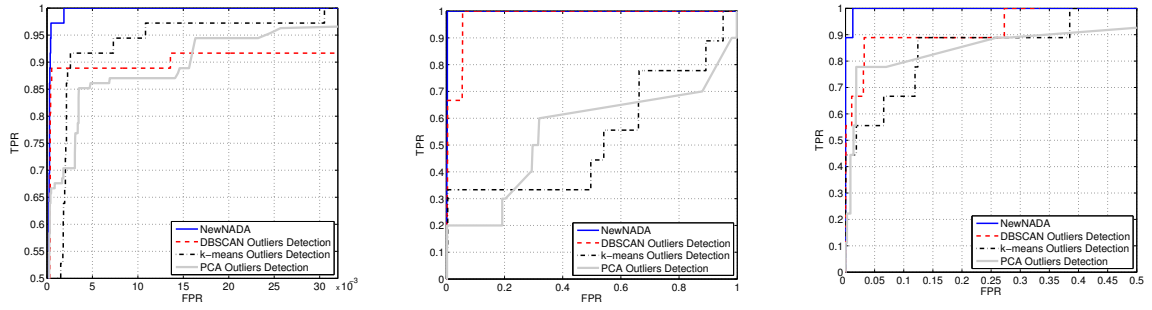
3.3 Detecting Attacks in WIDE and METROSEC real traffic

We begin by detecting and characterizing different attacks in WIDE through the EA4C and the EA4O algorithms. The selected set of *m* features used in these evaluations include the number of source/destination IP addresses and ports (nSrcs, nDsts, nSrcPorts, nDstPorts), the ratio of number of sources to number of destinations, the packet rate (nPkts/sec), the average packet size (avgPktsSize), and the fraction of ICMP and SYN packets (nICMP/nPkts, nSYN/nPkts.

The first case study corresponds to a distributed SYN network scan directed to many victim hosts under the same /16 destination network. The trace consists of traffic captured the 01/04/01. Incoming traffic is aggregated in IPdst/24 macro-flows, thus the attack is detected as a small-size cluster. Table 3.1, line 1, presents the signature produced for this attack. The selected filtering rules involve the number of IP sources and destinations, and the fraction of SYN packets. The signature makes perfect sense since the network scan uses SYN packets from a single attacking host to a large number of victims. Further analysis of the traffic that compose each of the flagged macro-flows reveals different IPdst/32 flows of SYN packets with the same origin IP address corresponding to the attacker. The signature permits to correctly identify all the flows of the attack. The main advantage of the unsupervised approach relies on the fact that this new signature has been produced without any previous information about the attack or the corresponding traffic pattern.

The next two case-studies correspond to flooding attacks. For practical issues, traffic corresponds to different combined traces (14/10/03, 13/04/04, and 23/05/06). Table 3.1, line 2, shows the different combined filtering rules obtained in the detection of a SYN DDoS attack. Traffic is aggregated in IPsrc/32 macro-flows. The obtained signature clearly confirms the nature of a SYN DDoS attack; this signature is able to correctly isolate the most aggressive attacking hosts of the DDoS, namely those with highest packet rate. Table 3.1, line 3, finally shows the signature obtained in the detection of an ICMP flooding DoS attack. Traffic is aggregated using resolution IPdst/32; thus, the attack is detected as an outlier rather than as a small cluster. The signature includes typical characteristics of this attack, such as a high packet rate of exclusively ICMP packets from the same source host.

Let us now focus the attention on the EA4O algorithm to detect anomalies and attacks as outlying macro-flows. Figure 3.2 depicts the True Positives Rate (TPR) as a function of the



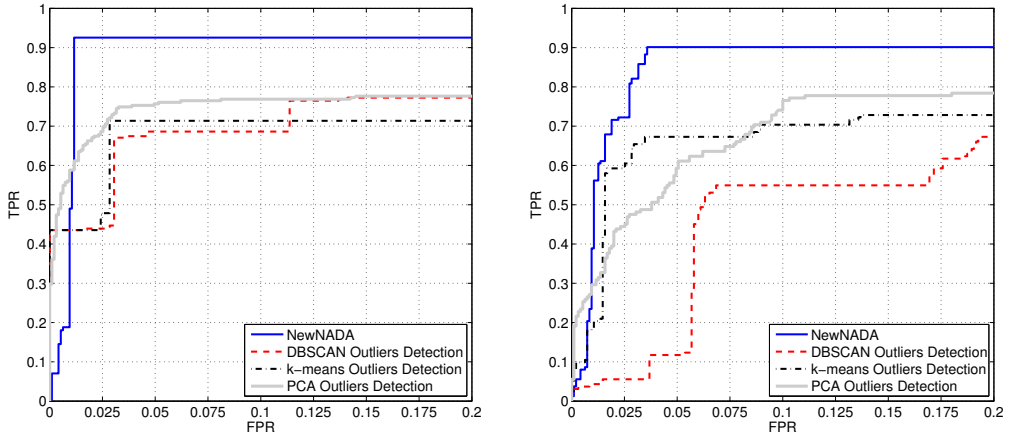
(a) WIDE, IPsrc/32. (b) WIDE, IPdst/32. (c) METROSEC, IPdst/32.

Figure 3.2: True Positives Rate vs False Alarms in WIDE and METROSection

False Positives Rates (FTR) in the detection of different attacks in WIDE and METROSEC Figure 3.2.(a) corresponds to the detection of 36 anomalies in WIDE traffic, using IPsrc/32 macro-flow resolution. These anomalies include network and port scans, worm scanning activities (Sasser and Dabber variants), and some anomalous flows consisting on very high volumes of NNTP traffic. Figure 3.2.(b) also corresponds to anomalies in WIDE traffic, but using IPdst/32 resolution. In this case, there are 9 anomalies, including different kinds of flooding DoS/DDoS attacks. Figure 3.2.(c) corresponds to the detection of 9 DDoS attacks in the METROSEC dataset. From these figures, we observe that 5 attacks correspond to massive attacks (more than 70% of traffic), 1 to a high intensity attack (about 40%), 2 are low intensity attacks (about 10%), and 1 is a very-low intensity attack (about 4%). The detection is performed using traffic aggregated at IPdst/32 resolution. In the three evaluation scenarios, the Receiver Operating Characteristic (ROC) curve is obtained by comparing the sorted dissimilarities in D_{rank} to a variable detection threshold.

We compare the performance of NEWNADA against three common approaches for unsupervised detection of network attacks and anomalies: DBSCAN-based, k-means-based, and PCA-based outliers detection. The first two consist in applying either DBSCAN or k-means to the complete features' space $\underline{\mathrm{X}}$, identify the largest cluster $C_{\max}$, and compute the Mahalanobis distance of all the flows lying outside $C_{\max}$ to its centroid. The ROC is finally obtained by comparing the sorted distances to a variable detection threshold. These approaches are similar to those used in previous work [94, 38, 76]. In the PCA-based approach, PCA and the sub-space methods [69, 70] are applied to the complete matrix $\underline{\mathrm{X}}$, and the attacks are detected by comparing the residuals to a variable threshold. Both the k-means and the PCA-based approaches require fine tuning. In k-means, we repeat the clustering for different values of clusters k, and take the average results. In the case of PCA, we present the best performance obtained for each evaluation scenario.

The results obtained permit to illustrate the significant advantage of using the EA4O algorithm for outliers detection compared to current approaches. In particular, all approaches considered for comparison purposes generally fail to detect all the attacks with a reasonable false alarm rate. Both the DBSCAN-based and the k-means-based algorithms get confused by masking features when analyzing the complete space $\underline{\mathrm{X}}$. The PCA approach shows to be not sensitive enough to discriminate different kinds of attacks of very different intensities, using the same representation for normal-operation traffic.



(a) Training dataset (b) Testing Dataset

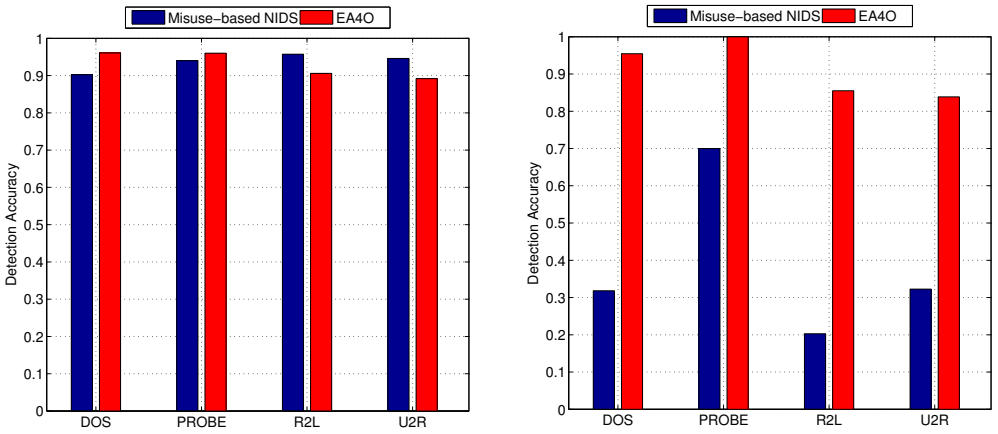
Figure 3.3: True Positives Rate vs False Alarms in KDD99.

3.4 Detecting Attacks in KDD99 artificial traffic

In this section we show that NEWNADA can detect other kinds of attacks when considering different features. In addition, we evidence the significant advantage of using Unsupervised Analysis techniques like those used in NEWNADA when discovering unknown attacks, comparing its performance against traditional signature-based detection systems; in particular, we use a signatures-based NIDS based on decision trees.

Contrary to previous evaluations, the analysis for KDD99 traffic is performed off-line without replaying any traffic traces, using the EA4O algorithm. The KDD99 dataset contains a wide variety of intrusions simulated in a military network environment. Traffic consists of packets aggregated into connections, each connection being a flow of TCP packets between a source and a destination IP address. Therefore, we shall not talk about aggregation and macro-flows in the following evaluations. Simulated attacks include DoS attacks, unauthorized access from a remote machine - R2L attacks (e.g., password guessing), unauthorized access to super-user privileges - U2R attacks (e.g., buffer overflows), and probing attacks (e.g., port scanning). Each connection or flow is described by a set of $m = 41$ features (e.g., number of bytes, TCP flags, failed remote-login attempts, etc.) and a label indicating either the name of the attack or if the flow corresponds to normal-operation traffic.

In order to compare the performance of NEWNADA against the decision-tree-based NIDS, we take two different data sub-sets. The first is used for training purpose and the second one for testing. The testing dataset corresponds to instances of different attacks that are not present in the training dataset; this dataset selection permits us to show the significant advantage of performing unsupervised detection when new attacks arise. DoS and probing attacks in KDD99 are represented by a large number of flows, in some cases even more flows than those corresponding to normal-operation traffic. While this was not an issue for NEWNADA when evaluated in previous section with real traffic traces, KDD99 flows provided at [7] are already pre-processed and can not be aggregated because the corresponding IP addresses are not available. To avoid this limitation in the already processed KDD99 flows, we have to select only a small fraction of flows for both DoS and probing attacks in our training and testing datasets. In any case, we have already proved that NEWNADA is able to detect both DoS and probing attacks in real



(a) Training dataset (b) Testing Dataset

Figure 3.4: NEWNADA vs Misuse-based NIDS in KDD99.

traffic traces when using macro-flows' aggregation. The training dataset has 950 normal flows and 255 attacks, while the testing dataset consists of 950 normal flows and 162 attacks.

Let us first evaluate the True Positives and False Positives rates obtained by NEWNADA in both training and testing datasets. Figure 3.3 depicts the corresponding ROC curves. Figure 3.3.(a) shows the results obtained by applying the EA4O algorithm to the training dataset, while figure 3.3.(b) shows the results obtained when analyzing the testing dataset. In both cases we can appreciate that NEWNADA is able to detect a large fraction of attacks (more than 90%) with very low false positive rates (less than 1% and 3.5% respectively). Figure 3.3 additionally compares the obtained detection performance against the three previous approaches used for unsupervised detection, i.e., DBSCAN-based, *k*-means-based, and PCA-based outliers detection. In both cases, we appreciate once again the outperforming ability of NEWNADA with respect to these approaches, which fail as before to detect as many attacks as EA4O with a reasonable false alarms rate.

Let us now compare NEWNADA against a largely studied misuse-based NIDS built through decision trees. We shall build a different decision-tree for each of the four different categories of attacks (DoS, probe, R2L, and U2R), using the training dataset and standard C4.5 decision trees [67, 74]. To train each of the trees for each different category of attacks, we consider that the flows belonging to the rest of the categories of attacks as well as the normal operation flows correspond to the "negative" class (there is no attack of the corresponding category). For example, let us suppose that we want to build a decision tree to detect R2L attacks; in that case, all the flows in the training dataset which belong to the R2L category belong to the "positive" class (there is a R2L attack), while the normal-operation flows as well as the DoS, probe, and U2R flows compose the negative class. In order to avoid over-fitting problems when training the decisions trees, we have decided to use early-stopping learning. For this reason, not all the attacks present in the training dataset are detected by the decision-trees-based NIDS.

Figure 3.4 presents the detection accuracy (number of correctly detected attacks) obtained either with NEWNADA or with the NIDS previously described in both the training and testing datasets. Results are individually presented for each of the four categories of attacks. As expected, results obtained by both systems are very similar in the training dataset: in the case of NEWNADA, we have already shown in figure 3.3 that more than 90% of the attacks can be

correctly detected; in the case of the NIDS, as can be expected, the system detects the attacks for which it has been designed. What it is interesting to appreciate is what happens with both systems when we try to detect unknown attacks. Figure 3.4.(b) illustrates the limitations of the NIDS to detect unknown attacks, and more importantly, the significant advantage of using the EA4O algorithm for detecting new previously unseen attacks. We refer to figure 3.5 to appreciate the detection accuracy obtained with both systems for the different attacks on each of the four different attacks categories available in the KDD99 dataset.

3.5 Computational Time and Parallelization

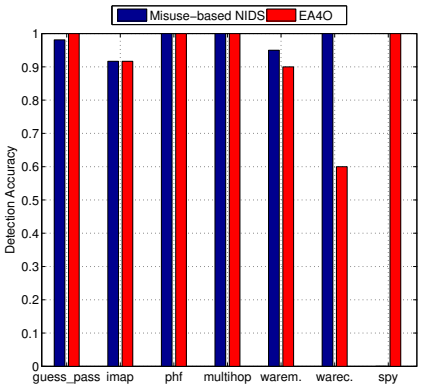
To complete our evaluation, we analyze is the Computational Time (CT) of NEWNADA. The clustering algorithm of the Unsupervised Analysis module performs multiple clusterings in $N = m(m-1)/2$ low-dimensional sub-spaces $X_i \subset X$. This multiple computation imposes scalability issues for on-line detection of attacks in very-high-speed networks. Two key features of the algorithm are exploited to reduce scalability problems in number of features m and the number of aggregated flows n to analyze. Firstly, clustering is performed in very low dimensional sub-spaces, $X_i \in R^2$, which is faster than clustering in high-dimensional spaces [59]. Secondly, each sub-space can be clustered independently of the other sub-spaces, which is perfectly adapted for parallel computing architectures. Parallelization can be achieved in different ways: using a single multi-processor and multi-core machine, using network processor cards and/or GPU (Graphic Processor Unit) capabilities, using a distributed group of machines, or combining these techniques. We shall use the term "slice" as a reference to a single computational entity. The current implementation of NEWNADA does not use parallelization, and each sub-space is sequentially analyzed, one after the other. However, we show that parallelization can significantly improve the CT, and therefore increases the volume of traffic that can be monitored in an on-line basis.

Figure 3.6 depicts the CT of NEWNADA's multi-clustering algorithm, both (a) as a function of the number of features m used to describe traffic flows and (b) as a function of the number of flows n to analyze. Figure 3.6.(a) compares the CT obtained when clustering the complete feature space X, referred to as $\mathrm{CT}(X)$, against the CT obtained with NEWNADA, varying m from 2 to 29 features. We analyze a large number of aggregated flows, $n = 10^4$, and use two different number of slices, $M = 40$ and $M = 100$. The analysis is performed on traffic obtained from the WIDE network, combining different traces to attain the desired number of flows.

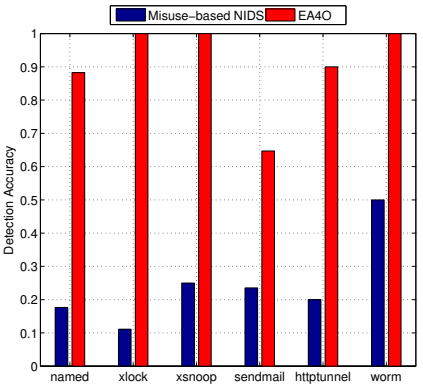
To estimate the CT of NEWNADA for a given value of m and M, we proceed as follows: first, we separately cluster each of the N sub-spaces X_i, and take the worst-case of the obtained clustering time as a representative measure of the CT in a single sub-space, i.e., $\mathrm{CT}(X_{\mathrm{SSCwc}}) = \max_i \mathrm{CT}(X_i)$. Then, if $N \leqslant M$, we have enough slices to completely parallelize the algorithm, and the total CT corresponds to the worst-case, $\mathrm{CT}(X_{\mathrm{SSCwc}})$. On the contrary, if $N > M$, some slices have to cluster various sub-spaces one after the other, and the total CT becomes $(N\%M + 1)$ times the worst-case $\mathrm{CT}(X_{\mathrm{SSCwc}})$, where % represents integer division. The first interesting observation from figure 3.6.(a) relates to the increase of $\mathrm{CT}(X)$ when m increases, going from about 8 seconds for $m = 2$ to more than 200 seconds for $m = 29$. As stated before, clustering in low-dimensional spaces is faster, which reduces the overhead of multiple clusterings computation.

The second important observation is about parallelization: if the algorithm is implemented in a parallel computing architecture, it can be used to analyze large volumes of traffic using many traffic descriptors in an on-line basis. For example, if we use 20 traffic features and a parallel architecture with 100 slices, we can analyze 10000 aggregated flows in less than 20 seconds. Modern network processor cards are able to perform traffic monitoring even in 10 Gbps network connections. In a average-loaded 10 Gbps link (about 50%-60%) there are about 500.000 packets per second; if we consider traffic flows with an average rate of 500 kbps (about 50 pkts/sec) and a average duration of at least 20 seconds, then we have about $n = 10000$ flows to analyze in each time slot, which means that NEWNADA could be even deployed in Gigabit networks if enough slices are available. Current network processor cards vendors offer multi-core solutions for high-performance networking with as much as 64 general purpose cores [2], which are perfectly adapted to deploy NEWNADA for very high speed knowledge-independent intrusions detection.

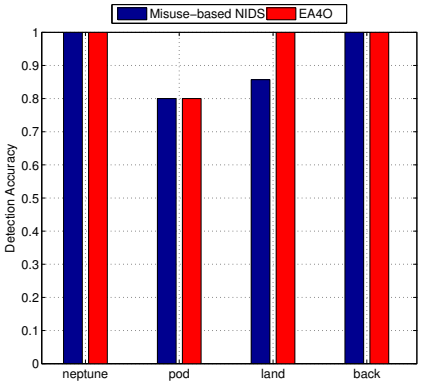
Figure 3.6.(b) compares $\mathrm{CT}(X)$ against $\mathrm{CT}(X_{\mathrm{SSCwc}})$ for an increasing number of flows n to analyze, using $m = 20$ traffic features and $M = N = 190$ slices (i.e., a completely parallelized implementation of NEWNADA). As before, we can appreciate the difference in CT when clustering the complete feature space vs. using low-dimensional sub-spaces: the difference is more than one order of magnitude, independently of the number of flows to analyze. Regarding the volume of traffic that can be analyzed with this 100% parallel configuration, NEWNADA can analyze up to 50000 flows with a reasonable CT, about 4 minutes in this experience. When evaluating with real traffic traces presented in Section 3.3, the number of aggregated flows in a time slot of $\Delta T = 20$ seconds rounds the 1500-2500 macro-flows, which represents a value of $\mathrm{CT}(X_{\mathrm{SSCwc}}) \approx 0.4$ seconds. For the $m = 9$ features that we have used ($N = 36$), and even without doing parallelization, the total CT is $N \times \mathrm{CT}(X_{\mathrm{SSCwc}}) \approx 14.4$ seconds.



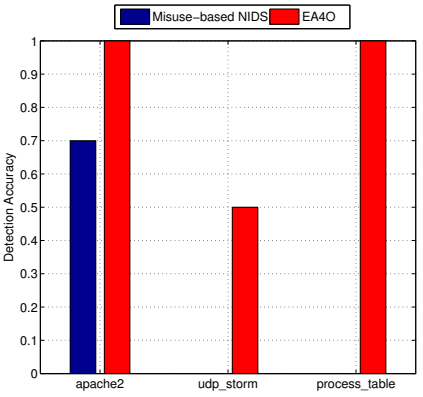
(a) R2L - Training dataset



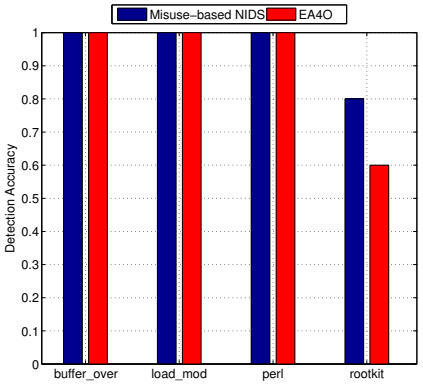
(b) R2L - Testing Dataset



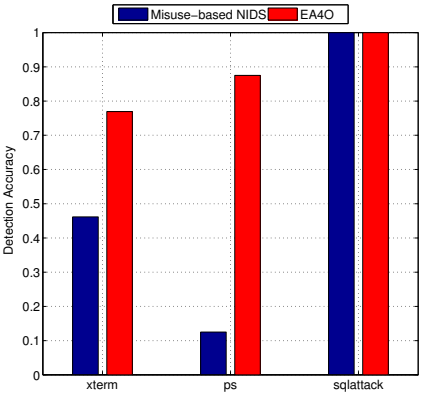
(c) DoS - Training dataset



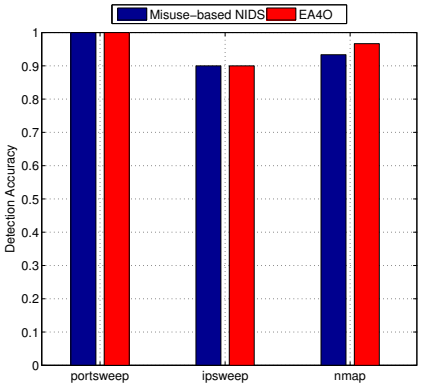
(d) DoS - Testing Dataset



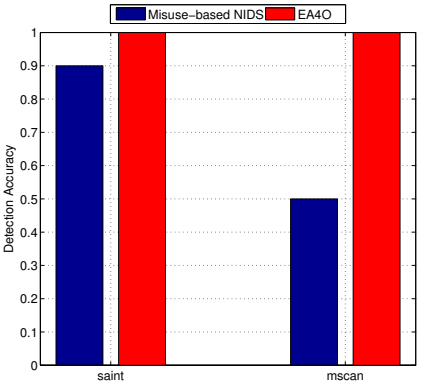
(a) U2R - Training dataset



(b) U2R - Testing Dataset

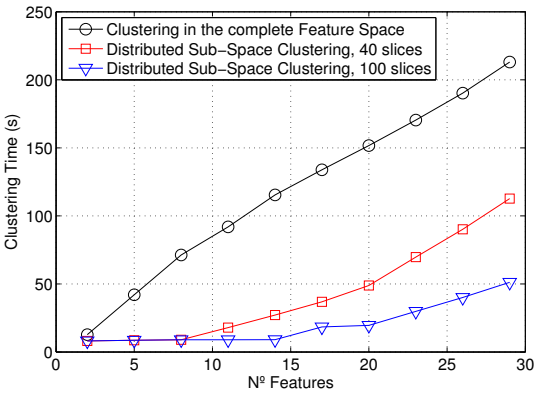


(g) Probing - Training dataset



(h) Probing - Testing Dataset

Figure 3.5: NEWNADA vs Misuse-based NIDS - R2L, DoS, U2R, and Probing attacks.



(a) Time vs. num features.

(b) Time vs. num macro-flows.

Figure 3.6: Computational Time as a function of number of features and number of macro-flows to analyze. The number of aggregated macro-flows in (a) is $n = 10000$. The number of features and slices in (b) is $m = 20$ and $M = 190$ respectively.

Chapter 4

ISP-Driven Informed Path Selection

4.1 Introduction

IDIPS is a path availability and performance service. IDIPS clients send a path ranking request to an IDIPS server. The ranking request contains a list of source addresses, a list of destination addresses and a ranking criterion. The IDIPS server determines all the possible <source, destination> pairs. IDIPS then computes the cost of each path accordingly to the ranking criterion requested by the client. The lower the cost, the better the path. IDIPS then ranks the paths to respect the COST but hides the computation and topology details. The IDIPS server then returns a list of <source, destination> pair to the requester. A rank value is associated to each such pair. The pairs with the lowest rank value are the most preferable paths. To simplify the processing at the requester, the returned list of ranked paths is ordered by rank value such that the first paths in the returned list are preferred over the last paths in the list.

Chapter 3 of deliverable D2.3 describes the IDIPS architecture. In this deliverable, we experimentally validate the IDIPS architecture. To do so, we evaluate the accuracy, the stability and the scalability that can be obtained with IDIPS. It is worth noticing that many experimentation results led us to architectural changes, even if D2.3 only presents the very final architecture we reached.

4.2 Accuracy

IDIPS by itself does not perform prediction, implying that there is no accuracy issue related to IDIPS itself. However, the way the measurement module is implemented can have an impact on the quality of the measurement and, subsequently on the prediction accuracy. In Section 3.5 of deliverable D2.3, we provide some example of measurement module and prediction module implementations. In this section, we show the impact of the measurement module implementation on its accuracy.

For the sake of the example, we propose an UDP ping Measurement module. This module does not aim at being used in a real environment where more robust measurements techniques should be used. To estimate the round-trip delay between a <source, destination> IP pair, we send a UDP segment to the destination on a port number that is very unlikely to be open. If the port is not opened and if no filtering applies, an ICMP port unreachable is expected to be returned to the Measurement module. The sending of the UDP segments is done by using

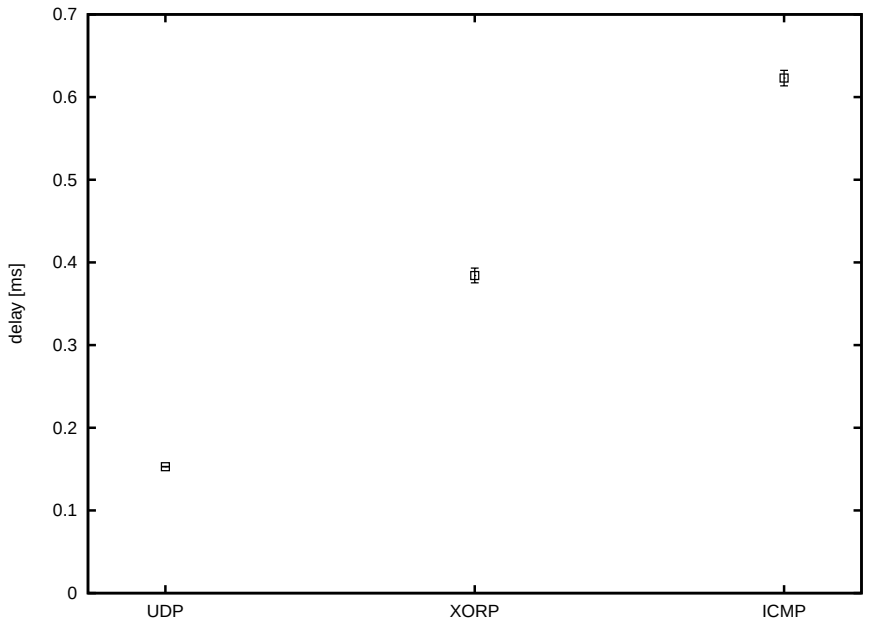


Figure 4.1: Delay precision comparison between standard UDP ping, ICMP ping and XORP UDP ping

the XORP socket API. XORP sockets are similar to the POSIX sockets except that they are asynchronous and that they are implemented with XRLs. In the remaining part of this section we will use the term *socket* to refer to the XORP socket abstraction. All the implementation details can be found in Section 3.5 of deliverable D2.3. The IP pairs are measured periodically. To implement periodic probing, we use a XORP periodic timer. Every second, this timer calls the `loop` method of our process. When this method is called, a UDP segment is sent to each <source, destination> IP pair that should have been measured at the latest when the `loop` method is called. It is not possible, without changing XORP to associate a time to an event on a socket. The implementation limits the inaccuracy of the delay estimation by determining the timestamp of the event as close as possible of the event as explained in deliverable D2.3.

Figure 4.1 compares the accuracy of the UDP ping we have implemented in a Measurement module with the standard traceroute UDP ping and the ICMP ping command. Fig. 4.1 plots the average of delay measured by the technique and the 95% confidence interval. For the comparison, we made 1,500 pings for each technique. The interval between two probes is one second. The setup uses two machines directly on the same VLAN. One running IDIPS on Linux, and the other receiving the UDP probes and running on Linux. The UDP ping is performed with the `traceroute` command and the ICMP ping with the `ping` command. The traceroute ran with the parameters `-q 1 -r -n -N 1 -U` to simulate a UDP ping.

From Fig. 4.1 `UDP ping` and `XORP` labels, we can see that using XORP introduces a bias in the measurement. This bias is mostly due to the process context switching introduced by the XRL-based implementation of the XORP sockets. Indeed, when a segment arrives at the host, it is not delivered immediately to the measurement process. It is first received by the main XORP process which then notifies the Measurement module process that an event occurred on the socket (i.e., the reception of the ICMP port unreachable). This design thus

implies process switching in addition to the standard kernel/user space switching with POSIX sockets. In addition, this scheme is also applied when sending the segment meaning that the time between the probes is sent from the Measurement module process and the time the probe is effectively received by the kernel depends on how rapidly the XRL is sent and processed by the main XORP process. However, the bias is of less than 0.3ms in our setup which is acceptable for most of the measurements. More interestingly, the measurement is better with the UDP ping implemented in IDIPS than the standard ICMP ping. Unfortunately, we did not manage to determine the reason why it performs better but the difference could come from the fact that content of the ICMP ping must be copied while it is not required with the UDP ping.

As any prediction based on any measurement technique can be implemented in IDIPS, it is not possible to determine the accuracy of the system. However, with this example, we show that IDIPS should not impact the accuracy of a measurement and thus prediction. However, as explained in deliverable D2.3, the implementation of the measurement in IDIPS requires particular engineering attention with respect to the constraints imposed by XORP (i.e., everything is asynchronous and not timestamped).

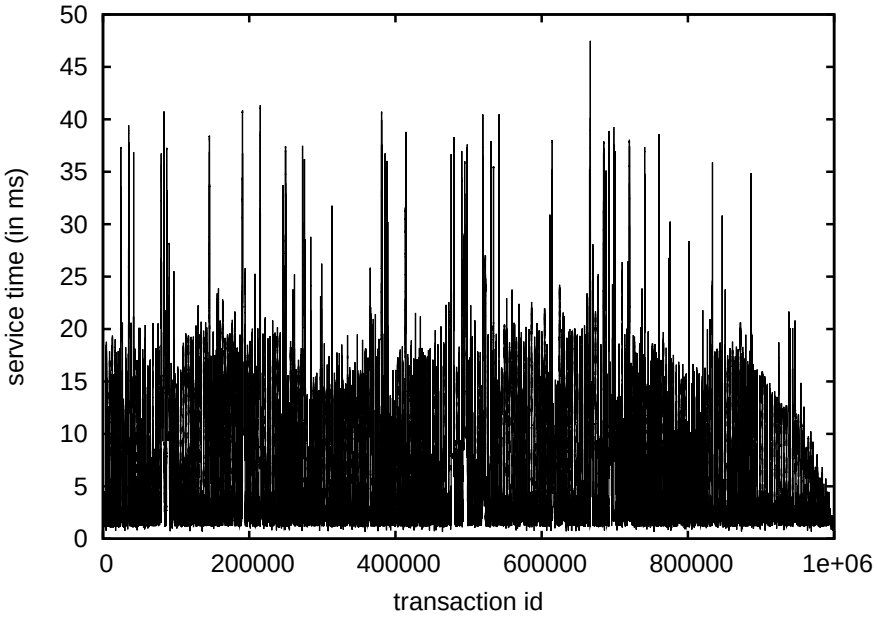
4.3 Stability

Figure 4.2 shows the IDIPS service time from the client perspective. In particular, Figure 4.2(a) gives the time (in ms) between each request and the associated reply for a particular run when the client request contains ten destinations. Figure 4.2(a) shows how IDIPS is stable over transactions, no drift is observed.

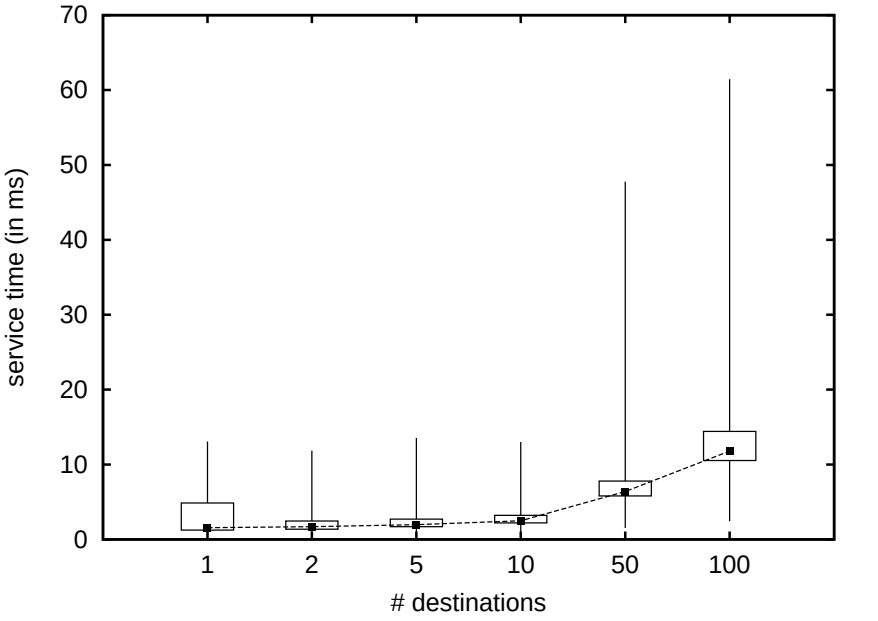
Figure 4.2(b) shows the IDIPS time distribution as quantiles. The dotted line represents the median value, while the box plot gives the minimum and 95th percentile values as well as the 25th and 75th percentiles. As expected, the IDIPS service time increases with the number of paths (i.e., the number of destinations in this example) induced by the client requests. The service time linearly increases with the number of paths. This linear dependency is a consequence of the conversion of the list received from the client in text into a binary format into the querying module implementation, the construction of the possible paths and the cost computation for each of such paths. The cost function having a temporal complexity of $\mathcal{O}(1)$, the total complexity is $\mathcal{O}(s*d) = \mathcal{O}(n)$ where s is the number of sources in the request, d, the number of destinations and n the number of paths. In our experiment, $s = 1$ making $n = d$. In addition, the larger the number of destinations, the less stable the service time as suggested by the service time distribution amplitude. The higher dispersion observed for the list of one destination can be explained by the overhead caused by the switching between the XORP processes (i.e., the finder and the querying module).

4.4 Scalability

Figure 4.3 breaks the IDIPS service time down into three categories: the network delay (labeled "network" - descending line pattern portion of the stacked bars), the path ranking (labeled "IDIPS" - ascending dashed line pattern portion of the stacked bars), and the internal XORP processing (labeled "XORP" - descending dashed line pattern portion of the stacked bars). For plotting those results, we consider the median value among the ten runs. Instead of plotting the median value of the service time, we rather consider the time proportion of each category.



(a) service time stability - destinations=10



(b) service time distribution

Figure 4.2: IDIPS service time as perceived by the client

The time consumed by the network is negligible. This is due to clients and server, which in our testbed, are separated by a single switch. However, as the IDIPS server is supposed to be deployed within a campus or an ISP network (alike a DNS service), one can imagine that the required network time (i.e., time spend in the network between the client and the server and vice-versa) would be very close to what we experienced in our testbed.

In general, the time spent in the whole ranking process in IDIPS increases linearly with the

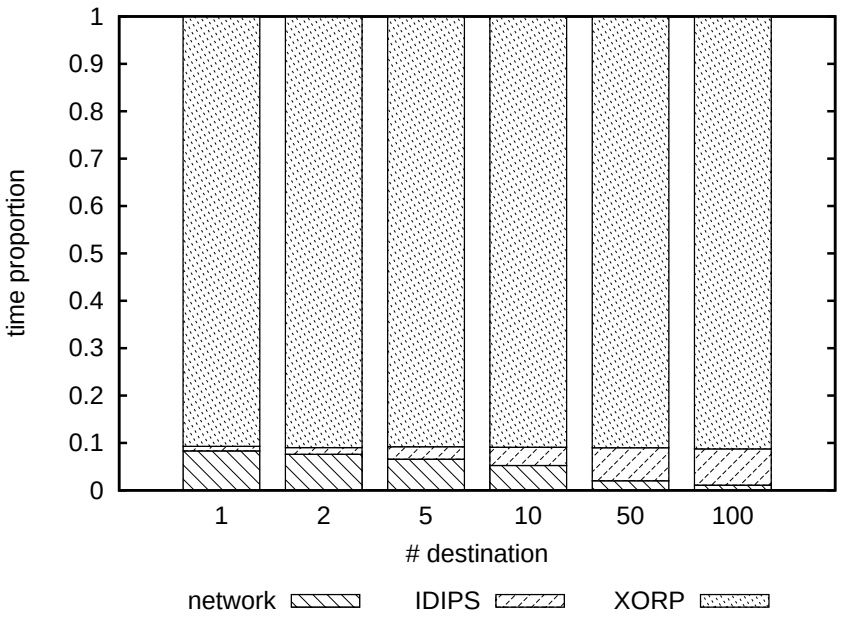


Figure 4.3: Proportion of the service time split - median over the ten runs

number of paths from the requests. With about only 12% to 20% of the time spent building the list to return to the client once the costs are computed. The rest of the time is spent by cost computation and attribute retrieval. The internal XORP processing represents most of the service time (around 90%) and is also linearly dependent with the number of paths from the requests. The time spent directly in the XORP internals is mostly due because of the marshaling and unmarshaling of the XRLs and the context switching between the `finder` and the querying module (remember that the XRLs are always processed by the `finder`.

Figure 4.4 shows the load on IDIPS in terms of requests/second number. As expected, the capacity of IDIPS to process requests decreases with the request size. It is expected behavior as large requests require more processing time, in terms of IDIPS (typically more cost function to evaluate and, thus, more lookups into the predicted values storage) and internal XORP processing (as already suggested by Figure 4.3).

We also notice that, in the worst case (i.e., 100 destinations per request), IDIPS can still process more requests per second than what it could be required for peer-to-peer applications [50].

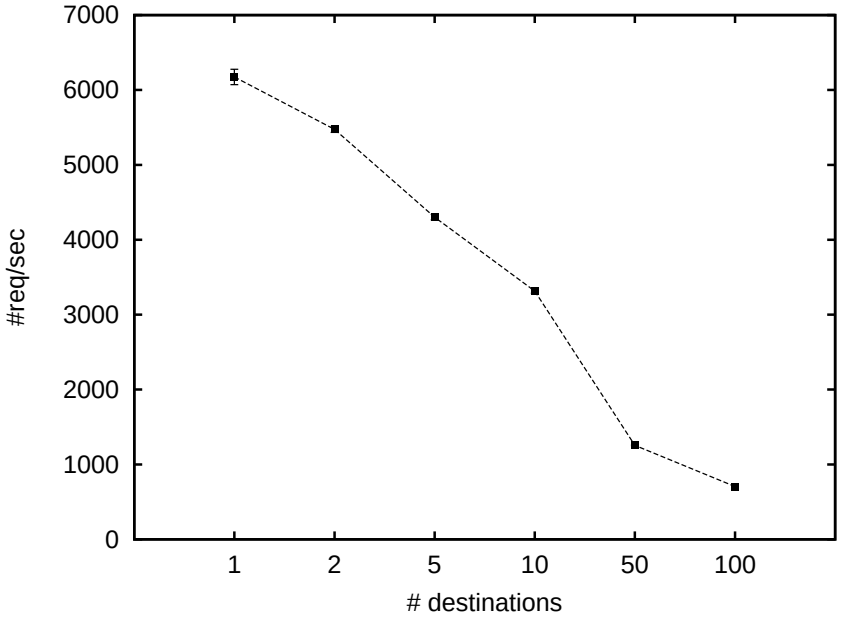


Figure 4.4: Load on IDIPS without XRL

Chapter 5

Path availability and coordinate system

In Appendix A, we propose a distributed system that meets the routing system scalability and quality objective. To this end, we first designed DMFSGD (Decentralized Matrix Factorization based on Stochastic Gradient Descent), a novel Network Coordinate Systems (NCS), which is a very scalable decentralized machine learning engine to predict unknown network performance metrics (typically delay, available bandwidth) from a relatively small number of measurements between some pairs of nodes. Secondly, we relied on this NCS to discover appropriate routing shortcuts in the network, namely paths through intermediate nodes that turn out to have smaller delays than the direct paths. In this section, we focus on the evaluation of these two components. Both parts have been extensively validated through simulations on various real datasets.

According to deliverable D4.1, our experimental evaluation criteria were defined as follows:

- For DMFSGD, the NCS, we will evaluate its distance prediction accuracy. We want to predict distances (delays) within the network, and the more accurate the predictions are, the better. Our reference distance matrix was obtained by active measurement over all pairs of nodes. In particular, we will show that our DMFSGD outperforms Vivaldi, which was one of our objectives. We also show the impact of the three key parameters of our algorithm [78, 77].

- For the shortcut detector, we will evaluate the gain of discovered routing shortcuts, which will give a direct insight of its performance. This gain is computed as the ratio between the best shortcut we have detected, and the existing direct path. For this deliverable we have refined this criterion. Indeed, as there are many paths for which shortcuts are detected, we thus obtain one value for each path and we build the CDF (Cumulative Distribution Function) of these values. Moreover, as we usually discover many candidate shortcuts per path, we will show the efficiency of our methods when we keep only the first k candidate shortcuts among those found, while varying k [26].

5.1 Network Distance Prediction by Decentralized Matrix Factorization

In this section, we evaluate our DMFSGD algorithm and compare it with state-of-the-art approaches. Assuming n nodes in the network, a $n \times n$ distance matrix is constructed with some

distances between nodes measured and the others unmeasured. Let D denote the measured distance matrix with d_{ij} the measured distance from node i to node j and $\hat{D}$ the predicted distance matrix with $\hat{d}_{ij}$ the predicted distance computed from some function.

5.1.1 Evaluation Methodology

The evaluations were performed under the following criteria and on the following datasets.

Evaluation Criteria

- **Cumulative Distribution of Relative Estimation Error** Relative Estimation Error (REE) is defined as
$$REE = \frac{|\hat{d}_{ij} - d_{ij}|}{d_{ij}}.$$
- **Stress** Stress measures the overall fitness and is used to illustrate the convergence of the algorithm, defined as
$$stress = \sqrt{\frac{\sum_{i,j=1}^{n} (d_{ij} - \hat{d}_{ij})^2}{\sum_{i,j=1}^{n} {d_{ij}}^2}}.$$
- **Median Absolute Error** Median Absolute Error (MAE) is defined as
$$MAE = median_{ij}(|d_{ij} - \hat{d}_{ij}|).$$

Datasets

- **Harvard226** contains dynamic and passive measurements of application-level RTTs, with timestamps, between 226 Azureus clients collected in 4 hours [72].
- **P2PSim1740** was obtained from the P2PSim project that contains static RTT measurements between 1740 Internet DNS servers [9, 49].
- **Meridian2500** was obtained from the Cornell Meridian project that contains static RTT measurements between 2500 nodes [117].
- **P2PSim525** is a complete sub-matrix between 525 nodes derived from P2psim1740.
- **Meridian2255** is a complete sub-matrix between 2255 nodes derived from Meridian2500.
- **Synthetic1000** contains the pairwise distances between 1000 nodes that are randomly generated in a 10-dimensional Euclidean space.

The first five datasets were obtained from real-world networks and contain a large percentage of Triangular Inequality Violation (TIV) edges, whereas the last one was synthesized and is TIV free. Here, an edge $\overline{AB}$ is claimed to be a TIV if there exists a triangle $\triangle ABC$ where $\overline{AB} > \overline{BC} + \overline{AC}$. The last three datasets were only used in Section 5.1.2 for the purpose of comparing the models of Euclidean embedding and matrix factorization.

Table 5.1: Properties of The Datasets

Dataset	Nodes	Symmetry	TIV percentage	Dynamic
Harvard226	226	/	/	Yes
P2PSim1740	1740	Yes	85.53%	No
Meridian2500	2500	Yes	96.55%	No
P2PSim525	525	Yes	76.17%	No
Meridian2255	2255	Yes	96.25%	No
Synthetic1000	1000	Yes	No	No

Table 5.1 summarizes these datasets. Note that we can neither tell the symmetry nor calculate the TIV percentage of the Harvard226 dataset as the measurements between network nodes vary over time largely, sometimes in several orders of magnitudes. The Harvard226 dataset is rather dense with about 3.9% pairwise paths unmeasured in 4 hours. The other paths are measured in uneven frequencies with one measured the maximal 662 times and one the minimal 2 times. About 94.0% of the paths are measured between 40 and 60 times.

Implementations for Different Datasets As mentioned earlier, the DMFSGD algorithm adopts the conventional random neighbor selection procedure in the scenarios where measurements are probed actively and maintains dynamically an active neighbor set for each node in the scenarios where measurements are obtained passively. Thus, for the Harvard226 dataset, we let each node maintain an active neighbor set containing the nodes it has contacted within the past 30 minutes and the timestamped measurements are processed in time order. For the other datasets, the random neighbor selection is used and the measurements are processed in random order with no neighbor decay as they are static.

To handle the dynamics of the measurements in Harvard226, the distance filter in [72] is adopted that smoothes the streams of measurements within a moving time window (30 minutes in our case), by a median filter. In the evaluation, we built a static distance matrix by extracting the median values of the streams of measurements between each pair of nodes and used it as the ground truth.

5.1.2 Euclidean Embedding vs. Matrix Factorization

One major difference between Euclidean embedding and matrix factorization is that they both solve the same problem but are subject to different constraints. Euclidean embedding requires $\hat{D}$ to be symmetric and to satisfy the triangle inequality, whereas matrix factorization only requires $\hat{D}$ to be low rank. Below, we compare empirically Euclidean embedding and matrix factorization to give some insights to why this difference makes matrix factorization more suitable for network distance prediction.

Algorithms To make the model comparison fair, we chose the state-of-the-art algorithms to solve the Euclidean embedding and matrix factorization problems so that both are solved to their limits. For Euclidean embedding, Multi-Dimensional Scaling (MDS) is the most popular technique that searches the optimal embedding using an iterative algorithm. We adopted the MDS implementation, mdscale, in the statistical toolbox of matlab [3].

For matrix factorization, SVD provides the analytic solution which is globally optimal [47]. Generally, SVD factorizes a given matrix D into three matrices of the form

$$D = USV^T,$$

where U and V are unitary matrices (a unitary matrix is a (square) $n \times n$ complex matrix U satisfying the condition $U^TU = UU^T = I$, where I is the identity matrix), and S is a diagonal matrix with nonnegative real numbers on the diagonal. The positive diagonal entries are called the singular values and their number is equal to the rank of D.

To obtain a low-rank factorization, we keep only the r large singular values in S and replace the other small ones by zero. Let S_r be the new S, $X = US_r^{\frac{1}{2}}$ and $Y^T = S_r^{\frac{1}{2}}V^T$, where $S_r(i,i)^{\frac{1}{2}} = \sqrt{S_r(i,i)}$. Then, $\hat{D} = XY^T$ is the optimal low-rank approximation to D.

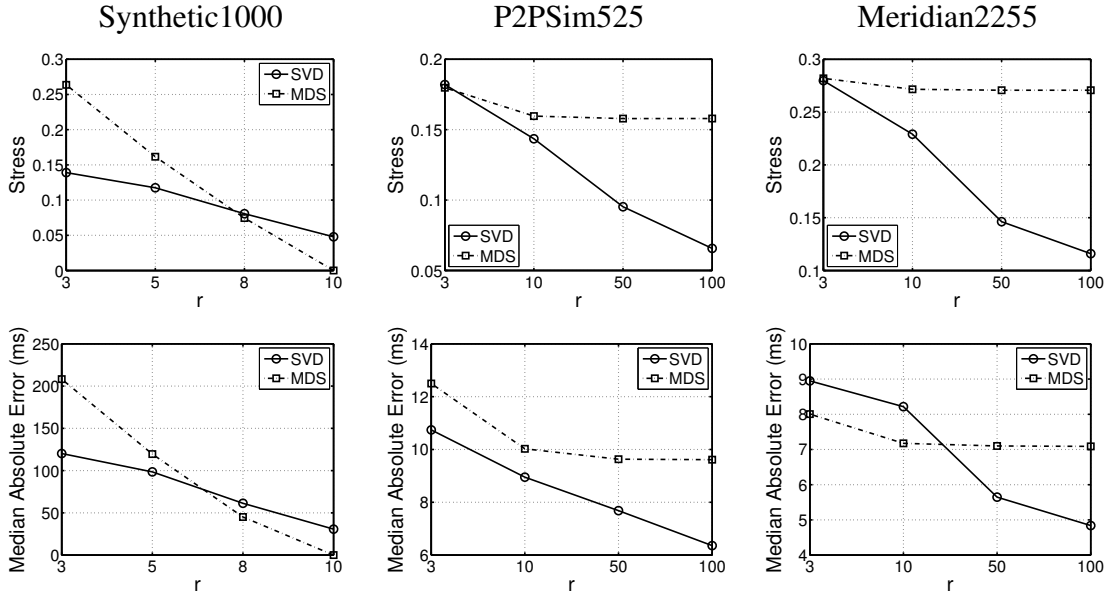


Figure 5.1: Comparison of MDS-based Euclidean embedding and SVD-based matrix factorization on synthetic1000, P2psim525 and Meridian2255. The stresses and the median absolute errors by both methods in different dimensions/ranks are shown on the first two rows respectively. Note that a perfect embedding with no errors was generated for Synthetic1000 in the 10 dimensional Euclidean space by MDS.

Evaluations Since SVD cannot handle missing data, we only compare MDS and SVD on the three complete datasets including Synthetic1000, P2psim525 and Meridian2255. On each dataset, we ran MDS and SVD in different dimensions and ranks and computed the stresses and MAE, shown in figure 5.1. It can be seen that the accuracies by SVD monotonically improve on all three datasets as the rank increases, whereas consistent improvements by MDS are only found on Synthetic1000 which is TIV-free. On P2psim525 and Meridian2255 where severe TIVs exist, MDS achieves no or little visible improvement after 10 dimensions.

These evaluations demonstrate the influences of different constraints imposed on the two techniques. For Euclidean embedding, the symmetry constraint doesn't cause any problem as the RTTs in all datasets are symmetric. However, the constraint of triangle inequality is strong

Table 5.2: Matrix Factorization vs. Euclidean Embedding

	Matrix Factorization	Euclidean Embedding
node coordinate	$x_i = (x_{i1}, \cdots, x_{ir})$ $y_i = (y_{i1}, \cdots, y_{ir})$	$x_i = (x_{i1}, \cdots, x_{ir})$
distance function	$\hat{d}_{ij} = x_i y_j^T$	$\hat{d}_{ij} = \sqrt{(x_i - x_j)^T (x_i - x_j)}$
constraints	low rank	Symmetry: $\hat{d}_{ij} = \hat{d}_{ji}$ Triangle Inequality: $\hat{d}_{ij} < \hat{d}_{ik} + d_{kj}$

and can't be relieved by increasing dimensions. In contrast, matrix factorization makes no assumptions of triangle inequality, thus is not affected by the TIVs in the data. Note that the accuracy improvement by increasing the rank is guaranteed for SVD-based matrix factorization. However, this conclusion cannot be extended to the cases where missing data is present. We will show later that increasing the rank beyond some value in matrix factorization for a large amount of missing data will not further improve the accuracy.

This comparative study reveals the model advantages of Matrix Factorization over Euclidean embedding. Overall, Euclidean embedding has a geometric interpretation which is useful for visualization. However, due to the existence of TIVs and the possible asymmetry in network distance spaces, low-rank matrix factorization is more suitable for modeling the network distance spaces. Table 5.2 lists a detailed comparison of matrix factorization and Euclidean embedding.

5.1.3 Impacts of Parameters

This section discusses and demonstrates the impacts of the parameters of our DMFSGD algorithm.

k, r and λ Our DMFSGD algorithm has two common parameters, the regularization coefficient λ and the rank r, and the additional parameter, the number of neighbors k for the scenarios where measurements are probed actively. Intuitively, r is the number of unknown variables in each coordinate and k is the amount of known data that is used to estimate each unknown variable in a coordinate. λ controls the extent of the regularization which avoids both overfitting and the drifts of the coordinates.

Clearly, increasing k is equivalent to adding more data and thus always helps improve the accuracy. However, a larger k also means more probe traffic and consequently higher overheads. On the other hand, only a certain number of unknown variables can be accurately calculated from a certain amount of known data. Thus, increasing r beyond some value for a fixed value of k will only lead to severe overfitting and consequently, a large λ is needed to address it. In practice, the value of k should be fixed according to the requirements of the applications by trading off between accuracies and measurement overheads. Following the suggestion in Vivaldi [31], we set $k = 32$ for P2psim1740 and for Meridian2500 in the rest of this section. Note that $k = 32$ makes the available measurements considerably sparse. For instance, $32/1740 = 1.84\%$ measurements are available for each node in P2PSim1740 and $32/2500 = 1.28\%$ for each node in Meridian2500. Recall that no k is set for Harvard226.

Experiments under Different Configurations We then experimented with different configurations of $r = \{3, 10, 100\}$ and $\lambda = \{0.01, 0.1, 1, 10\}$, with different loss functions and whether to incorporate the non-negativity constraint, shown in Figure 5.2. η is adapted by the line search, with the initial value of 10^3 for the L_2 loss function and of 10^2 for the L_1 loss function.

In particular, we made the following observations. First, the DMFSGD algorithm is generally more accurate when the robust L_1 loss function and the non-negativity constraint are incorporated. The likely reasons are that the L_1 loss function is insensitive to large fitting errors some of which are introduced by measurement outliers and that the non-negativity constraint reduces the searching space which makes it easier to find a stable solution. Thus, the robust L_1 loss function and the non-negativity constraint are incorporated in the DMFSGD algorithm by default.

Second, $\lambda = 1$ seems to be a good choice under most configurations and is thus adopted by default. Third, r has different impacts to different datasets due to their different data properties. In Harvard226 where available measurements are dense, the prediction accuracy improves monotonically with r, whereas in the other two datasets where available measurements are sparse due to the setting of a small k, better performance is achieved with $r \leqslant 10$ and a large λ is needed to overcome the overfitting caused by larger r's, which confirms our analysis in the previous section. Thus, by trading off between the performance on all three datasets, $r = 10$ is adopted by default.

η As mentioned earlier, SGD is sensitive to the learning rate η where a too large η leads to the overflow of the solutions and a too small η slows down the convergence. Although this sensitivity is reduced by minibatch SGD, it is still difficult to find an appropriate constant that works for all datasets and in all situations. We experimented with different constant η's and with the line search to adapt η dynamically. From Figure 5.3, it can be seen that the line search strategy performs best in terms of both accuracy and convergence speed. Note that the convergence speed is illustrated by the stress and MAE improvements with respect to the average measurement number per node, i.e., the total number of measurements used by all nodes divided by the number of nodes[1]. It can be seen that the DMFSGD algorithm converges fast after each node probes, on average, $10 \times k$ measurements from its k neighbors. Although no k is set for Harvard226, we treat it as $k = 226$.

Discussions By incorporating the line search strategy, the L_1 loss function and the non-negativity constraint, our DMFSGD algorithm is left with two tunable parameters: the rank r and the regularization coefficient λ. The default configuration of $\lambda = 1$ and $r = 10$ is not guaranteed to be optimal in different situations and on different datasets. However, fine tuning of parameters is difficult, if not impossible, for network applications due to the measurement dynamics and the decentralized processing where local measurements are processed locally at each node with no central nodes gathering information of the entire network. Empirically, the default parameter setting leads to good, though not the best, prediction accuracy to a large variety of data.

[1]For P2PSim1740 and Meridian2500, at any time, the number of measurements used by each node is statistically the same for all nodes due to the random selections of the source and the target nodes in the updates. For Harvard226, this number is significantly different for different nodes because the paths were passively probed with uneven frequencies.

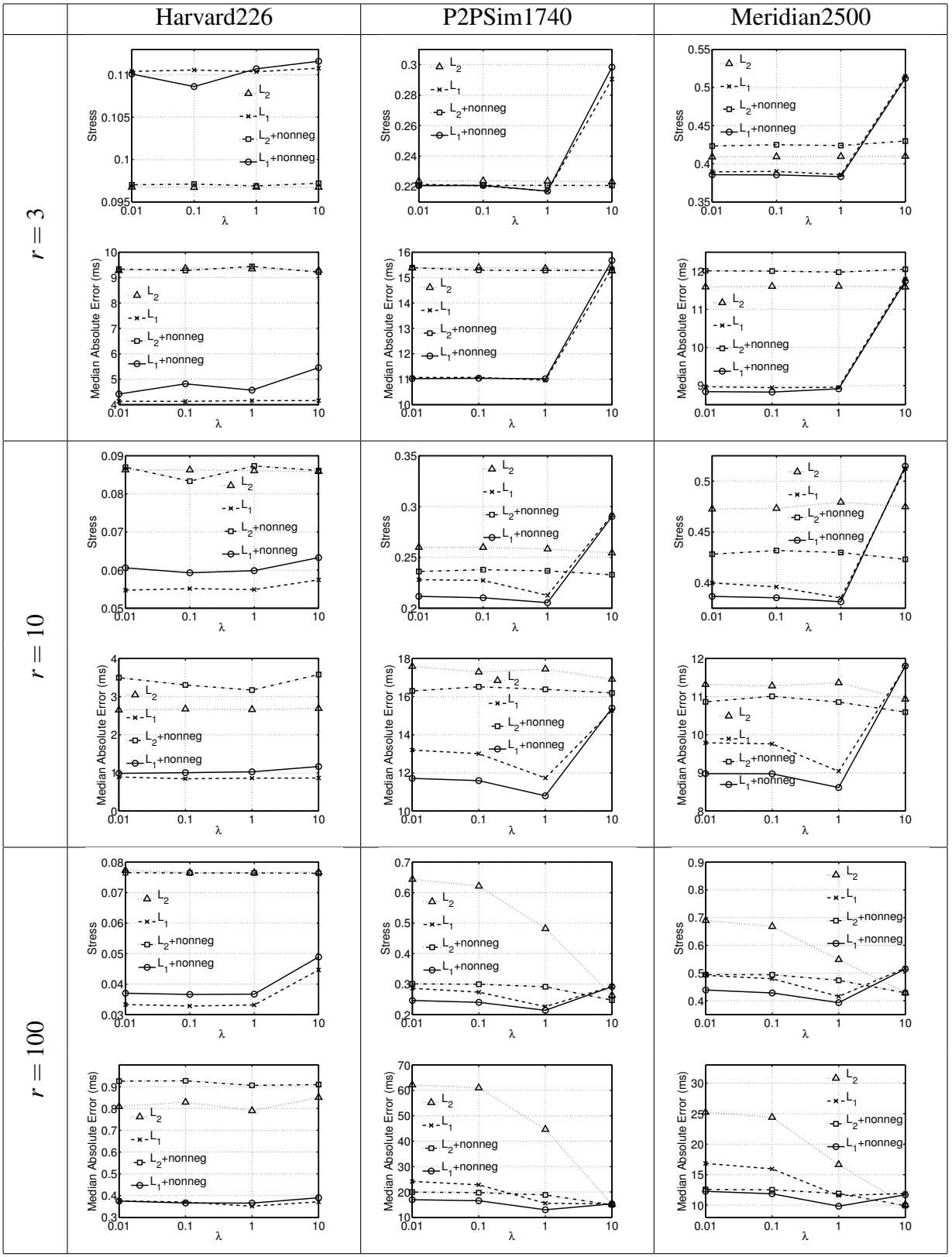


Figure 5.2: Impact of parameters.

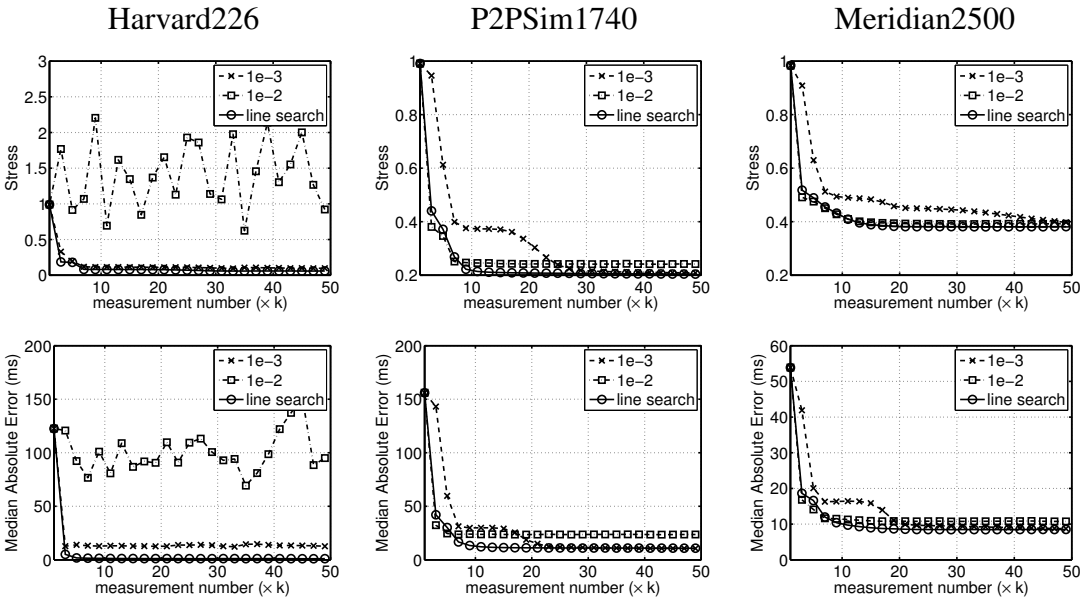


Figure 5.3: Impact of η under $\lambda = 1$ and $r = 10$ with the L_1 loss function and the non-negativity constraint. k is treated as 226 for Harvard226 and $k = 32$ for P2PSim1740 and Meridian2500.

The setting of $k = 32$ has been commonly adopted in many systems such as Vivaldi. However, most systems contain network nodes of a few thousands or less. For large systems of more nodes, k has to be scaled with the number of nodes n. According to the theory of matrix completion [25, 24, 65], one can recover an unknown $n \times n$ matrix of low rank r from just about $O(nr\log n)$ noisy entries with an error which is proportional to the noise level. Thus, $k \propto r\log n$ to guarantee a decent prediction accuracy.

5.1.4 Comparisons with Vivaldi

Among numerous approaches on network distance prediction, we consider Vivaldi [31] as the state of the art because of its accuracy and its practicability. To the best of our knowledge, Vivaldi is the only system that has been adopted in a real Internet application, Azureus [114]. Other approaches such as GNP [88] and IDES [31] are impractical due to the usage of landmarks which makes them impossible to be evaluated on the Harvard226 dataset. We consider these landmark-based systems as a special variation of a generic decentralized model.

We only compare our DMFSGD algorithm with Vivaldi. To address the measurement dynamics and the skewed neighbor updates, we adopted the Vivaldi implementation in [72] [2] when dealing with the Harvard226 dataset. The conventional Vivaldi in [31] was adopted to deal with the other two datasets. We refer to the former as Harvard Vivaldi to make the distinction. In addition, despite the impracticality, we also demonstrate the flexibility of the DMFSGD algorithm in dealing with the landmark-based architecture, referred to as DMFSGD Landmark, by forcing each node to only select the landmarks as neighbors. Note that we only ran DMFSGD Landmark on P2PSim1740 and Meridian2500 because the dynamic measurements in Harvard226

[2]The source code was downloaded from `http://www.eecs.harvard.edu/~syrah/nc/`.

were obtained passively and thus we cannot select landmarks and force each node to only communicate with them. To make the comparison fair, 32 landmarks were randomly selected.

Figure 5.4 shows the comparisons between DMFSGD, Vivaldi/Harvard Vivaldi and DMFSGD Landmark. It can be seen that on different criteria, our DMFSGD algorithm either outperforms or is competitive to Vivaldi. On Harvard226, DMFSGD is significantly better on all criteria, especially on the MAE where DMFSGD achieved the $1ms$ MAE, in contrast to the $5ms$ by Harvard Vivaldi, meaning that half of the estimated distances have an error of less than $1ms$. On P2PSim1740, DMFSGD is better on the MAE and the cumulative distributions of REE, whereas on Meridian2500, DMFSGD achieved similar performance as Vivaldi on all criteria. Note that DMFSGD and DMFSGD Landmark performed similarly on P2PSim1740 and Meridian2500.

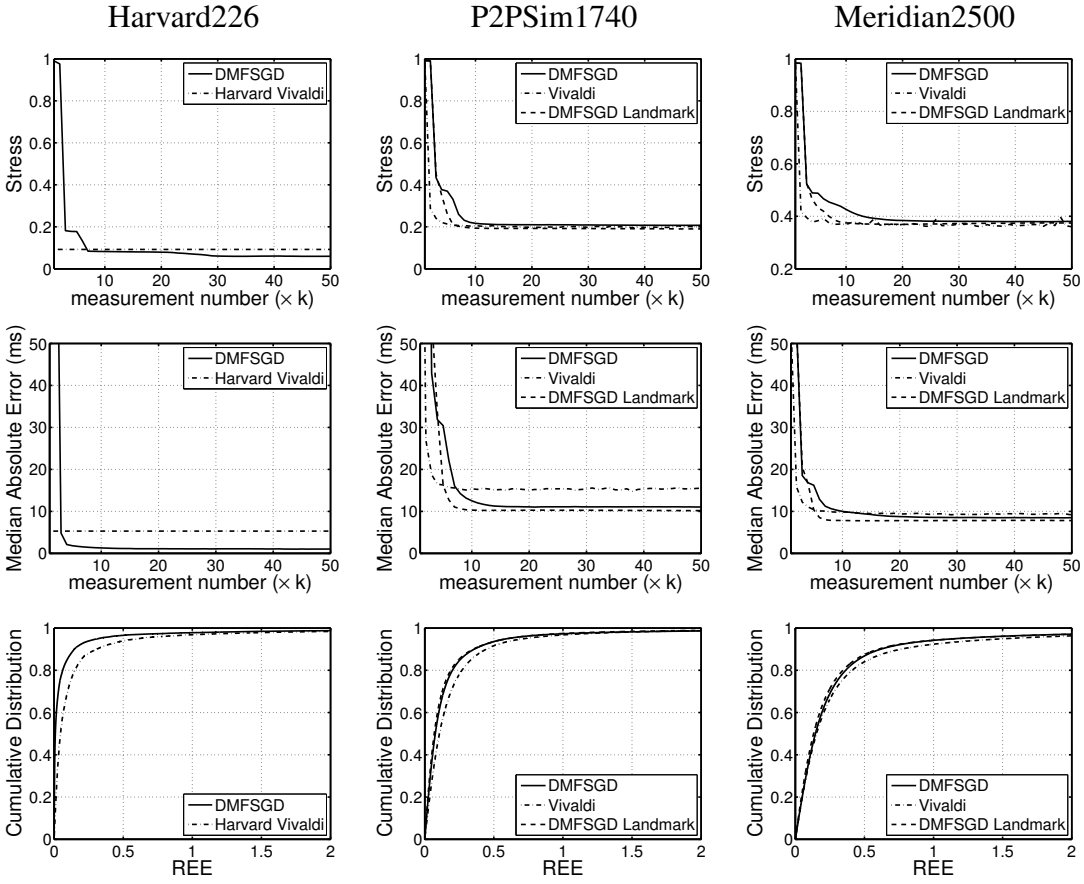


Figure 5.4: Comparison of DMFSGD and Vivaldi. The default configuration of $\lambda = 1$ and $r = 10$ with η adapted by the line search, the L_1 loss function and the non-negativity constraint is used in DMFSGD and DMFSGD Landmark. The 10 dimensional Euclidean space with the Height model is used in Vivaldi and Harvard Vivaldi. Note that as the implementation of Harvard Vivaldi only outputs the results in the end of the simulation, the final stress and the final MAE are plotted as a constant.

5.2 Finding routing shortcuts

The knowledge of estimated delays between nodes can also be useful to select better paths for real-time applications. In deliverable D2.3, we have proposed some methods that rely on the nodes running an ICS to detect routing shortcuts in networks.

5.2.1 Problem Formalization

We say that an edge AB is a TIV-edge when there exists a routing shortcut ACB via some node C in terms of delay. In such case, using C as a relay node to go from A to B instead of sending the data directly from A to B reduce the delay experienced between A and B and is called overlay routing. The second subproblem we address then consists in finding C nodes that are routing shortcuts for a given path AB.

5.2.2 Experimentation and evaluation

To model Internet latency, we used three delay matrices containing results of measurements performed in real networks: two publicly available data sets, the P2PSim data (1740 nodes) [9] and the Meridian data (2500 nodes) [117], and a data set we obtained by doing measurements between 180 nodes on Planetlab [93]. In these matrices, the percentage of paths for which there exists at least one shortcut is respectively 86%, 97% and 67%. Since a shortcut is not necessary useful[3], we define an *interesting shortcut* as a shortcut that provides at least an absolute gain of $10\,ms$ and a relative gain of 10%. The percentage of paths for which there exists at least an interesting shortcut in our matrices is respectively 43%, 83% and 16%. So, searching shortcuts in the networks modelled by these matrices can provide an improvement in terms of delays for many paths.

We have simulated the behavior of Vivaldi on these three networks by using the P2PSim [9] 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. We will now evaluate the quality of the sets of detected nodes provided by our criteria.

Shortcut detection

To evaluate the performance of our detection criteria, we first use the classical true positive rate and false positive rate indicators. For a path AB, a good shortcut detection criterion must detect a node C as a shortcut if it is a shortcut for the path AB (i.e., if it is a positive) and must reject a node C if it is not a shortcut for the path AB (i.e., if it is a negative). The percentage of positives detected as shortcuts is the *true positive rate* (TPR) and the percentage of negatives detected as shortcuts is the *false positive rate* (FPR). We also define the *interesting true positive rate* (ITPR) as the percentage of interesting shortcuts detected as shortcuts by the criterion. A good detection criterion must provide a high (I)TPR and a low FPR.

[3]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, such shortcuts are useless in practice.

	EDC			ADC			HDC		
	TPR	ITPR	FPR	TPR	ITPR	FPR	TPR	ITPR	FPR
P2PSim	53%	83%	2%	65%	84%	9%	60%	83%	3%
Meridian	54%	64%	9%	70%	76%	25%	56%	66%	9%
Planetlab	37%	75%	1%	60%	81%	5%	59%	84%	2%

Table 5.3: Criteria shortcut detection results

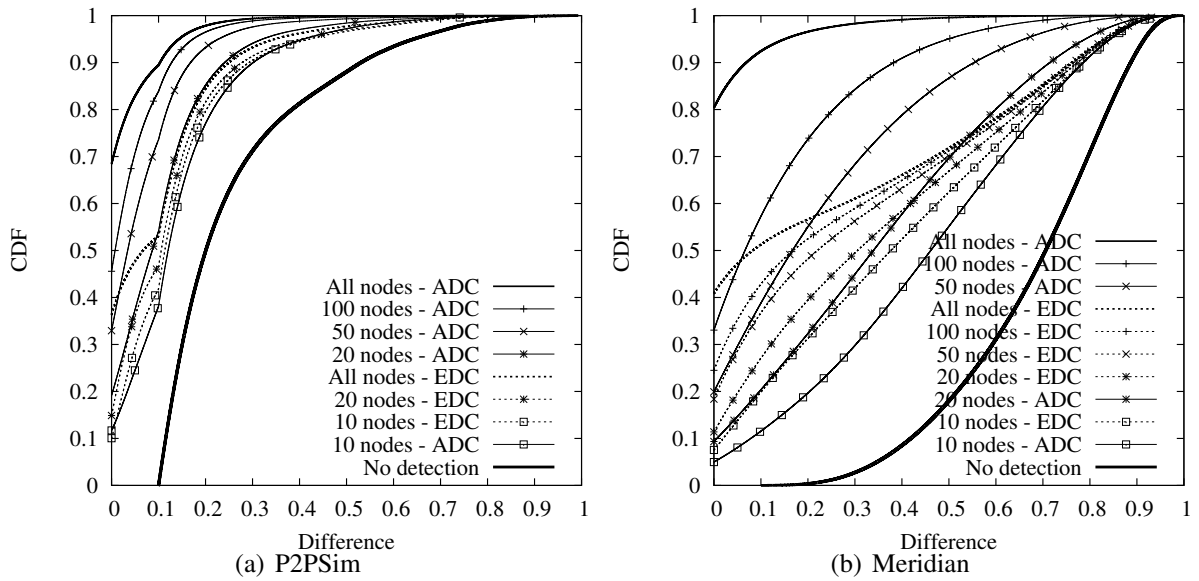
The true positive rates and false positive rates obtained with our criteria are given in table 5.3. We see that the percentage of interesting shortcuts detected as shortcuts (ITPR) is good in most of the cases for each criterion. Furthermore, the percentage of non-shortcuts detected as shortcuts (FPR) is generally quite low. Considering these results, EDC seems to perform better than ADC: although ADC is always able to detect slightly more shortcuts than EDC, it also gives more false positives. EDC seems also to perform better than HDC: HDC is able to detect a little bit more shortcuts than EDC (and less than ADC) but, HDC also gives more false positives than EDC (but less false positives than ADC). So, in terms of quantity of detected shortcuts, HDC appears like an intermediate solution between EDC and ADC.

Detection of the best shortcuts

Being able to detect lots of the shortcuts 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 one interesting shortcut, the percentage of paths for which the most interesting shortcut is detected in the matrices P2PSim, Meridian and Planetlab is respectively 36%, 41% and 49% with the EDC criterion, 68%, 80% and 70% with the ADC criterion and 64%, 71% and 74% with the HDC criterion.

Regarding those results ADC seems to be a better criterion than EDC. Indeed, EDC is able to find the best shortcut for 40% of the paths (on average) while the ADC is able to find the best shortcut for 70% (on average) of the paths. The results obtained with HDC are better than those obtained with EDC but are worse than those obtained with ADC. So, HDC misses interesting shortcuts that ADC is able to find. However, we must perhaps moderate our conclusion. Firstly because ADC returns large sets of *C* nodes (including a non-negligible number of false positives) compared to HDC and EDC. At the limit, a criterion that detects as shortcut all *C* nodes will obviously detect the best shortcut for each path but is completely useless. So, if we choose to use ADC, we absolutely need a criterion[4] to rank the *C* nodes of a set in order to keep only a subset of the nodes. Moreover, EDC and HDC 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 investigate that during the evaluation of the quality of the ranking of the *C* nodes.

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



(a) P2PSim (b) Meridian

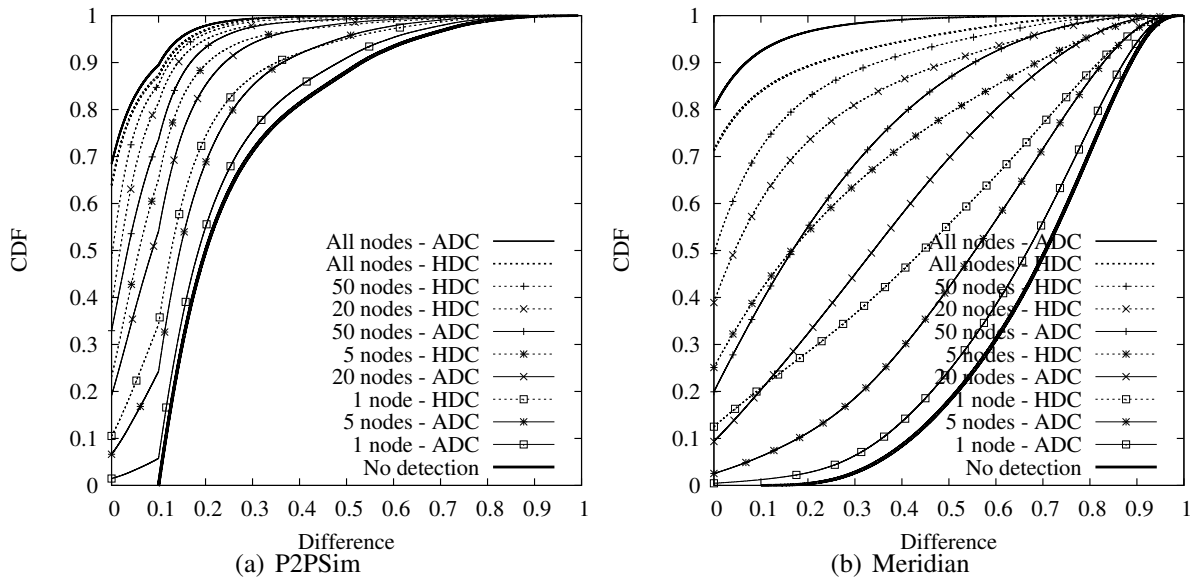
Figure 5.5: Comparison of EDC and ADC: Difference of G_r between the best shortcut and the best detected shortcut.

Ranking of the detected nodes

For a given matrix and a given detection criterion, we proposed to rank the C nodes of each path on the 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. For this evaluation, we only consider the paths for which there exists at least one interesting shortcut. To perform this evaluation, we consider for each path that only the first k C nodes of the ranking are detected by the criterion (for several values of the parameter k). For these subsets of C nodes we 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 thus obtain one value for each path and we build the CDF (Cumulative Distribution Function) of these values. The CDFs obtained with different values of the parameter k for the criteria EDC and ADC are given in figure 5.5.

The graphs named "no detection" in figures 5.5(a) and 5.5(b) give the CDF of the G_r provided by the best existing shortcut for each path of the matrix. 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 one shortcut detection criterion, we will detect some shortcuts and, thus, 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" in the subfigures of figure 5.5 give the CDF computed by considering all the nodes selected by the shortcut detection criterion XDC (ADC or EDC). This is equivalent to using $k = \infty$. These are the best results that the given detection criterion applied on the given matrix can provide. We can see that ADC still gives better results than EDC considering those graphs. Indeed, with ADC, there are only a small part of the paths for which the difference between the G_r provided by the best existing shortcut and the G_r provided by the best detected shortcut is bigger than 0.2. That means that, for a small part of the paths AB, it is still possible to find another shortcut C that would improve by more than 20% of $RTT(A,B)$



(a) P2PSim (b) Meridian

Figure 5.6: Comparison of ADC and HDC: Difference of G_r between the best shortcut and the best detected shortcut.

the gain provided by the alternative path proposed by our shortcut detection criterion. This difference is generally bigger with EDC.

Let us see what the situation is if we keep only the first nodes of the rankings. The graphs named "k nodes - XDC" in the subfigures of figure 5.5 give the CDF computed by considering as detected only the first k nodes of the rankings obtained by using the shortcut detection criterion XDC (ADC or EDC). The first thing we see is that even if we take only a few nodes in the ranking (e.g., 10 nodes), we obtain already a good improvement compared to the situation without shortcut detection. We also see 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[5]. 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 to obtain good results may seem important for Meridian but it represents only 4% of the total number of nodes.

Given these results we can conclude that, with ADC, when 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 paths for which there exists at least one interesting shortcut. But it is not a final conclusion: we have still to observe the results obtained with HDC. The CDFs obtained with different values of the parameter k for the criterion HDC (and the CDFs obtained with ADC for comparison) are given in figure 5.6.

Considering the quality of the ranking of the C nodes, we see in figure 5.6 (graphs named "k nodes - XDC") that HDC performs a lot better than ADC. Indeed, with HDC, even if we

[5]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.

take only the first node of the ranking, we obtain already a significant improvement compared to the situation without shortcut detection. Furthermore, if we only consider the first 5 nodes of a ranking, we obtain better results than if we consider the first 50 nodes of the rankings with ADC. So, with HDC, we can provide a substantial improvement of the RTT for lots of paths by considering only about 2% of the total number of nodes (that represents only 50 nodes for Meridian, 34 nodes for P2PSim and 3 nodes for Planetlab).

5.2.3 Conclusion

We have presented a novel approach to network distance prediction. The success of the approach roots both in the exploitation of the dependencies across distance measurements between network nodes and in the stochastic optimization which enables a fully decentralized architecture. A so-called Decentralized Matrix Factorization-based on Stochastic Gradient Descent (DMFSGD) algorithm is proposed to solve the distance prediction problem. The algorithm is simple, with the same architecture as Vivaldi, scalable, able to deal with dynamic measurements in large-scale networks, and accurate, generally superior to Vivaldi. In particular, experiments on real data collected from Azureus demonstrate the potential of the algorithm being utilized by Internet applications, which we would like to study in the future.

We showed that, for any given path AB, using only the RTT of that path and the information available in an ICS, it is possible to select a small set of nodes containing very likely an interesting one-hop shortcut (but not necessarily the best one) when shortcuts exist for that path. We obtained the best results with our shortcut detection criterion called HDC. With that criterion we are able to limit the number of potential shortcuts for any given path AB to about one or two percent of the total number of nodes in the network. So, to improve significantly the latency between A and B, we will only have to do measurements between A, B and these few candidate nodes to know if they are really shortcuts and which of them is the best shortcut.

Chapter 6

OSPF SRG inference

This section provides some insight into the issues that arise when implementing the SRG inference for the OSPF module. A generalized SRG table using probabilities allows to infer pre-learned SRGs given a currently observed failing link. In the next section, we show how the SRG table can be used to do this inference for any set of links, using conditional probability analysis. After this, we show how the OSPF protocol impacts its failure detection accuracy, and how the inference scales in terms of detection timing, memory sizes required for the SRG table and calculations for multiple failure inference.

6.1 Conditional probabilities

The SRG inference information is presented to the OSPF module in the form of a table of conditional probabilities as shown on Figure 6.1. Here L_i corresponds with router links in the OSPF area (each bi-directional physical link typically results in two adjacencies, which are flooded as two router links in the LSA messages). Note that in the SRG table, the set L consisting of L_i is allowed to grow as more failures and SRGs are detected. Generally, only L_i for which a failure has been observed at least once will be included in the table.

$$L = \{L_1, L_2, ..., L_m\}$$
$$|L| = m$$

SRG_j are the shared risk groups consisting of one or more (*not* zero) links L_i. SRG_j are therefore sets consisting of L_i:

$$SRG_j \subset L \text{ with } SRG_j \neq \emptyset$$

i.e.,

$$SRG_i \in 2^L \setminus \emptyset$$

The elements in the SRG table matrix consist of the conditional probabilities $P(SRG_j|L_i$, indicating probabilities of failure of SRG_j occurring if failure of link L_i is detected. For the conditional probabilities corresponding to a link L_i, the following stands:

$$\sum_{j=1}^{n} P(SRG_j|L_i + P(\{L_i\}|L_i)) \leq 1 \qquad \forall i \in 1, ..., m$$

LSA	SRG_1	SRG_2	...	SRG_n	{L}
L_1	$P(SRG_1\|L_1)$	$P(SRG_2\|L_1)$	...	$P(SRG_n\|L_1)$	$P(\{L_1\}\|L_1)$
L_2	$P(SRG_1\|L_2)$	$P(SRG_2\|L_2)$	...	$P(SRG_n\|L_2)$	$P(\{L_2\}\|L_2)$
...	...	...	...	...	...
L_m	$P(SRG_1\|L_m)$	$P(SRG_2\|L_m)$	...	$P(SRG_n\|L_m)$	$P(\{L_m\}\|L_m)$

Figure 6.1: Outline of SRG table

Where $P(\{L_i\}|L_i)$ denotes the conditional probability of the failure of singleton SRG $\{L_i\}$ will occur, that is, no further links will fail after L_i. $P(\{L_i\}|L_i)$ may be zero. The following convention is adopted for including $\{L_i\}$ as a column in the SRG table: if $\{L_i\}$ is not found, then it is assumed that the 'left-over' conditional probability of the row is assigned to $P(\{L_i\}|L_i)$:

$$P(\{L_i\}|L_i) = 1 - \sum_{j=1}^{n} P(SRG_j|L_i) \quad \geq 0 \qquad \forall i \in 1,...,m$$

$P(\{L_i\}|L_i)$ is then not stored but calculated at detection time (only necessary when no links other than L_i were detected as failing).

Alternatively, $\{L_i\}$ is reported as a column in the SRG table. In this case, the leftover conditional probability is assigned to the occurrence of an as-of-yet unknown $SRG_?$, with $|SRG_?| > 1$ and $SRG_? \notin \{SRG_1, SRG_2, ..., SRG_n\}$.

$$P(SRG_?|L_i) = 1 - \sum_{j=1}^{n} P(SRG_j|L_i) - P(\{L_i\}) \quad \geq 0 \qquad \forall i \in 1,...,m$$

In this case, $P(SRG_?|L_i)$ can be used to indicate confidence in the SRG inference table. A larger $P(SRG_?|L_i)$ leads to a lower confidence of correctness of the inference (for a certain L_i). For example, $P(SRG_?)$ can be initialized to a certain value and then lowered as more observations about L_i and its SRGs are available in failure history for the inference algorithm to base on. We will assume $P(SRG_?|L_i) = 0$ to simplify the discussion.

The SRG table contains conditional probabilities that can be used to predict directly for a single link failure. However, if an SRG failure occurs, link failures will be detected one by one. The set of link failure continues to grow. Conditional probabilities for multiple link failures can be calculated starting from the information present in the table. These multiple link failure conditional probabilities have the form:

$$P(SRG_j|L_1 \cap L_2 \cap ... \cap L_s) \qquad s \leq |L|$$

These can be derived from known information in the SRG table using the chain rule, with the following assumptions:

$$P(L_i|SRG_j) = 1 \qquad \forall i,j : L_i \in SRG_j \tag{6.1}$$

$$P(SRG_j|L_i) = 0 \qquad \forall i,j : L_i \notin SRG_j \tag{6.2}$$

For example, for a conditional probability for a double link failure $\{L_1, L_2\}$:

$$P(SRG_j|L_1 \cap L_2) = \frac{P(SRG_j \cap L_1 \cap L_2)}{P(L_1 \cap L_2)} \tag{6.3}$$

$$= \frac{P(L_1|SRG_j \cap L_2)P(SRG_j|L_2)P(L_2)}{P(L_1|L_2)P(L_2)} \tag{6.4}$$

$$= \frac{P(SRG_j|L_2)}{P(L_1|L_2)} \tag{6.5}$$

$$= \frac{P(SRG_j|L_2)}{P(\bigcup_{k}^{L_1 \in SRG_k} SRG_k|L_2)} \tag{6.6}$$

$$= \frac{P(SRG_j|L_2)}{\sum_{k}^{L_1 \in SRG_k} P(SRG_k|L_2)} \tag{6.7}$$

In (6.3)(6.4), we use the chain rule for conditional probabilities. (6.5) uses (6.1). Finally, we can state that failure of L_1 occurs under condition L_2 whenever failure of an SRG containing L_1 occurs. This basically means we trust the completeness of the SRG inference table, in that no L_2 failures are assumed to happen outside of the known SRGs. Note that for the probability that any of the SRGs containing L_1 or L_2 occur:

$$P(\bigcup_{j}^{\{L_1,L_2\} \cap SRG_j \neq \emptyset} SRG_j|L_1 \cap L_2) = \frac{\sum_{j}^{\{L_1,L_2\} \cap SRG_j \neq \emptyset} P(SRG_j|L_2)}{\sum_{k}^{L_1 \in SRG_k} P(SRG_k|L_2)}$$

$$= \frac{\sum_{j}^{\{L_1\} \cap SRG_j \neq \emptyset} P(SRG_j|L_2)}{\sum_{k}^{L_1 \in SRG_k} P(SRG_k|L_2)}$$

$$= 1$$

By applying the chain rule differently and isolating $P(L_2|SRG_j)$ instead of $P(L_1|SRG_j)$, we can also write:

$$P(SRG_j|L_1 \cap L_2) = \frac{P(SRG_j|L_1)}{\sum_{k}^{L_2 \in SRG_k} P(SRG_k|L_1)} \tag{6.8}$$

Note that in both (6.7) and (6.8) the summation is essentially done over the same SRG_k. In both equations, we normalize an initial link-conditional probability over the other-link-conditional probability of any SRG_k failure. Identity of (6.7) and (6.8) is indeed established when completeness of the SRG inference table is assumed. For example, assume the table is filled in using simple counting of SRG failure events and calculating the conditional probabilities as such from their associated event's relative frequency. With n_T the total number of SRG

events, n_{T_k} the number of SRG_k events, n_{SRG_j} the number of SRG_j events, we have:

$$P(SRG_j) = \frac{n_{SRG_j}}{n_T} \tag{6.9}$$

$$P(L_i) = \frac{\sum_{i}^{L_i \in SRG_i} n_{SRG_i}}{n_T} \tag{6.10}$$

and for the conditional probabilities, depending on whether SRG_j includes L_i or not:

$$P(SRG_j|L_i) = 0 \qquad \forall L_i \notin SRG_j \tag{6.11}$$

$$\begin{aligned} P(SRG_j|L_i) &= \frac{P(SRG_j \cap L_i)}{P(L_i)} \\ &= \frac{P(SRG_j)}{P(L_i)} \\ &= \frac{\frac{n_{SRG_j}}{n_T}}{\frac{\sum_{i}^{L_i \in SRG_i} n_{SRG_i}}{n_T}} = \frac{n_{SRG_j}}{\sum_{i}^{L_i \in SRG_i} n_{SRG_i}} \qquad \forall L_i \in SRG_j \end{aligned} \tag{6.12}$$

Therefore, starting from (6.7):

$$\begin{aligned} P(SRG_j|L_1 \cap L_2) &= \frac{P(SRG_j|L_2)}{\sum_{k}^{L_1 \in SRG_k} P(SRG_k|L_2)} \\ &= \frac{n_{SRG_j} \Big/ \sum_{i}^{L_2 \in SRG_i} n_{SRG_i}}{\sum_{k}^{\{L_1,L_2\} \subset SRG_k} \left(n_{SRG_k} \Big/ \sum_{i}^{L_2 \in SRG_i} n_{SRG_i} \right)} \\ &= \frac{n_{SRG_j}}{\sum_{k}^{\{L_1,L_2\} \subset SRG_k} n_{SRG_k}} \end{aligned} \tag{6.13}$$

as the terms of the summation in the denominator are 0 for $L_2 \notin SRG_k$. Obviously starting from (6.8) we reach the same result, which is symmetric in L_1, L_2.

Similarly, we can derive conditional probabilities for SRG_j occurring if failure of a set S of multiple links is detected.

$$S = \{L_1, L_2, ...L_s\} \tag{6.14}$$

$$P(SRG_j|L_1 \cap L_2 \cap ... \cap L_s) = \frac{n_{SRG_j}}{\sum_{k}^{S \subset SRG_k} n_{SRG_k}} \tag{6.15}$$

In fact, S itself is an SRG (and so is $\{L_1, L_2\}$ from the previous discussion). It is easily seen that when $S = SRG_j$, then $P(SRG_j|L_1 \cap L_2 \cap ... \cap L_s) = 1$ in (6.15). n_{SRG_j} are not available from the SRG inference table, however the conditional probability for a failure $(L_1 \cap L_2 \cap ... \cap L_s)$

consisting of joint failure of the links in S, can also be derived again from $P(SRG_j|L_i)$ using the chain rule:

$$\begin{aligned} P(SRG_j|L_1 \cap L_2 \cap ... \cap L_s) &= \frac{P(SRG_j \cap L_1 \cap L_2 \cap ... \cap L_s)}{P(L_1 \cap L_2 \cap ... \cap L_s)} \\ &= \frac{P(L_1|SRG_j \cap L_2 \cap ... \cap L_s)P(SRG_j|L_2 \cap ... \cap L_s)P(L_2 \cap ... \cap L_s)}{P(L_1|L_2 \cap ... \cap L_s)P(L_2 \cap ... \cap L_s)} \\ &= \frac{P(SRG_j|L_2 \cap ... \cap L_s)}{P(L_1|L_2 \cap ... \cap L_s)} \\ &= \frac{P(SRG_j|L_2 \cap ... \cap L_s)}{\sum\limits_{L_1 \in SRG_k}^{k} P(SRG_k|L_2 \cap ... \cap L_s)} \end{aligned} \tag{6.16}$$

Recursive application of (6.16) allows calculating $P(SRG_j|L_1 \cap L_2 \cap ... \cap L_s)$ in terms of the initial $P(SRG_j|L_i)$.

6.2 Failure detection and accuracy

The OSRF SRG inference relies on the detection of link failures. A clustering algorithm allows for grouping such failures into an SRG as a set of link failure. The basic information model for exchanging information such as detected failures and the SRG inference table itself is based on the link failure primitive. However, OSPF in itself does not detect link failures, as it will simply update its topology information after processing of incoming router LSAs and then recompute the shortest-path tree.

Detection of a link failure may be straight-forward for local links, if they are monitored and reporting hardware alarms, these alarms are escalated to the OSPF protocol. In the case of XORP, an interface going up or down is reported to the OSPF module and leads to detection (initiation or scheduling of a recompute) within some tens of milliseconds. On the other hand, non-local link failure must be detected through correlation of incoming LSAs and the existing link-state database, as described in deliverable D2.3. In this case, (router) LSA generation and therefore failure detection rely on the OSPF HELLO protocol. This means that link failure detection time is dictated by the HELLO message timer, namely, the time interval between two HELLO messages, and the (Router)DeadInterval which indicates the number of seconds before a router declares a routing adjacency as down (as it stops hearing adjacent router's HELLO messages).

In Figure 6.2, we show an example of remote link failure detection for a simple topology. The remote routers send HELLO messages over the link that will be failing. These routers detect the failure when a number of HELLO messages have not been received. This also means that instants in time where detection can occur are limited by the expected arrival times of HELLO messages from the adjacent router. For example, if the HELLO interval is 1 s, then a link failure will only be detected once per second, and always an integer number of seconds (an integer multiple of the HELLO interval) *after* reception of the last HELLO message before the failure occurred. For example, as shown on the figure, for a 'router dead interval' of 2 s, a link failure is detected after two missing HELLO messages. This allows for the actual link failure to occur anywhere within a time window stretching from the reception of the last HELLO message, and the (not received) next message.

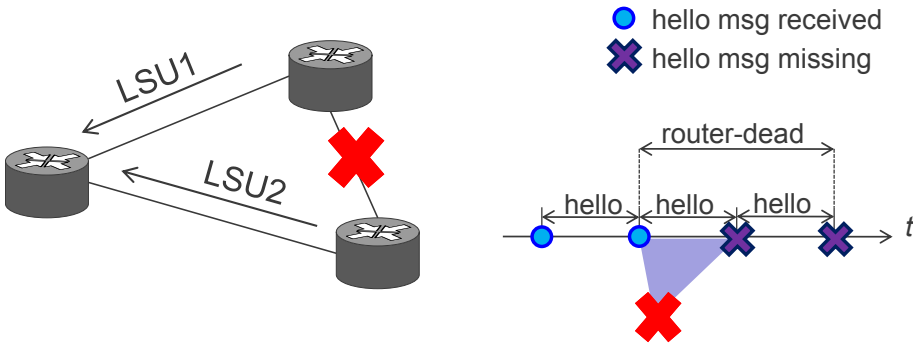


Figure 6.2: HELLO based failure detection

measurement	time
Δ_{LSA} (avg.)	1.544 s
Δ_{LSA} (st.dev.)	0.414 s
$\lvert T_{LSA_1} - T_{LSA_2} \rvert$ (avg.)	0.997 s
$\lvert T_{LSA_1} - T_{LSA_2} \rvert$ (st.dev.)	1.5 ms

Table 6.1: HELLO based failure detection times

As expected, there will be some variance to the time between actual link failure, and its detection through HELLO messages (and its subsequent detection at remote nodes through link-state updates). For the HELLO interval = 1 s and router dead interval = 2 s, Table 6.1 shows average and standard deviation of the delay Δ_{LSA} between physical failure and remote detection. The average is more or less as can be expected, since the minimum detection time is slightly large than HELLO interval (1 s), and the maximum time set to the RouterDeadInterval (2 s), as shown on Figure 6.2. The detection time also includes OSPF processing time. We also show the difference $\lvert T_{LSA_1} - T_{LSA_2} \rvert$ between arrivals of LSAs from both ends of the failing link. These depend on the exact time of HELLO messages in both endpoints of the link, which is fairly stable but largely arbitrary in a general OSPF topology.

6.3 Recovery timing

A proof-of-concept setup was demonstrated [97, 96] that uses only simple clustering and a non-overlapping SRG scenario. A comparison of recovery times for normal OSPF and SRG inference enhanced OSPF were shown by routing a number of video-streams over the network. As shown on the screenshot (Figure 6.3), this comparison was easily understandable by the audience as each video stream was rendered as transported by standard OSPF (left half of stream) and SRG inference enhanced OSPF (right half). The topology display showed current usage of bandwidth on links part of the OSPF area as well as the clustering of SRGs.

Apart from the visual demonstration however, the timing behavior was also examined using packet traces, in order to determine the duration of interruption of the video streams. Framerate of the videos is 30 fps, which resulted in (bursts of) packets at 30 Hz, which gives up a resolution of 33ms for timing results.

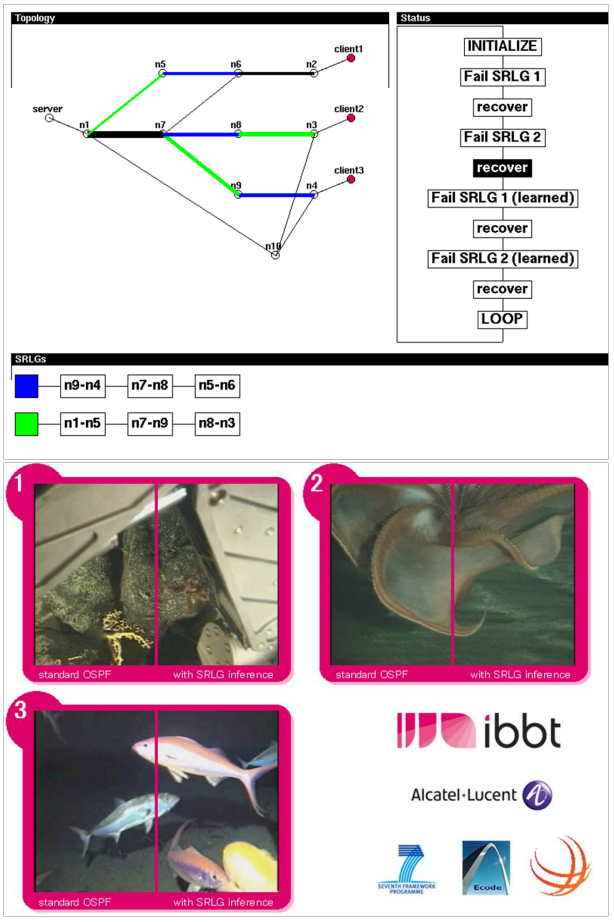


Figure 6.3: Proof-of-concept demonstration screenshot

Figure 6.4 shows packet traces obtained when applying the proposed technique to recover three video streams. The streams are routed over different paths, visiting a set of spread-out links in two non-overlapping SRGs. Interruption times for normal OSPF (top three traces) and SRG inference augmented OSPF (bottom three traces) are shown, indicated as an interruption in the corresponding trace line over time.

Four failure scenarios are executed. First, both SRGs are failed and recovered (i.e., the OSPF adjacencies are interrupted and then re-established) once. This allows the SRG inference algorithm to learn the existence of these SRGs. Initially, both cases show normal OSPF re-convergence; the three streams are recovered one by one as the failures in each of the paths links are detected. The exact interruption duration for each stream depends on timing of the OSPF HELLO messages, and therefore varies per link, and also per stream, since each stream is routed over a different link as far as the node doing the SRG inference is concerned. Note that the SRG2 link corresponding with end-to-end connectivity for stream 1 was set up to be monitored locally through a hardware alarm, yielding faster detection than normal HELLO protocol based neighbor discovery seen for the other streams. Next, the SRGs are both failed again, to show the improvement offered by the SRG inference. For the SRG inference scenario (bottom traces) the streams are now recovered simultaneously as the first LSA caused by failure of one of the SRG links arrives. Furthermore, when locally detecting one of the failing links in

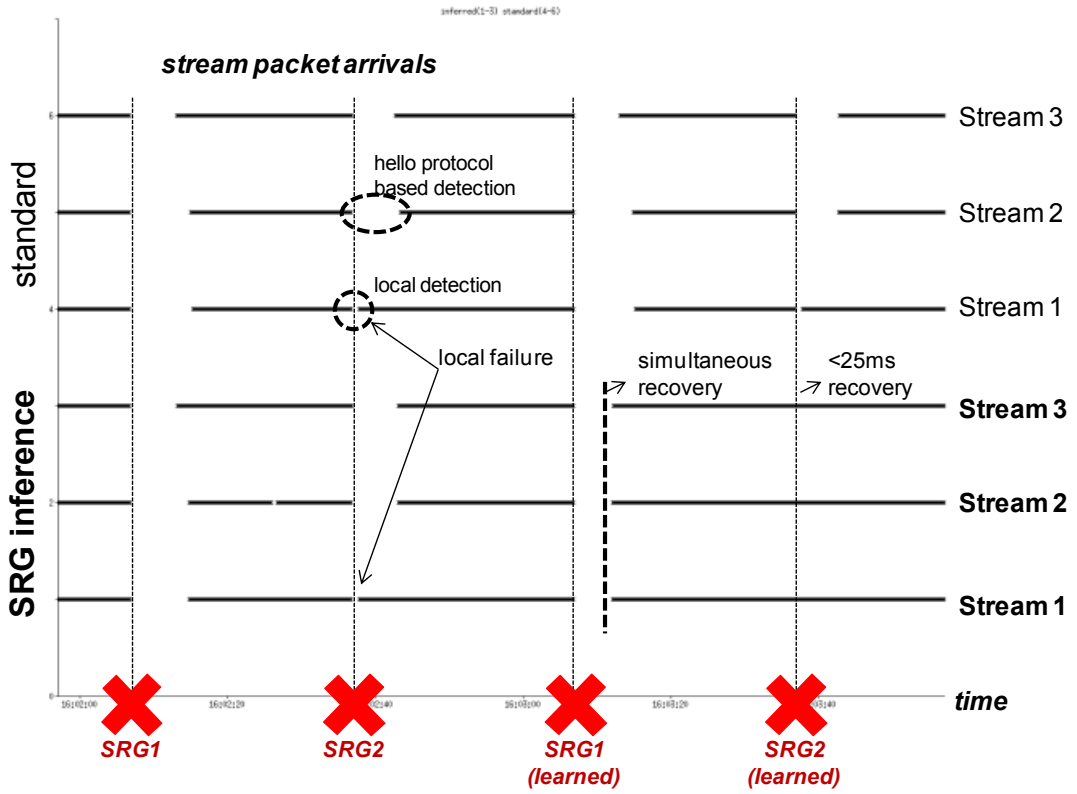


Figure 6.4: Packet traces

a SRG, as is the case for SRG2, all SRG links and thus all streams benefit from fast detection, which allows for up to sub 25 ms recovery in the demonstration setup. Figure 6.5 shows the traces for learned SRG2 on a shorter timescale. Incidentally, we see an interruption time which is limited to loss of a single frame in the video stream. Local detection is used also in the standard OSPF case (trace 4); however, in this case the normal recompute hold-off time is maintained (see D2.3), so that recovery is delayed by 1 s. Recovery of the other streams then takes much longer as remote HELLO based detection is still used.

6.4 Scaling of detection times

The previous section showed some sampled timing results for detection of a single link, which, as far as OSPF is concerned, appears as an SRG of two forwarding adjacencies (two opposite directed edges corresponding with the single link. Failure of the adjacencies is detected and reported by the two separate advertising routers.

With failure detection that relies on HELLO protocol messages, the exact detection time depends on the timing of those HELLO packets. These should have the same interval in the entire OSPF area (they are sent at the same rate for all adjacencies), but there is a time offset to the HELLO packet generation for the respective adjacencies. Once an SRG is learned, a single failing SRG link may suffice to infer that SRG to be failing. The expected detection time depends on the size of the SRG.

Taking an arbitrary moment t_0 in time as calibration, the offset h for HELLO protocol mes-

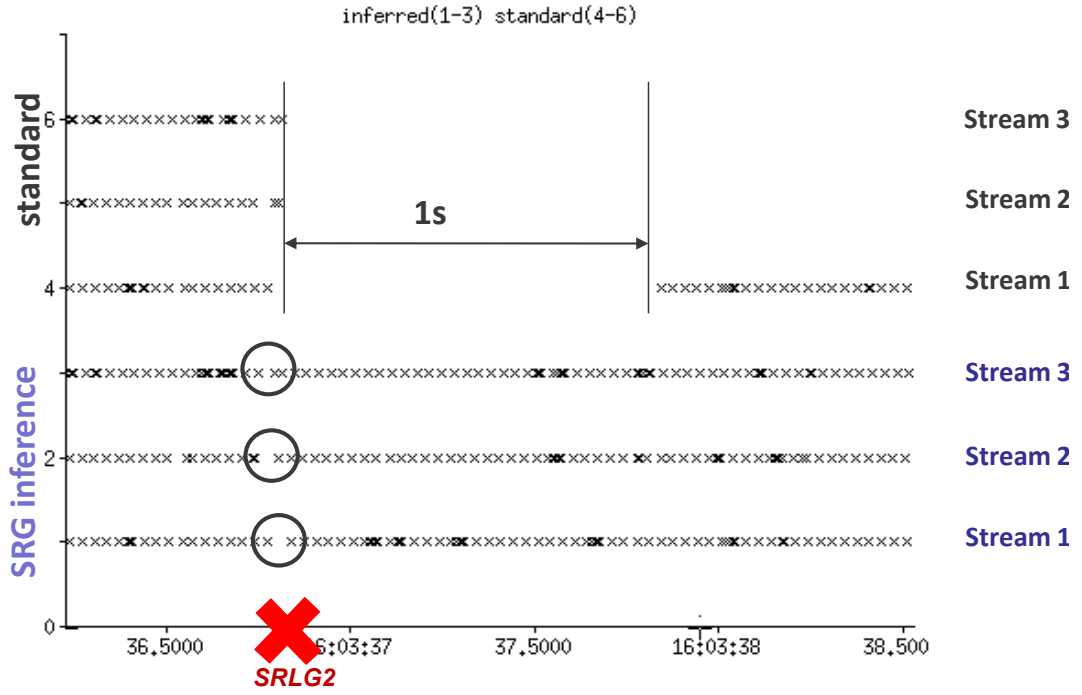


Figure 6.5: Packet traces, detail for learned convergence, local detection

sage generation has a certain distribution $p_h(t)$ (e.g., typically uniformly). By convention, this offset is at most the HELLO Interval H, i.e., falls within the interval $[0,H[$.

$$p_h(t) = 0 \qquad t \notin [0,H[\tag{6.17}$$

$$p_h(t) > 0 \qquad t \in [0,H[\tag{6.18}$$

$$\int_{-\infty}^{+\infty} p_h(t) = 1 \tag{6.19}$$

Detection occurs when two consecutive HELLO messages are not received (see Figure 6.2). This means detection time Δ (time between failure t_F and detection t_D) for a failure at $t_0 + nH$ is given by:

$$\begin{aligned} \Delta &= t_D - t_F \\ &= (t_0 + nH + H + h) - (t_0 + nH) \qquad = H + h \end{aligned} \tag{6.20}$$

As also shown on Figure 6.6. Note that the HELLO message offsets are basically time calibrated on the failure time t_F; in the case of a uniform distribution of the offsets, this has no impact on $p_h(t)$.

For the distribution $p_\Delta(t)$ of the detection time Δ, this yields:

$$p_\Delta(t) = H + p_h(t) \tag{6.21}$$

From this, the statistical properties can be derived for the detection time of a single adjacency failure, e.g., for a uniform distribution of $h(t)$, average detection time is $\mu_\Delta = 1.5H$.

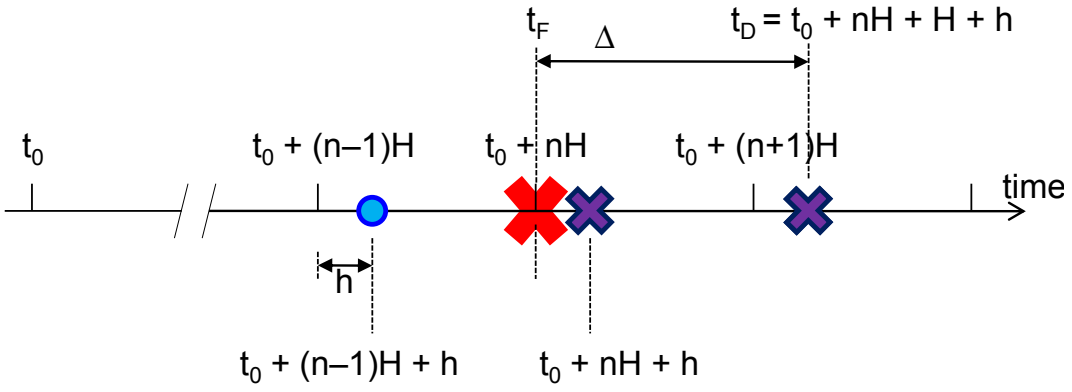


Figure 6.6: Detection time *Delta*

For a larger SRG, detection time is Δ of the first detected link. The detection time will be lower as more links serve as detection for the SRG. This can be seen by deriving the distribution of detection time Δ_n for a SRG consisting of n entities (adjacencies). For this purpose, we first calculate the cumulative distributed function $P_{\Delta_1}(t) = P_\Delta(t)$:

$$\begin{aligned} P_\Delta(t) &= \int_0^{2H} p_\delta(t)dt \\ &= \int_H^{2H} p_h(t)dt \end{aligned} \tag{6.22}$$

In case of a uniform distribution:

$$P_\Delta(t) = \begin{cases} 0 & t < H \\ \frac{t-H}{H} & t \in [H, 2H[\\ 1 & t >= 2H \end{cases} \tag{6.23}$$

$P_\Delta(t)$ corresponds with the probability that $\Delta \leq t$. The probability that the detection time Δ_n for n elements is similarly limited by t can be derived from $P_\Delta(t)$. The n elements have

respective detection times $\Delta_1, \Delta_2, ...\Delta_n$.

$$\begin{aligned} P_{\Delta_n}(t) &= \min_{i=1}^{n} D_i < t \\ &= P(\Delta_1 \leq t \cup \Delta_2 \leq t \cup ... \cup \Delta_n \leq t) \\ &= 1 - P(\Delta_1 > t \cap \Delta_2 > t \cap ... \cap \Delta_n > t) \\ &= 1 - \prod_{i=1}^{n} P(\Delta_i > t) \\ &= 1 - \prod_{i=1}^{n} (1 - P(\Delta_i \leq t)) \\ &= 1 - (1 - P_\Delta(t))^n \end{aligned} \tag{6.24}$$

With the assumption that the n elements have independent detection times. And with (6.23):

$$P_{\Delta_n}(t) = \begin{cases} 0 & t < H \\ 1 - (1 - \frac{t-H}{H})^n & t \in [H, 2H[\\ 1 & t > 2H \end{cases} \tag{6.25}$$

So for the distribution $p_{\Delta_n}(t)$ of the detection time of the entire n-element SRG:

$$\begin{aligned} p_{\Delta_n}(t) &= \frac{d}{dt} P_{\Delta_n}(t) \\ &= n(1 - P_\Delta(t))^{n-1} p_\Delta(t) \end{aligned} \tag{6.26}$$

So, for (6.25), and $p_\Delta(t) = 1/H$ for $t \in [H, 2H[$:

$$p_{\Delta_n}(t)|_{t \in [H,2H[} = \frac{n}{H}(1 - \frac{t-H}{H})^{n-1} \tag{6.27}$$

For example, the expected average detection time for n elements is therefore:

$$\begin{aligned} \mu_{\Delta_n} &= \int_{-\infty}^{+\infty} t.p_{\Delta_n}(t)dt \\ &= \frac{n}{H} \int_H^{2H} t(1 - \frac{t-H}{H})^{n-1} dt \\ &= \frac{n+2}{n+1} H \end{aligned} \tag{6.28}$$

E.g., $\mu_{\Delta_1} = 1.5H$ as earlier. Similarly, for σ_{Δ_n}:

$$\begin{aligned} \sigma_{\Delta_n} &= \int_{-\infty}^{+\infty} (t - \mu_{\Delta_n})^2 p_{\Delta_n}(t)dt \\ &= \frac{n}{H} \int_H^{2H} (t - \frac{n+2}{n+1}H)^2 (1 - \frac{t-H}{H})^{n-1} dt \\ &= \frac{\sqrt{n+2}}{(n+1)(n+2)} \sqrt[3/2]{H} \end{aligned} \tag{6.29}$$

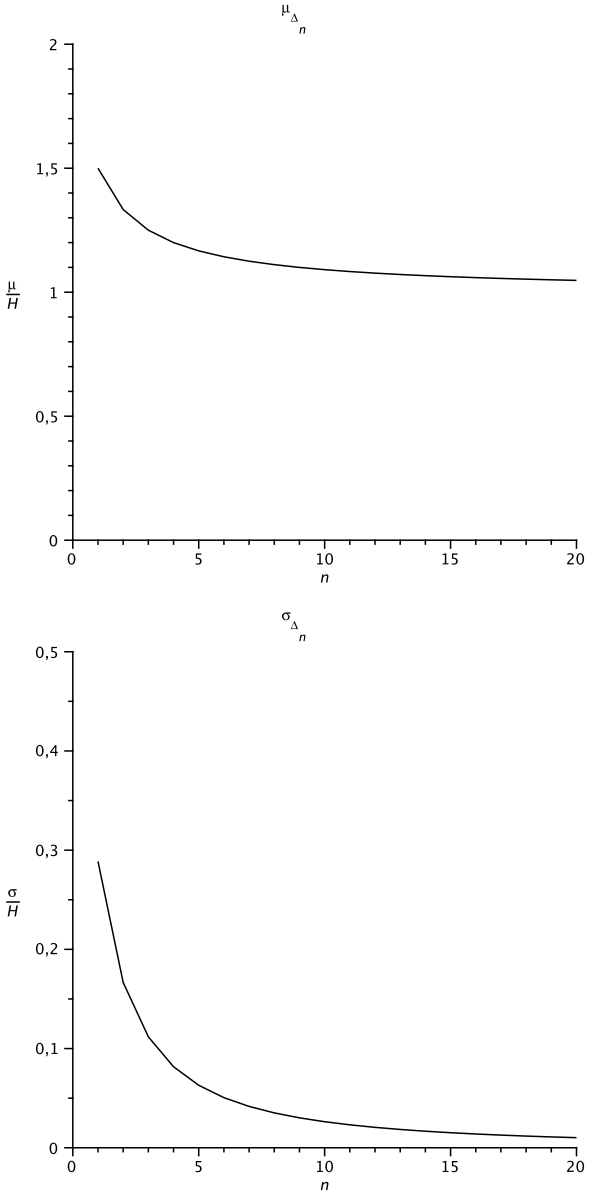


Figure 6.7: Detection time Δ_n for larger SRGs (n elements)

As can be seen from Figure 6.7, the average detection time approaches $1H$ (from $1.5H$) for SRGs consisting of a large set of elements (adjacencies). An important aspect is that, as σ_{Δ_n} converges to 0 quite quickly, the detection times become predictable for large SRGs. It should be noted that if one of the elements in the SRG is detected locally, then the detecting router will initiate the flooding of a trigger LSA that will in turn lead to fast recovery of the SRG in all other nodes as well.

Also, (6.28) and (6.29) assume the HELLO message generation offsets to be independent random variables. This is true when looking at, e.g., the average detection time of random SRGs. It is also true when looking at the average detection time of a particular SRG, considered over a long period of time. In this case, intermediate OSPF reconfiguration inter-node clock drift

and similar effects will make sure that the relation between HELLO offsets *of the elements in that SRG* will be weak.

When looking at repeated failures of a single SRG in a short period of time, then the mutual HELLO offsets of the SRG's elements will be more or less fixed. The detection time statistical properties will depend on the distribution of these HELLO offsets. For example, is the offset is 0 (all HELLO messages in the SRG element set are generated at the same time), then – statistically– it is as if there is only one element, so detection times will be those for $n = 1$.

Depending on the implementation, the mutual offset for the anti-parallel adjacencies that make up a single link may be fixed as well. In this case (assuming an SRG consisting of L bidirectional links), we should use (6.28)(6.29) not with the number of adjacencies $n = 2L$, but with $n/2 = L$.

Note that in the case of a failing node (where all L incident links make up the SRG), we must rely on the neighboring routers to detect its failure since the failing node is now disconnected from the OSPF area (to which it used to belong before its failure) and neighboring routers do not receive LSA from the failing node anymore. This means that, although the HELLO offsets of the outgoing adjacencies may be interrelated, the L incoming adjacency HELLO offsets are the ones that are relevant for detection timing.

6.5 Scalability of the SRG table

The SRG table is serialized for sending to the OSPF module. The minimum required information for identification of an adjacency (or failure) is the OSPF Link ID, Link Data and link type. ID and Data are 32-bit IP prefixes. Type can be encoded as a single byte. This means information for an adjacency requires at least 9 bytes. Each SRG is encoded as a set of links. As the set of links is first serialized in an array and sent to the OSPF module, the SRGs use indexes into this array. Using a byte-sized index, we can index 256 adjacencies (the OSPF area may have more adjacencies, as long as only 256 appear in an SRG). The conditional probabilities that make up the elements of the SRG table are encoded as 32-bit floating point numbers.

The set of SRG will generally at least contain the singleton link SRGs for each link in the OSPF area — note that these are actually then SRG sets of the two anti-parallel adjacencies that make up the link. Also, we can expect the OSPF nodes to each act as an SRG. Additional SRGs may appear because multiple links in the OSPF are being routed as multi-hop connections in an lower layer. A worst case scenario would be a full mesh OSPF area established as over a physical ring network.

A full mesh network of n nodes has $1/2 \times n \times (n-1)$ links (and twice as many adjacencies). The physical ring network has n links, each of which will form an SRG of the OSPF area links routed over it. With L and L_{ring} the set of OSPF area links and physical links respectively, N the set of nodes, S the set of SRGs:

$$\begin{aligned} |L| &= \frac{n(n-1)}{2} \\ |L_{ring}| &= n \\ |N| &= n \\ |S| &= 2n + \frac{n(n-1)}{2} \end{aligned}$$

The size of the node SRGs is $n-1$. The size of a singleton link SRG is 1. Finally, the size of a physical link SRG is about $1/2 \times |L|$. This gives, for the memory required for encoding the table:

$$
\begin{aligned}
M_{links} &= \frac{n(n-1)}{2} \times 9B \\
M_{SRGs} &= \frac{n(n-1)}{2} \times 1B && \tfrac{n(n-1)}{2} \text{ singleton SRGs} \\
&+ n(n-1) \times 1B && n \text{ node SRGs} \\
&+ n\frac{n(n-1)}{4} \times 1B && n \text{ physical link SRGs} \\
M_{probabilities} &= \frac{n(n-1)}{2}(2n + \frac{n(n-1)}{2}) \times 4B
\end{aligned}
$$

Adding everything up, we have:

$$M_{tot} = \left[n^4 + \frac{9}{4}n^3 + \frac{11}{4}n^2 - 6n \right] \times 1B \tag{6.30}$$

Note that generally, a single link consists of two adjacencies, so the SRGs are in fact twice as large, containing both anti-parallel adjacencies. The node SRGs however do not double in size since a failing node does not report its side of its incident links, these are not observed and clustered. They are not pruned, but instead still become unavailable as the anti-parallel adjacencies *are* pruned, so the failing node (and its outgoing adjacencies) becomes disconnected from the topology. Finally, the number of links (rows) also doubles. This gives for actual total memory when considering adjacencies and not links:

$$M_{tot,adj} = \left[4n^4 + \frac{1}{2}n^3 + \frac{13}{2}n^2 - 11n \right] \times 1B \tag{6.31}$$

So, the SRG table size in bytes is $O(n^4)$. For $n = 100$ nodes, we have a 100+ MB SRG table. Generally however, only SRGs that have been observed will be in the SRG table. If the table still remains too large, it can be truncated. This means that SRGs with a low occurrence (and therefore low inferred probability) can be omitted from the table. This may also cause some links (rows) to be removed from the table, as they no longer are part of any of the SRGs reported in the table.

6.6 Scalability of multiple failure inference

The recursive application of (6.16) allows calculating conditional probabilities of the SRGs for any combination of failing links. This allows us to do inference for a currently observed multiple link failure, even although the SRG table only contains these probabilities under single link failures.

The algorithm 1 shows how this recursive application is performed. The algorithm starts of with any link L_{start} part of the set L of multiple failures. If L_{start} is the only link in L, the probabilities $P(SRG_j|L_{start})$ can be taken from the SRG table. If not, the probabilities stored in p_j (for each SRG_j) are normalized for each of the remaining link in the failure set. Note

Algorithm 1 Inference for multiple failures

L $\leftarrow$ set of failing links
$L_{start} \leftarrow$ any failing link $\in L$
$p_j = P(SRG_j|L_{start})$
$R \leftarrow L \setminus L_{start}$ {remaining links}
$D \leftarrow \{L_{start}\}$ {done links}
for all $L_i \in R$ **do**
  $D \leftarrow D \cup L_i$
  for all $j : D \subset SRG_j$ **do**
    $t_j = p_j / \sum_{k}^{L_i \in SRG_k} p_k$
  end for
  $p_j = t_j$ {conditional probabilities for links in D}
  $R \leftarrow R \setminus L_i$
end for
$P(SRG_j|L) = p_j$

that with each iteration, the set of j in the inner loop diminishes as the condition becomes more stringent (D grows with each normalization).

In a general case, with $|S|$ the number of SRGs and $|L|$ the number of links in the failure set, a single normalization requires one division and a number of additions in the order $|S|$. The number of normalizations is of order $|S|$. $|L|-1$ sets of normalizations are required. The total number of operations is therefore of order:

$$N_{op} = O(|S|^2 \times (|L|-1)) \tag{6.32}$$

For our previous example of full mesh routed over a physical ring, the number of operations N_{op} (n = number of nodes):

$$\begin{aligned} N_{op} &= O\left((2n + \frac{n(n-1)}{2})^2 \times (\frac{n(n-1)}{2} - 1)\right) \\ &= O(n^6) \end{aligned} \tag{6.33}$$

Chapter 7

Automated Learning of Loop-Free Alternate Paths for Fast Re-Routing

7.1 Introduction

Connection-less IP networks independently decide how to forward received packets or datagrams. The information determining how they forward these packets (i.e., which outgoing interface and next hop they will take) is stored in their local Forwarding Information Base (FIB). The FIBs comprise (forwarding) entries that are derived from the information exchanged by link-state routing protocols such as Open Shortest Path First (OSPF, [84]). These protocols discover the local topology (link states) and distribute the discovered information over the network using Link State Update messages. As a consequence, every router can independently compute the shortest routing paths towards other nodes in the network. Exchanges of link state routing information resulting from topology changes (dynamic reaction to topological changes due to, e.g., link/node failures) lead to the re-computation of the routing paths and reconfiguration of the corresponding FIB entries (re-convergence), as well as the update of the corresponding routing and forwarding entries (note that these steps outline the IGP re-convergence process). However, as every router performs the routing path computation independently of other routers, transient (micro-)loops may be formed during the periods when a network is re-converging due to inconsistent FIB entries. This problem is inherent to any asynchronous distributed routing protocol and caused by inconsistent FIB entries resulting from the propagation time of the routing updates as well as the time needed to re-compute and distribute FIB entries. Packets which are trapped into transient loops, never reach their destination and are simply lost after TTL expiration. Prior work [45] has demonstrated that these loops can take hundreds of milliseconds. Therefore, our goal is to minimize the re-routing time: the time needed for each node, after the occurrence of a topological change, to use updated FIB entries -without relying on the full IGP[1] re-convergence- along loop-free alternate paths for the maximum number of destinations. The three-fold objectives of the paper (and of fast-rerouting techniques in general) are:

- Maximize the percentage of links (or nodes) that can be fully protected (i.e., for all destinations)

[1]The term Interior Gateway Protocol (IGP) refers to any link state routing protocol running within a routing system.

- Maximize the percentage of destinations that can be protected for all link (or node)
- Minimize the stretch increase on the routing paths between source and destination.

The proposed fast-rerouting technique relies on the avoidance of transient loops by detecting them before failure occurrence. More precisely, it operates following three main steps. Initially, each node determines its loop domain with respect to other (destination) nodes. The loop domain is determined by the set of nodes for which the loop-free neighbor criteria is not verified along certain alternate routing paths before occurrence of topological change (when traffic forwarded by node u and directed to destination t arrives at node v that forwards this traffic along a path that reaches node u, i.e., v is a not loop-free neighbor of u). Then, the detecting node selects an alternate routing path that ensures loop-freeness up to loop domain boundaries by instantiating an alternate forwarding entry on each intermediate node (pointing to the loop-free neighbor). Upon failure occurrence, the node triggers that loop-free alternate path (when traffic from u directed to t arrives at v, v does not forward traffic along a path that reaches node u, i.e., v becomes a downstream neighbor of u).

This Chapter is structured as follows. Section 7.2 provides an overview of the current fast re-routing techniques and positions the contribution brought by our approach. Section 7.3 provides a detailed description of the proposed technique including a learning approach finding nodes out of the loop domain of each node and the computation of the loop-free alternate path. Section V details the experimental results we have obtained by simulation when running these procedures on topologies representative of core networks to which the proposed technique would typically apply. Finally, Section VI formulates our conclusions and some suggestions for future work.

7.2 Related work

Fast re-routing (or fast repair) techniques can be classified into the following three basic categories (see [106]).

7.2.1 Equal cost multi-paths (ECMP)

ECMP [54] can be used when a set of two or more paths towards the same destination d is available. Assuming one of them doesn't traverse the failure, that alternate path can be used as repair path.

7.2.2 Loop-free alternate (LFA) paths

A Loop-Free Alternate path [13] exists when a direct neighbor of the router adjacent to the failure has a path to the destination that can be guaranteed not to traverse the failure (loop-free neighbor condition). The average coverage on common networks (that is strongly dependent on the topology) shows variations from 60 to 90 percent. Indeed, when a link or a node fails, only the nodes incident to the failure are initially aware that the failure has occurred and only these nodes can re-route traffic along paths round the failure. These repairing routers have to steer datagrams to their destinations despite the fact that most other routers in the network are unaware of the nature and the location of the failure. A common limitation in most of the base LFA mechanisms is an inability to indicate the identity of the failure and to explicitly steer the

repaired datagram round the failure. Consequently, the extent to which this limitation affects the repair coverage is topology dependent.

An advanced LFA solution [44] consists in sequencing the FIB updates either spatially (topologically ordered FIB update from far-end to the near-end neighbor contiguous to the failure) or temporally (timely synchronized FIB updates). For instance, ordered FIB update provides 100 percent loop-free convergence at the expense of a FIB update time proportional to $R \cdot MAX_{FIB}$, where R is the max (hop) length among paths to edge r used to reach destination t (downstream SPF neighbor prior to the failure), and MAX_{FIB} is a network-wide constant that reflects the maximum time T_{max} required to update a FIB irrespective of the change required. Hence, time performance degrades proportionally to the path length, i.e., FIB updates are actually committed at the near-end after reception of a completion message traveling back from the source of max (hop) length among path to edge r used to reach destination t. This solution is not considered outside network maintenance operation as it suffers from slow activation.

7.2.3 Multi-hop repair paths

When there is no feasible loop-free alternate path it may still be possible to locate a router, which is more than one hop away from the router adjacent to the failure, from which traffic will be forwarded to the destination without traversing the failure. Multi-hop repair paths are more complex both in the computations required to determine their existence, and in the mechanisms required to invoke them. Multi-hop repair paths techniques can be further classified as:

- Mechanisms where one or more alternate FIBs are pre-computed in all routers, and the repaired datagram is instructed to be forwarded using a "repair FIB" by some method of per-datagram signaling involving, e.g., the detection of a "U-turn" [12].
- Mechanisms functionally equivalent to loose source routing that is invoked using the normal FIB. These include tunneling-based approaches [21] that consist in "by-passing" the topology change by pre-configuring tunnel whose path is not affected that change. There are multiple variants of "tunnel-based solutions": single-sided (near-end or far-end), double-sided (near-end and far-end), and distributed (tunnel segments). They all suffer from the same problems: i) computational complexity, ii) tunnel pre-configuration and maintenance, and iii) impact on forwarding plane. Thus, they all involve a high degree of configuration for tunnels that in turn decrease the forwarder performance. Other mechanisms such as the Not-Via technique [22] employ special addresses that are installed in the FIBs together with pre-computed routes that avoid certain components of the network. This technique encapsulates the datagram to an address that explicitly identifies the network component that the repair path must avoid. This produces a mechanism that always achieves a repair, provided the network is not partitioned by the failure.

Hence, several fast path repair/fast re-routing techniques already exist. Some of them are used in operational networks such as base Loop-Free Alternates (LFA) and Equal Cost Multi-Path (ECMP). They all aim to address the objectives introduced in Section 7.1. In this competitive context, our approach is threefold: i) the proposed technique relies on distributed learning of the loop-domain at each node and ”best-alternate path” to a given destination[2]; ii) The

[2]both can either be performed on-line or by mining the link-state routing topology and the routing table (RT) entries

proposed re-routing scheme does not assume modification of the link-state routing protocol operations outside of the transient re-routing periods (as alternate forwarding entries take local precedence over default IGP routing entries). Once, the IGP has re-converged unflagging datagrams leads to the use of the primary path entries; iii) the coverage of the proposed re-routing scheme is almost 100 percent.

7.3 Automated learning of Loop-Free Alternates

7.3.1 Assumptions

The proposed approach aims to accelerate the re-routing of traffic along loop-free alternate routing paths in link state routing networks. Upon failure occurrence, the failure detection technique is assumed to provide local information. Failure information propagation does not rely on associated fast failure notification protocol (operating next to the link-state IGP) or IGP parameter tuning. The only condition for our approach to be operational is that the loop domain's diameter is smaller than the flooding domain of the IGP. Otherwise, the technique resumes as a best exit node selection to avoid loops inside the IGP routing domain but then relies on neighboring domains for the alternate path to re-merge with the primary path (outside the loop domain).

7.3.2 Preliminaries

The network topology is modeled by a weighted undirected graph $G = (V, E, \omega)$ with positive edge cost *omega*, where V is the set of vertices or nodes ($|V| = m$) and E is the set of edges or links ($|E| = n$). A non-negative cost function $\omega : E \rightarrow Z^+$ associates a cost u, v to each link $(u, v) \in E$. For $s, t \in V$, let $d(s,t)$ denote the cost of the path $p(s,t)$ from s to t in G, where the cost of a path is defined as the sum of the costs along its edges. We first introduce the following distinction:

- For the pair $s, t \in V, s \neq t$, if there exists a vertex u adjacent to vertex s, (i.e., edge $(s,u) \in E(G)$) such that $d(u,t) < d(s,u) + d(s,t)$, i.e., u is a loop-free neighbor of s to destination t, then the path $(v_0(=s), v_1, \ldots, v_m(=t))$ is a loop free alternate path where $\forall i : d(v_i, v_m) < d(v_{i-1}, v_i) + d(v_{i-1}, v_m)$.
- For the pair $s, t \in V, s \neq t$, if there exists a vertex u adjacent to vertex s, (i.e., edge $(s,u) \in E(G)$) such that $d(u,t) < d(s,t)$, i.e., u is a downstream neighbor of s to t, then the path $(v_0(=s), v1, \ldots, v_m(=t))$ is a distance decreasing downstream path, where $\forall i : d(v_i, v_m) < d(v_{i-1}, v_m)$. As a particular case, neighbor u of node s is the downstream SPF neighbor of s for destination t, if node u provides the shortest path to t according to a shortest-path first (SPF) routing scheme.

Note that the set of distance decreasing downstream paths is a subset of the set of loop-free alternate paths meeting the condition $\forall i : d(v_i, v_m) < d(v_{i-1}, v_m)$. We define the loop domain of node $u \in V(G)$ as the set of node $B(u)$ such that if a path $p(s, \ldots, u, \ldots, w, \ldots, t)$ traverses node u and then node w it will loop back via node u before reaching destination t, i.e., w does not sit along a loop-free alternate path to destination t from node u.

7.3.3 Steps and mechanisms

The proposed fast re-routing approach (ALFA) comprises three main steps:

- Step 1: each node u determines its loop domain $B(u)$ with respect to each destination t that it can reach (as indicated by its routing table entries). For this purpose, node u sends a probe message towards destination t on the interface directed to one of its non-shortest path from u to destination d. If the message returns to u (source of the probe message) the message didn't reach a node v located outside of the loop domain. We refer to such node v as a loop-free node (LFN).

- Step 2: determine a node v located outside the loop domain of node u for destination t and that sits along a non-shortest path towards destination t. Node v is referred to as the loop-free node (LFN) and the path $(u,\ldots,v,\ldots,t)$ as the loop-free alternate path (or more synthetically $p(u,v,t)$). Inside the loop-domain $B(u)$ of node u, along the non-shortest path that is selected as the loop-free alternate path and to which the probe message sourced at node u is forwarded, alternate forwarding entries are configured for that destination t. Indeed, the default forwarding entries at these nodes for destination t refer to a path that traverses node u. More precisely, $\forall w \in B(u)$ node w does not verify the loop-free condition, the path $p(w,t)$ includes node u, i.e., $p(w,u,t)$. When the probe message reaches node v, that message is returned to node u with the indication that no FIB entry configuration is required to reach destination t (node v verifies the loop-free condition: $d(v,t) < d(u,v) + d(u,t)$. Note that with the BFS+ technique (as documented in Section 7.3.6), the loop-free alternate path $p(u,v,t)$ is the non-shortest path that differs the most from the shortest path (considered as the primary path) before failure of a link incident to u along the primary path from u to t, $p(u,t) : v \notin p(u,t)$.

- Step 3: upon failure detection by node u (assume, e.g., the failure of one of the links incident to node u along its primary path towards destination t), the loop-free alternate path is activated. The action of activation by node u of its loop-free alternate path $p(u,v,t)$ refers to the triggering operation of the alternate forwarding entry along the loop-free alternate path inside the loop domain of node u, $B(u)$. The alternate forwarding entries are triggered from the reception of datagrams including as indication in their header that these datagrams were re-routed by node u along the loop-free alternate path. Activation of the alternate forwarding entries is performed until reaching node v. Outside of the loop-domain of node u, datagrams remain flagged but without triggering any action at the nodes traversed by these datagrams (the alternate and the primary forwarding entries are indeed identical). This condition is sufficient to guarantee that the path $p(v,t)$ followed by the datagrams leaving the loop-domain is loop-free, i.e., as long as the path $p(v,t)$ is the distance decreasing SPF downstream path to destination t (the path $p(v,t)$ does not re-enter the loop domain of node u). When exiting the local routing domain (i.e., the link state routing protocol flooding domain), the datagrams flagged by the re-routing node u are unflagged by the boundary node of the domain.

The next paragraphs of this section explain each of these steps together with a description of the corresponding procedures.

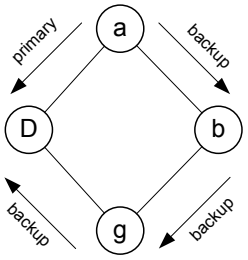


Figure 7.1: Interface-specific forwarding

7.3.4 Router Model

A router consists of a Routing Information Base (RIB) and a Forwarding Information Base (FIB). The terms RIB and routing table are used equivalently since we assume that a single routing protocol is running in each routing domain. The FIB stores forwarding entries each comprising the outgoing interface to be taken by individual datagrams for a given destination prefix. It also stores an alternate forwarding entry for any given destination prefix. The use of the alternate entry is triggered by the indication of a flag (bit) in the header of an incoming datagram (to be decided in which field), further referred to as the alternate flag. Datagrams are marked with this flag, from the moment a failure is noticed on the link towards the next hop according to the primary forwarding entry.

In our router model, the forwarding decision is also conditioned on the incoming interface, which implies that the alternate entry for a given destination prefix can be different for datagrams arriving at interface x, compared to those arriving at interface y in a given router. This interface-dependence allows us to keep using shortest path routing on the primary forwarding entries. To ensure that the alternate forwarding entries have node-wide significance, the identifier of the triggering node (that is the node that flags the datagram) should be known and stored at configuration time as part of the alternate entries and be included as part of the flagged datagram. This is illustrated in Figure 7.1. The shortest path towards node D from node a and g is via their direct link. However, using node-wide significant alternate routing entries to node D enforces them to choose whether node a or node g is on the primary path.

If the primary next hop of node u along its primary path to a given destination becomes unreachable due to a link or node failure, then i) the datagrams for that destination are flagged (as indicated before) and ii) the alternate forwarding entry for the interface corresponding to the failing link or node is chosen to forward the flagged datagrams along the alternate path. At node w, the use of the alternate forwarding entry must not result into flagged datagrams being sent back to node u (rule.1). Along the alternate path, flagged datagrams arriving from primary interface (i.e., the interface corresponding to the next hop as indicated in the primary forwarding entry) or more generally any interface if the identifier of the triggering node can be retrieved from the incoming datagram, the alternate flag will automatically trigger the use of the alternate forwarding entry to avoid looping behavior (rule.2). To avoid that the flagged datagrams loop back to node u, the proposed technique comprises a cycle-free alternate path computation technique. This technique is described in the next section.

7.3.5 Cycle-free alternate path computation

Initial FIB configuration

We initiate the Primary FIB (pFIB) of all nodes using the usual shortest-path computation techniques for (connection-less) link-state routing protocols such as OSPF or IS-IS. The alternate FIB (aFIB) stored at each node is initially a copy of the pFIB, using the same next hop for on all interfaces as the one determined by the shortest path calculation for the pFIB. With one noticeable exception: the aFIB entry corresponding to the primary forwarding entry is populated with the next hop according to the shortest path excluding the link indicated by the primary forwarding entry. We will refer to this entry as the Alternate Shortest Path entry (ASP entry). Note also that after configuration, the forwarding entries for which the primary and the alternate next-hop for the same destination are identical can be removed from the aFIB.

Alternate FIB configuration

As previously explained, once a single failure is locally detected by a given router, its incoming datagrams toward the affected destinations are flagged, and the datagrams are forwarded according to the alternate forwarding entry (the ASP entry as defined here above). However, because downstream routers still forward flagged datagrams according to their locally computed shortest path, it is likely that the flagged datagrams will be looped back to the flag-originating-node, causing a forwarding loop.

To avoid forwarding loop situations, we combine two techniques: i) the discovery of a node referred to as the loop-free node (LFN) which sits outside of the loop-domain of a given node with respect to a given destination, and the LFN is out of the loop-domain of the given node with respect to the LFN itself, and ii) the configuration of the aFIB entries along the path towards the given LFN, this path is the one referred to as the alternate path. The loop-domain of a given node u for a given destination t is defined as the set of downstream nodes (with respect to the directionality of the traffic flow towards destination t) that forward incoming datagrams received from node u along a path that traverses node u. Once flagged, the datagrams reach the LFN, the path followed according to the rest of the AFIBs lead to the destination without looping back to the original node again.

7.3.6 Loop-domain detection using BFS+

As indicated earlier, in order to ensure a loop-free alternate path from a node s towards a destination t, the former needs to find a loop-free node sitting outside of its loop-domain with respect to t. For this purpose, we devise an extended Breadth First Search method (referred to as BFS+). The recursive mechanism works as follows:

- Send a probe message towards t via all the neighbors of the node s (hop count diameter 1 from s).
- If all probe messages pass via the node s, mark the visited nodes, and repeat the procedure with the neighbors of the marked nodes (excluding the already visited nodes) until at least one probe message reaches a target node v sitting outside the loop-domain of node u without passing via source node s. Upon the arrival of the probe message at node v, the receiving node sends an acknowledgment message towards node s.

- When multiple LFNs are found within a given diameter from node s, the LFN is chosen such that the alternate routing path towards t has the lowest similarity with the primary path from s to t[3].

The BFS+ algorithm is illustrated on Figure 7.2. The loop-domain of node s is indicated with the circle with dotted lines, containing the nodes probe1 and probe2. Probe1 and probe2 are upstream nodes to s with respect to destination t. This implies that the shortest path of these nodes will always pass via node s. Because these nodes are within hop count diameter 1, BFS+ will first send probe via these nodes towards destination t. When the node s intercepts these probe messages, BFS+ triggers probe messages to be send from hub nodes on hop count diameter 2. Afterwards, the nodes probe3 and probe4 are tested. Both nodes forward the probe message to destination t without passing via node s, and thus are LFNs. However, because the path taken from node probe3 differs more from the primary path compared to the path taken from node probe4 (which uses the same last link towards destination t), probe3 is elected as the LFN by the procedure.

7.3.7 Configuration of path to LFN

Once an LFN node is elected using the previously described BFS+ technique, the loop-free alternate path towards the LFN must be configured. This operation is realized by installing the alternate forwarding entries along the alternate routing path from node s to the LFN. For this purpose, the node s sets its forwarding entry towards the LFN as its alternate entry towards destination t. The same procedure is used as indicated by the next hop(s) until the LFN is reached.

The ALFA learning procedure executes the above LFN detection and LFN path-configuration process from all nodes towards all other nodes (destination). When a node detects that the outgoing interface corresponding to the primary routing entry for a given destination is not available, based on a loss-of-signal event or a Hello timer timeout (as in OSPF), the alternate forwarding entry towards the destination is used. This procedure will bring the packet to the LFN (as it was previously configured to do so), and from then on, shortest path routing entries will bring the packet from the LFN to the destination.

7.4 Experimentation

7.4.1 Environment

A custom simulation environment was developed by means of Python/C++ libraries to evaluate the proposed cycle avoidance technique on a number of different networks. The default routing behavior (primary routing entries) of the experimented networks follows shortest path routes as configured by a distributed link-state routing protocol such as OSPF. To obtain representative results, the computed routes were randomized and made independent between the nodes in the network. This implies that, if multiple shortest paths are available between two network nodes, every run will randomly choose a route, and configure the routing tables accordingly. This

[3]The similarity of paths from two nodes towards a third, can be measured by actively storing temporary forwarding states during the probing process, or using traceroute measurements as is performed in [55]

Table 7.1: Network topologies

Network	*Nodes*	*Links*	*Degree*		
			MIN	*AVG*	*MAX*
abilene	11	14	2	2.55	3
nobel-us	14	21	2	3.00	4
nobel-ge	17	26	2	3.06	6
garr	22	36	2	3.27	9
nobel-eu	28	41	2	2.93	5
geant2	30	47	2	3.13	8
renater	36	49	2	2.72	7
cost266	37	57	2	3.08	5
germany5	50	88	2	3.52	5
xwin	57	77	2	2.70	6

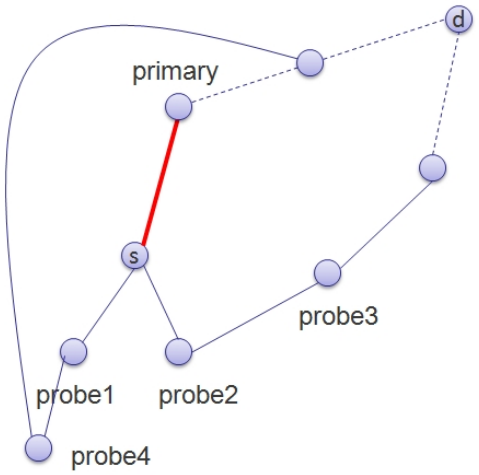


Figure 7.2: The probing process

performed independently on every network node. Every experiment has been re-run 100 times with these randomized settings, and the reported quantitative results are averages over these runs.

7.4.2 Network topologies

A set of 10 representative reference networks was used for evaluating the described techniques. Most of these networks are known for research purposes (e.g. [90]), or are research networks themselves. The number of nodes of these networks ranges between 11 and 57 nodes, and their node degree is in the range [3,9]. For some of these topologies, single connected nodes have been removed, because alternate routing paths are not possible for these anyhow. These topologies are representative of environments where the proposed technique would primarily apply. The properties of the reference networks are summarized in Table I.

7.4.3 Experimented techniques

The techniques with the following labels were benchmarked:

1. random. This technique refers to the configuration of usual shortest path routing entries as performed by a link-state protocol such as OSPF, augmented with random alternate entry routing entries (the only requirement is that the alternate next outgoing interface is different from the primary outgoing interface).

2. learn_backup_ipfrrlf. The scheme technique refers to the configuration of Loop-Free Alternates (also referred to as FRR-LFA) as backup entries as discussed in Section II.B [13], if they are available. Finding a loop-free alternate entry is performed by probing the paths from nodes' neighbors to check if they loop-back towards the originating nodes.

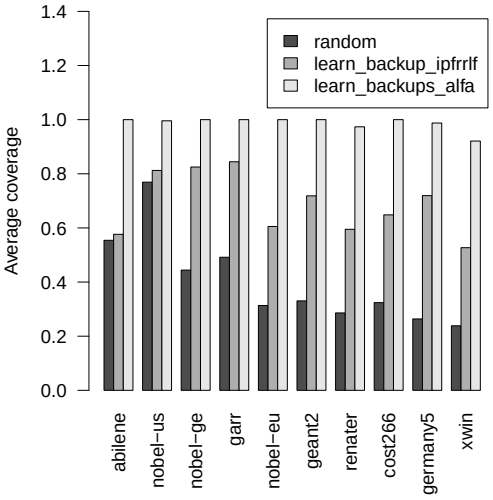
3. learn_backups_alfa. Here, alternate forwarding entries are configured using the proposed ALFA technique detailed in Section 7.3, which finds Loop-Free alternates using the BFS+ method. In this case, the LFN is chosen within the diameter of the closest node out of the loop-domain, having a path towards the targeted destination which differs maximally within the probed neighborhood.

7.4.4 Experimental results

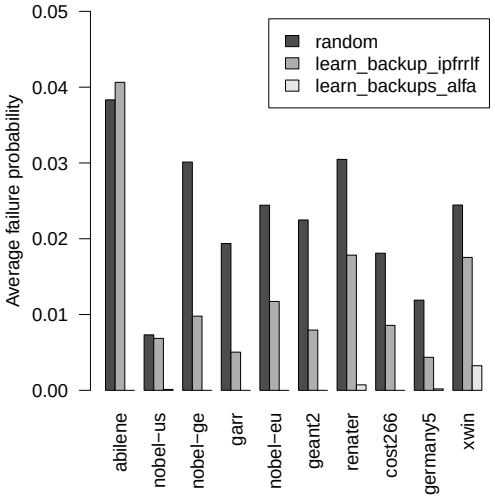
This section discusses the performance of the mentioned techniques with respect to their ability to cover link failures (coverage), their consequences on the resulting length of the backup paths (stretch), their communication cost for learning adequate entries, and their sensitivity with respect to network characteristics.

Coverage

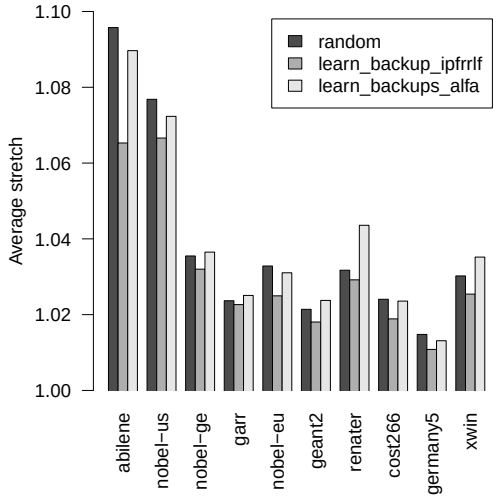
For each topology (see Table I), every possible single link failure was simulated. When a configured alternate forwarding entry (using one of the three presented techniques) is not able to recover the connectivity between all network nodes for a given link failure, the link is considered to be uncovered. The percentage of links which cannot be fully recovered upon link failure is denoted as the link failure coverage of the technique (the complement of this percentage denotes the percentage of links which can induce cycles among at least one source destination pair). Figure 7.3(a) depicts the link failure coverage of all considered re-routing schemes on all evaluated networks. From this figure, we can observe that provisioning randomly alternate entries is (as expected) only able to cover 20 to 50 percent of the link failures. FRR-LFA is able to cover larger percentages of link failures, typically between 50 and 80 percent. The ALFA technique which we propose is able to cover almost all link failures, and at least 95 percent. The probability that a link failure will cause a cycle when selecting the alternate routing path between a random pair of nodes can be calculated, by evaluating the connectivity between all pair of nodes, for all possible single link failures. Figure 7.3(b) shows the resulting end-to-end cycle probability induced by a single link failure for the experimented networks for all the considered schemes. One can observe from this figure that the probability that a cycle occurs between a given pair of nodes is high when using only random alternate entries, and lowest - close to zero - when using the ALFA technique. For small networks, the difference between provisioning random alternate forwarding entries and FRR-LFA is negligible.



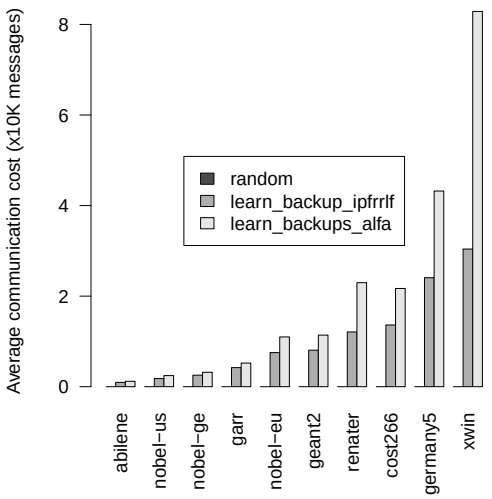
(a) Average link failure coverage vs. topology



(b) Average failure probability vs. topology



(c) Average stretch vs. topology



(d) Average communication cost vs. topology

Figure 7.3: Average performance vs. topology

Stretch

The previous metric measured the quality of the protection techniques in terms of the proportion of link failures they can potentially (fully) recover from. However, this metric doesn't give us information on the quality of the resulting alternate routing paths with respect to the path length. It may be expected, that higher recoverability could have a detrimental influence on the resulting path length. To assess this assumption, we calculate and depict the average stretch of the alternate routing paths of all techniques in all networks in Figure 4. The stretch metric indicates the ratio of the length of the alternate routing path (in hop count) vs. the length of the shortest routing path when no failure occurs. When the alternate routing path has the shortest length, the stretch is equal to 1. The average stretch calculates the average ratio over all resulting path lengths. Figure 7.3(d) illustrates that the length of the selected alternate paths taken out of all experimented techniques is at worst 10 percent longer than the (primary) shortest path between two nodes. In most cases, FRR-LFA uses the shortest backup paths. This may be a consequence of the fact that upon the failure only the first hop is different from the primary shortest path in the network, while the ALFA technique may use longer detour paths to ensure that the packet is out of the loop-domain of the failure detecting node. This explains the higher stretch values for the ALFA technique.

Communication cost

Finding adequate alternate routing paths ensuring that no cycles occur requires some probing and learning activity in the network. Techniques relying on probing lead to a cost with respect the number of probing messages; this cost is referred to as the communication cost. Both FRR-LFA and the ALFA technique involve probing: the first probes for a loop-free alternate neighbor, the second probes for a node out of the loop-domain of the failure detecting node. Figure 5 depicts the number of probing messages that were needed before the required techniques converged. The results illustrate that the ALFA technique requires a probing communication cost between 20 percent (for the smallest networks) and 270 percent (for the largest) higher networks.

7.5 Conclusion

We have proposed an alternative (learning) method for populating alternate routing entries. The resulting technique is able to avoid almost 100 percent of potential routing cycles upon the occurrence of single link failures in a given set of representative reference networks. We showed that this had low impact on the quality of the resulting backup paths, which were at most 10 percent longer than the shortest paths in the fully operating network. Future work could focus on reducing the induced communication cost of learning adequate alternate entries. Using the spatial correlation between several nodes, clustering techniques could correlate groups of destinations allowing common LFNs along their alternate path.

Chapter 8

Profile-based accountability

8.1 Introduction

Today's Internet has undergone a major shift since its original adoption: narrowband services such as web browsing and e-mails are not the most popular services any more. Instead, they have been replaced with multimedia services such as video streaming and on-line gaming, which are all very bandwidth intensive and require strict Quality of Service (QoS) guarantees to ensure a smooth service delivery. Even small drops in bandwidth can deteriorate the quality as perceived by the end user.

As the demand for bandwidth in the Internet grows, so does the length of each connection. While a best-effort web browsing service (e.g. served by a web server) will typically have to support many connections that last for a fraction of a second and transmit only a few kilobytes, a video streaming service will have connections that can last for minutes and transmit megabytes of data. These longer lasting connections increase the need for techniques that ensure an adequate fairness between connections. The longer connections last, the more they can take advantage of their achieved rate (e.g. by starving newly started connections) as they are further away from their start-up phase, which is often characterized by a slow start behavior. Enforcing fairness between connections is therefore becoming more and more important in today's Internet.

Today, the TCP protocol is by far the most widely used transport protocol [101], even for video services where more and more progressive download based mechanisms are used. Although the widely used TCP protocols all contain congestion avoidance algorithms that, in theory, should enforce fairness between connections, practice has shown that existing differences between TCP stacks result in important limitations. This is motivated by one of the IRTF's recent RFCs [91], discussing the open research issues in Internet congestion control, which identifies misbehaving senders and receivers as one of the 7 main challenges in congestion control. Furthermore, the authors of [107] show that the presence of a single misbehaving connection can lead to a collapse of the throughput of other connections. According to them, in the worst case, the achieved gain of the misbehaving connection is a factor 30 million. As TCP only resides on the terminals of a connection, a network relies on the correctness of the TCP's congestion avoidance algorithms. However, it is not in the TCP stack's best interest to feature such altruistic behavior. Therefore, TCP implementations can exploit this trust to achieve a higher throughput while violating TCP's congestion avoidance principles, at the cost of the throughput of other connections. As we will show in this section, such behavior can even be

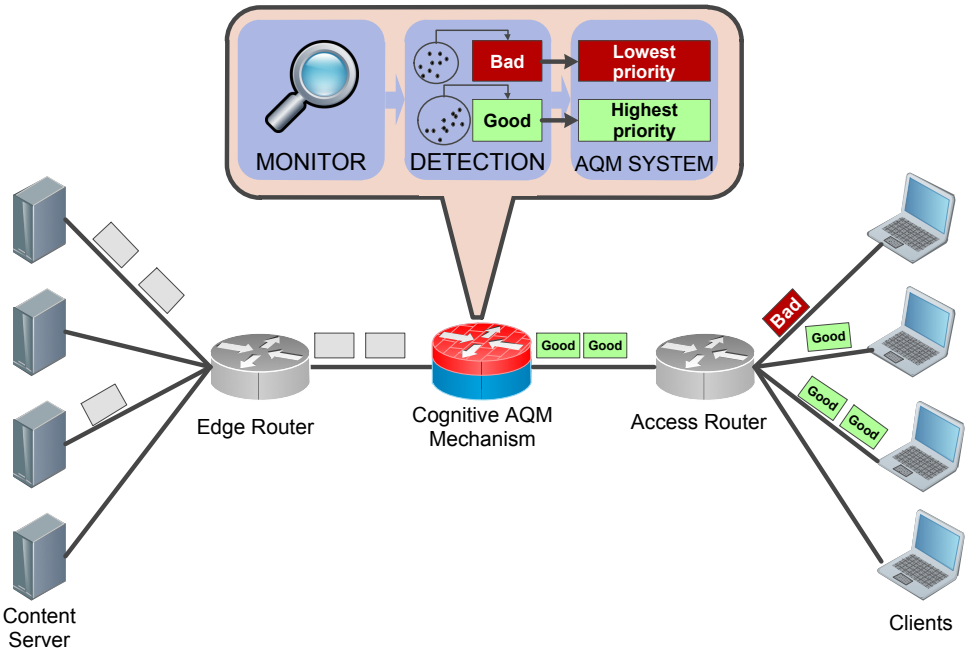


Figure 8.1: Overview of the role of a cognitive mechanism on top of an existing AQM system to introduce a higher level of fairness. Individual TCP connections are monitored to detect unresponsive TCP stacks, which can then be penalized to favor the responsive TCP stacks.

inflicted without modifying the implementation of a TCP/IP stack in the kernel. We call TCP stacks that feature these limitations unresponsive, because they do not or inadequately respond to congestion signals. Without in-network management, it is not possible to adequately manage unresponsive TCP stacks.

Instead of solely trusting on the congestion avoidance behavior of the terminals, more intelligence is often introduced into the network's routers through the Active Queue Management (AQM) paradigm. Instead of only dropping packets when a router's queue is full, an AQM system can drop packets earlier or can signal the existence of congestion to the terminals through the Early Congestion Notification (ECN) [20], which marks the packets by setting a flag in the packet header. The TCP terminals are then expected to respond to this congestion signal by lowering their rate. Additionally, by assigning the connections to different traffic classes, a router can also choose to prioritize one traffic class over the other.

In this deliverable, we focus on both misbehaving receivers and unresponsive senders. A *misbehaving receiver* is a TCP receiver who deliberately ignores TCP congestion warnings to willingly achieve an advantage in throughput at the cost of other, well-behaving TCP receivers. A misbehaving TCP receiver does this by tricking the TCP sender. On the other hand, an *unresponsive sender* is a TCP sender who reacts differently to TCP congestion warnings than is generally expected. This difference in reaction may or may not be deliberate and may cause a difference in TCP throughput between different responsiveness levels. Often, different responsiveness levels are caused by different TCP congestion avoidance mechanisms (e.g., a Multiplicative Increase Multiplicative Decrease congestion avoidance mechanism compared to an Additive Increase Multiplicative Decrease congestion avoidance mechanism). The problem of misbehaving receivers is discussed in Section 8.3, while the problem of unresponsive senders is discussed in Section 8.4.

8.2 Related work

Congestion control algorithms are probably the most widely available feedback loop algorithms available in the current Internet. As the Internet connects users from all over the world, competing for the same bandwidth, already from the Internet's early adoption it became clear that, in order to guarantee a reliable and fair data transfer, the rate at which the competing sources are sending needs to be controlled. Due to the distributed nature of the Internet and thus the lack of a centralized manager, these congestion control algorithms must apply adaptive heuristic algorithms that estimate the network load and react by changing their rate on the sign of congestion. Some key papers from Jacobson [58] and Chiu and Jain [28] highlighted the gain of an adaptive algorithm, called Additive Increase Multiplicative Decrease (AIMD), that linearly increases the rate in the absence of congestion and exponentially decreases the rate in the presence of congestion.

In today's Internet, congestion control algorithms are typically deployed both on the terminals (i.e. the end users) of the network as in the network itself. At the terminals, congestion control in the TCP protocol was standardized in [10] and the concept of a congestion window, denoting the maximum rate at which a sender may transmit, was introduced. Since then, congestion control algorithms have found their way in all TCP implementations used, such as TCP New Reno [42], which linearly increases the congestion window, and TCP CUBIC, which is the default protocol in Linux in recent kernel versions and uses a cubic function to increase the congestion window.

In the network itself, the introduction of AQM has led to a finer control on the traffic that passes through the routers. Instead of using DropTail queues, which simply drops packets once the queue is full, Random Early Detection (RED) [43] algorithms and variants such as BLUE [40] maintain an exponentially weighted queue length and base their dropping probability on the average queue length. These alternate queuing algorithms have the effect that the amount of packet drops, which is a signal of congestion for most TCP congestion control algorithms, exponentially increases as the load increases. By combining RED with the ECN mechanism, other congestion warnings than packet drops can be given earlier on to TCP terminals without experiencing data loss.

The growing diversity in congestion control on one hand and the increasing importance of achieving fairness on the other hand has led to modeling approaches where the interaction between TCP congestion control and AQM is analytically modeled [34, 120]. Kelly et al. [64] showed that a TCP congestion control algorithm on one hand and AQM scheme on the other hand can be modeled as two separate components, a primal and dual component, that both try to maximize a utility function. Depending on the implementation of the congestion control algorithm, other utility functions can be provided. In striving to maximize their separate utility functions, Kelly showed that they reach an equilibrium which corresponds to different fairness levels depending on the specifics of the utility functions.

While these different levels of fairness can be characterized off-line with the knowledge of the implementation details of the congestion control algorithms, for a router in an on-line scenario there is no way of knowing these utility functions and the corresponding achieved fairness. Therefore, this is a sub-optimal solution for a network provider: he typically wants network fairness that does not depend on the type of congestion control algorithms used. As he has no control over the terminals in the network, misbehaving TCP stacks can deteriorate the quality of other, more trustworthy, TCP stacks [104].

In this section, we focus on misbehaving stacks caused by an unresponsive ECN mechanism. This type of unresponsiveness has received much attention as the ECN mechanism can easily be exploited to ignore ECN messages without requiring any change in the kernel's implementation. This has led to an extension of ECN, called ECN-nonce, defined by the IETF in [110]. ECN-nonce enables a sender to check the integrity of a receiver. The authors of [68] extend the ECN-nonce technique to also support misbehaving receivers in multicast scenarios. However, ECN-nonce relies on the integrity of the server, which is sometimes not the case (e.g. in P2P networks where the operator does not have control over any of the end hosts). Moreover, performing the same integrity check by an AQM system in a router can be challenging. Our approach is specifically designed to be deployed on top of an existing AQM system. Both [57] and [51] address the issue of improving the fairness between connections, either through the design of a new AQM algorithm or the adaptation of RED, respectively. Here, both approaches are focused on penalizing misbehaving connections in the form of unresponsive UDP connections. In this section, we focus on unresponsive TCP connections.

We argue that a cognitive approach, consisting of an unsupervised machine learning algorithm, is most suited to detect misbehaving TCP stacks, because of its adaptive nature to changing circumstances and because the detection of a misbehaving stack is in essence a clustering problem that finds the cluster of good stacks versus the cluster of bad stacks. Machine learning algorithms have been successfully applied to network management domains such as improved routing [56], topology control [113] and anomaly detection [108, 18].

More specifically related to the problem of congestion control, the authors in [98] use a classification method to identify the type of application based on statistical features of the connections such as the length of the packets and the inter arrival times. In our approach, we use similar statistical information of a connection to decide on the responsiveness of a connection. In [37], a classification mechanism is proposed that is able to distinguish between losses related to congestion and losses related to link errors. As many congestion control algorithms interpret packet loss as a sign of congestion, the proposed classification method is able to filter out the data losses that have nothing to do with congestion, which increases the throughput of the TCP connections. The authors of [61] tackle the same problem but use Hidden Markov Models and Expectation Maximization algorithms as machine learning techniques. Our proposed technique is complementary to these techniques as it can only benefit from more responsive and hence accurate decisions on congestion signals.

8.3 Misbehaving receivers

In this section, we propose a cognitive mechanism, which we call a cognitive accountability mechanism, that is able to detect unresponsive TCP stacks and penalizes them to achieve a better fairness between the individual connections. The cognitive mechanism can be deployed on top of a traditional AQM management scheme and is transparent to the other nodes in the network. It provides a service to the operator by altering the forwarding of packets to improve the fairness. The main idea is that the algorithm holds subscribers accountable for their actions by penalizing unresponsive behavior. A general overview of the proposed cognitive mechanism is illustrated in Figure 8.1. The major contributions of this section are the following. First, we propose a machine learning based detection algorithm that monitors the traffic passing through a router and decides whether or not a TCP stack is unresponsive. The detection algorithm uses a

combination of outlier detection and clustering based on extracted flow statistics to find groups of responsive and unresponsive stacks. Second, we present a differentiated AQM mechanism that penalizes unresponsive stacks by disabling their ECN support. Third, we evaluated the cognitive mechanism by deploying it on a testbed facility of more than 400 nodes. The performance evaluation characterizes the accuracy and speed of the detection algorithm and identifies the obtained gain of the differentiated AQM mechanism in terms of improved fairness for different types of unresponsive stacks.

The flexibility provided by an AQM system can thus be exploited to introduce a greater level of fairness between connections. If it is possible to detect unresponsive TCP stacks, an autonomic differentiated AQM management can be introduced that distinguishes between responsive and unresponsive stacks to penalize unresponsive TCP stacks and reward responsive TCP stacks. As this detection consists of finding different behavior between connections, we argue that a cognitive approach is best suited. In this section, we provide an answer to the following research questions: (1) How can the occurrence of unresponsive TCP stacks be detected inside a router's AQM system? (2) Can the penalization of unresponsive connections lead to a better fairness between connections? and (3) How fast and accurate is the response of a differentiated AQM management mechanism?

The remainder of this section is structured as follows: Section 8.3.1 characterizes the destructive effect of unresponsive TCP stacks on the obtained fairness. An overview of the experimental set-up used throughout the rest of the section is provided in Section 8.3.3. Section 8.3.4 and Section 8.3.5 present the cognitive mechanism's detection algorithm and differentiated AQM mechanism, respectively. Both the detection and penalization algorithms are evaluated in Section 8.3.6 and the section is concluded in Section 8.3.7.

8.3.1 Problem statement

In this section, we discuss the impact of introducing unresponsive TCP connections into a set of TCP connections. We first describe how TCP stacks can be unresponsive and characterize their gain afterwards.

Unresponsive TCP stacks

We introduced various levels of unresponsive TCP stacks by altering the response of these TCP stacks to ECN signals. ECN messages are piggybacked with normal data segments and are intended as an explicit congestion warning to the receiver that congestion has been experienced, without requiring to drop packets. An ECN capable router such as those in an AQM system, will typically set ECN's Congestion Experienced flag in a segment to signal the existence of congestion instead of dropping the segment. To achieve this, an AQM system uses a queuing algorithm such as Random Early Detection (RED) [43] where segments are ECN marked when the queue size exceeds a threshold and are only dropped under very high load conditions, i.e. if another threshold is exceeded.

When an ECN message arrives at the TCP receiver, the receiver is expected to echo these congestion signals back to the sender. This is done through ECN Echo (ECE) messages. The TCP sender is expected to respond to this ECE message by lowering his congestion window and confirming the decrease of the congestion window through a Congestion Window Reduced (CWR) message.

Partially ignoring ECN messages A TCP receiver can easily exploit this mechanism by ignoring some or all ECN messages he receives. In this case, the congestion signals are not echoed back to the TCP sender, and he cannot adapt its congestion window leading to a higher throughput until the load becomes too high and the TCP sender responds to actual packet loss. After that, the rate is again gradually increased until packet loss is again experienced. When only a fraction of the TCP stacks ignore these messages, they can achieve a higher throughput because other TCP stacks will lower their send rate earlier. In this case, the unresponsive TCP stack is positioned at the receiver side instead of the sender side. Note that an implementation of such unresponsive TCP stack is straightforward and does not even require modifications of the TCP/IP stack in the kernel. Received packets can easily be captured and modified to remove the ECN flag, at which point they are sent to the actual TCP stack. In our experiments, we varied the percentage of ECN messages that are ignored to introduce different levels of unresponsiveness. In the remainder of this article, these type of unresponsive TCP stacks are abbreviated as IgnoreX, where X is the percentage of ECN messages that are ignored. Hence, Ignore30 will ignore 30% of all ECN messages.

Adaptively ignoring ECN messages Instead of ignoring random ECN messages, a second type of investigated unresponsive TCP stacks performs a more adaptive ignoring of ECN messages. This type of ECN flag manipulation takes place on the receiver side, but assumes that the receiver knows the configuration parameters and current queue size of the router sending the ECN messages at all time. Note that this type of information is hard to estimate for the receiver. However, this type of unresponsiveness denotes a more advanced ignoring as it continuously switches between a responsive and unresponsive mode, which is of interest from a theoretical viewpoint, as it is harder to detect this behavior. In our experiments, the AdaptiveIgnore component was deployed just after the queue to allow it to have access to the queue length.

In this case, the ECN Congestion Experienced flag is only ignored in situations where the queue is not in an extreme load mode. These situations are denoted by the length of the queue with respect to the RED thresholds. The idea is that the unresponsive stack pushes the queue to the limit, without actually causing losses. Therefore, ECN messages are ignored until the queue length is higher than the RED max threshold, at which a fraction of the packets of the packets are being dropped by the queue. When this RED max threshold is exceeded, the unresponsiveness is temporarily suspended which makes the TCP stack behave as a regular TCP stack again. Once the decrease in the queue length is sufficient, denoted by the RED min threshold, the unresponsiveness is again enabled and all ECN messages are ignored. In the remainder of this article, this type of unresponsive TCP stacks is abbreviated as AdaptiveIgnore.

Experienced fairness

We investigated the impact of both types of unresponsive TCP stacks on the experienced fairness. As the gain unresponsive connections achieve varies depending on the number of connections present, we illustrate a scenario where only 5 connections are active as well as a scenario with an average of 400 active connections. Both scenarios use a network model that will be presented in Section 8.3.3.

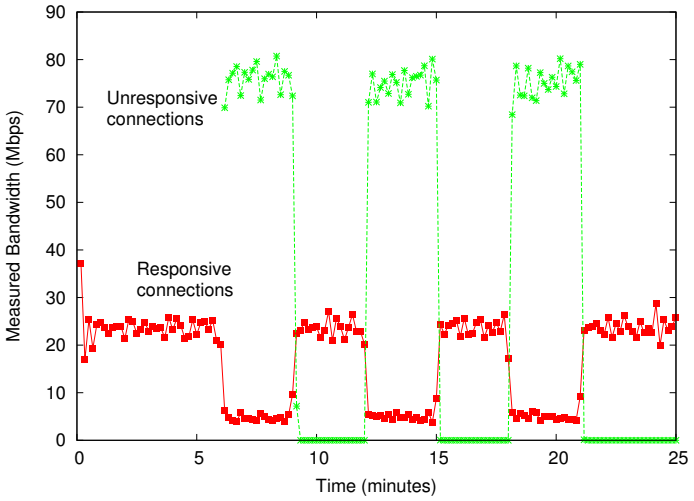


Figure 8.2: Average measured throughput for 4 responsive connections and 1 unresponsive connection. A new unresponsive connection is started every 6 minutes for 4 minutes leading to a starvation of responsive connections during that period.

8.3.2 Limited connection set

In this scenario, a maximum of 5 connections are present at each time: 4 responsive connections that are active during the complete duration of the experiment and 1 unresponsive connection that is initiated every 6 minutes and is stopped 4 minutes later. The 5 connections are transmitted over one bottleneck link of 100 Mbps. Figure 8.2 illustrates the average measured throughput of the responsive and unresponsive connections and clearly shows the disruptive nature of an unresponsive connection on a small set of responsive connections. As can be seen, each time an unresponsive connection is initiated it receives approximately 80% of all available bandwidth; the 4 responsive connections, that in fact should receive 80% to have perfect fairness, only receive a throughput of 5 Mbps each. This means that one unresponsive connection achieves a throughput, which is 16 times higher than that of a responsive connection. Each time the unresponsive connection is stopped, the throughput of the responsive connections is restored, but quickly drops again as a new unresponsive connection is introduced.

Larger connection set The previous section showed that responsive connections can starve if only a few connections are present in the network leading to throughputs that are 16 times higher than their responsive counterparts. As traffic inside a router typically contains much more active connections, in this experiment, we look at a scenario with a larger active connection set. For this experiment, we introduced TCP connections that either ignore all ECN messages (Ignore100) or adaptively ignore ECN messages (AdaptiveIgnore). We varied the number of unresponsive stacks that were introduced. Figure 8.3 illustrates the gain each type of unresponsive connection can achieve as a function of the time and depending on the amount of unresponsive connections. This gain is defined as the fraction of the average measured throughput of unresponsive connections to those of responsive connections, taking into account both

bottlenecks. Thus, the gain value is calculated as follows:

$$\text{gain} \equiv 50 \times \sum_{j=1}^{2} \frac{m \times \sum_{i=1}^{n} \text{unresponsive}_{i,j}}{n \times \sum_{i=1}^{m} \text{responsive}_{i,j}} - 100$$

where $\text{unresponsive}_{i,j}$ and $\text{responsive}_{i,j}$ are the throughput of the ith connection of bottleneck link j in the set of unresponsive and responsive connections, respectively. Hence, the gain value defines how much percent, on average, the unresponsive connections outperform the responsive connections. For this experiment, we also illustrate the achieved gain during the initial start-up phase, which lasts from 0 to 600 seconds, to show what happens if responsive and unresponsive connections race for the same bandwidth. The gain illustrated from 600 seconds onwards, shows the situation in a steady state.

As shown, all types of unresponsive connections achieve a remarkable gain when compared to responsive connections during the start-up phase. Although the gain is lower than the scenario with a limited connection set, where the gain was 1500%, it is still significant. During this phase, all unresponsive connections are able to obtain a throughput which is more than 100% higher than that of responsive connections. In the case where all ECN messages are ignored, the throughput is even 200% more than its counterpart. As more connections are being started, and all connections try to find an appropriate congestion window the gain decreases and differences between the type of unresponsive connections start occurring. When only 1 single connection is unresponsive, the gain is only notable during the start-up phase and disappears as the connections converge to a steady state. This is because the single unresponsive connection only has an influence on other, responsive, connections if there are only a few responsive connections as well. If the number of responsive connections is high then they will only have a minimal impact on their throughput. As such, ignoring ECN messages in a steady state only results in additional packet loss for the unresponsive connection, and automatically forces the unresponsive connection to lower its rate as well. As such, single unresponsive connections can only achieve a gain during a start up phase, although this phase can last for several minutes.

However, if the share of unresponsive connections increases, a higher gain can be achieved during the steady state as well. As shown in Figure 8.3, the AdaptiveIgnore type is able to profit more from ignoring ECN messages if 30% of all connections are from that type as opposed to only a single connection. Here, the achieved gain decreases as well over time but stabilizes around the 900 second mark. At this point, the AdaptiveIgnore type still has a limited gain of approximately 2.5%, which is almost negligible. This limited gain is due to the fact that the network continuously suffers from high loads, which makes that the differences in responsiveness between the regular connections and the AdaptiveIgnore connections disappear. The adaptive nature of the unresponsive connection results in a TCP stack that does not ignore ECN messages when packets are dropped from the queue. As the queue is continuously on the verge of dropping packets, the AdaptiveIgnore connections behave just as regular connections, explaining the only marginal gain.

The largest gain is achieved when all ECN messages are ignored, described as Ignore100. A decreasing trend is also noticeable here as the start-up phase evolves to a steady state. However, in this case, the gain of being unresponsive stabilizes at approximately 11.5%. This means that the throughput of every unresponsive connection that ignores all ECN message continuous to

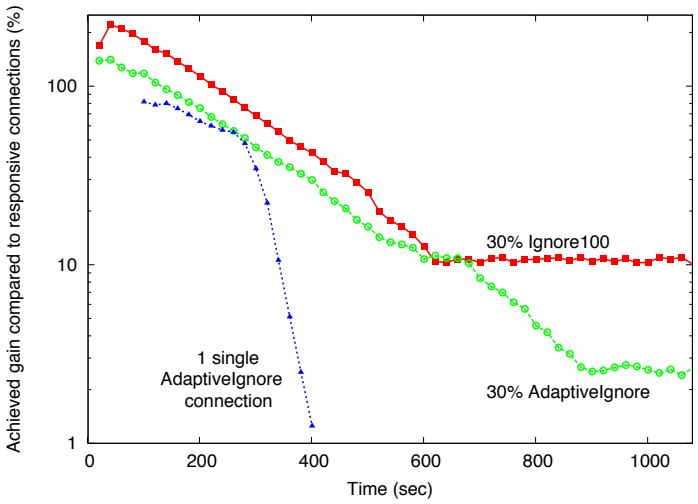


Figure 8.3: Achieved gain of being unresponsive when compared to responsive connections. Three situations are considered: 30% connections ignoring all ECN messages (Ignore100), 30% of the connections adaptively ignoring ECN messages (Adaptive Ignore) and 1 single AdaptiveIgnore connection.

be 11.5% higher than any responsive connection. This is because this type of unresponsive connections is by far the most aggressive: they continue to increase their rate as they ignore any congestion signals. Only when they actually suffer from losses, they will lower their rate: at that time, other connections have also lowered their rate, but much more as they responded to ECN signals as well. From then on, the same process repeats itself: the rate is gradually increased until losses occur again.

In summary, these results illustrate three important points: First, all unresponsive connections do achieve a significant gain at the cost of responsive connections, especially in the start-up phase. Second, even in the steady state, the gain is still notable if the share of unresponsive connections is high enough: only a single unresponsive connection will not harm the fairness between connections. Third, and finally, the connections with the highest aggressiveness are able to obtain largest gain up to 11.5% more than responsive connections.

8.3.3 Scenario generation environment

In this section, we discuss the approach that was taken to construct the experiments, and will be used throughout the remainder of this article. As we propose a machine learning approach as part of the detection algorithm in the cognitive mechanism, this experimental approach was used both for the creation of datasets and to evaluate the overall cognitive mechanism.

Experimental set-up

Experimental environment All experiments were carried out on the iLab.t Virtual Wall, which is a large test-bed facility that uses the Emulab [116] management software to dynamically construct network topologies. The Virtual Wall consists of 100 nodes that are connected to a non-blocking 1.5 Tb/s VLAN Ethernet switch and a display wall for experiment visualization. The nodes themselves are dual processors, dual core servers that each contain 4 or 6 gigabit

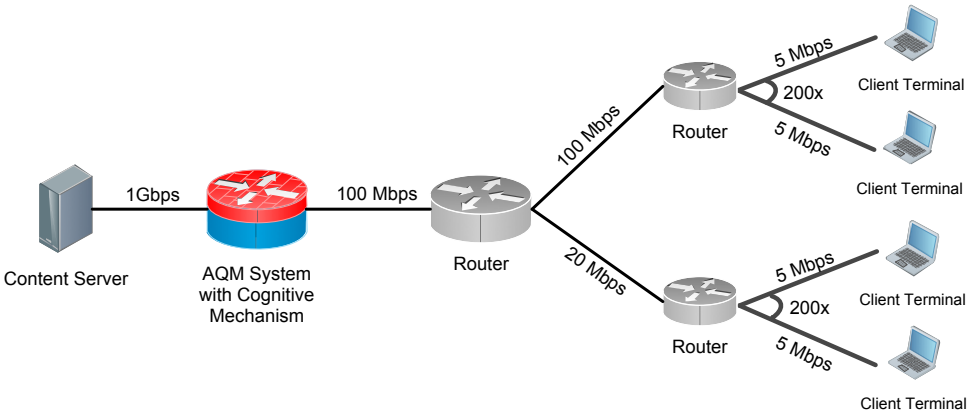


Figure 8.4: Used network topology for all experiments: the type of client terminals are varied in the different experiments. Clients are assumed to request the download of a content item (e.g., video file) which is then sent from server to, possibly unresponsive, clients.

links, that are connected to the non-blocking switch. The Virtual Wall allows defining the network topology through scripting; this includes the option of installing an operating system's image on the nodes and defining additional software to be installed on startup. The Virtual Wall includes several images that can be installed to emulate typical network functionalities such as server, client terminals, routers and impairment nodes. Once configured, root access is provided to all nodes so that an operator has complete control over the execution of the experiment. As such, the Virtual Wall can be used as a generic test environment for emulating large network topologies, evaluating newly developed algorithms and performing scalability tests. The Virtual Wall supports virtualization so that multiple virtual machines can be mapped to one physical machine. Thus, the Virtual Wall supports emulated network topologies of more than 400 nodes.

Investigated topology Figure 8.4 illustrates the network topology that was used in the Virtual Wall testbed facility. This network topology emulates part of a tree-based access network topology of moderate size. As shown, a file server connects 400 client terminals. The network consists of two bottleneck links: one bottleneck of 100 Mbps affecting all client terminals and one bottleneck of 20 Mbps affecting only 200 of the 400 client terminals. Just in front of the 100 Mbps bottleneck link the cognitive accountability algorithm is introduced. As will be detailed in the next Section, different types of unresponsive stacks are introduced at the client terminals.These client terminals initiate a TCP connection between client and server and start downloading files from the server. Note that the server is assumed to be responsive; however, the occurrence of unresponsive client terminals on the other side of the network may trick the server in sending at a too high rate and thus unwillingly favour unresponsive client terminals. The cognitive mechanism monitors both the downstream and upstream traffic to detect these unresponsive stacks. Note that, although we only focus on downstream (from server to client) originating traffic, the approach can easily be extended to also support upstream originating traffic as both are independent of the other. In that case, the cognitive AQM system would also be deployed on the regular router in Figure 8.4.

Scenario definition To model the request rate and characteristics of each TCP connection, the following probabilistic models were used. First, a Weibull process was used to model the inter arrival times between different requests from client to server. The Weibull process was configured with $\lambda \equiv 2.15$ and $k \equiv 0.9$, which resulted in an average inter-arrival time of 1.5 seconds between TCP requests. This results in an average of 400 connections in 10 minutes. We uniformly distributed the requests amongst the different physical nodes. Note that these values were chosen based on observations in [95]. Second, for each connection, the size of the downloaded content was modelled using a Pareto distribution where the average download time of a connection was set to 10 minutes. The effect of these request models is that, after a start-up phase, during the experiment, new connections are started and old connections are stopped but the expected number of active connections remains fixed at 400. As probabilistic distributions are used to model the inter-arrival time and download length, this is the expected number of active connections and the exact value may slightly differ depending on random values of the probabilistic models.

Scenario and dataset creation tool

To generate the scenarios as discussed above, a framework was used that manages the execution of the experiment on the virtual wall. The tool, which has been proposed by the authors in the past [71], is intended to support the generic execution of experiments but was specifically designed for this scenario. The tool allows configuring specific aspects of a tree based network experiment both through a Graphical User Interface (GUI) and through XML. This configuration consists, amongst others, of the definition and configuration of the Weibull and Pareto distributions to model the request rate of the connections and a definition of the type of TCP stacks that are assigned to the various client terminals. Moreover, the framework allows emulating variations of network algorithms (e.g. because of parameter configuration changes) under the same conditions, i.e. by using the same random seed, as well as changing the random seed to introduce variability in the circumstances and ensure repeatability of the experiments.

The configuration and execution of experiments is then straightforward using the management framework. For example, configuring a subset of the TCP client terminals on one hand and the AQM system in the network topology on the other hand can be performed through the following XML statements:

```
<user percent="33" ecn_ignore="off"
        ecn="on" stack="reno"></user>
...
<nodes function="ecn-stripper" level="1">
        <config up_delay="50ms" down_delay="25ms" stability="9"
            max_p="0.02" queuelen="300" max_th="90" min_th="30"
            red="default" Mbps="100">
        </config>
</nodes>
```

This configuration defines that 33% of all client TCP stacks use the TCP Reno protocol and support ECN messages. The config option allows setting the parameters of the Random Early Detection (RED) queue in the AQM system.

The management framework was applied to two phases in the experimentation: in a first phase, the logging functionality of the management framework was used to generate appropri-

ate datasets that allowed an off-line evaluation of machine learning algorithms for the cognitive mechanism. In this phase, the management framework was thus used as a means to create a data set for the machine learning components. The results of this phase were then used in the second phase, where the cognitive mechanism with the best performing detection algorithm was deployed into the actual network topology, allowing an on-line performance evaluation. The automated behavior of the management framework enabled to easily repeat the same experiments as those used for the dataset generation, as well as introduce new ones to create realistic scenarios.

8.3.4 Detecting unresponsive connections

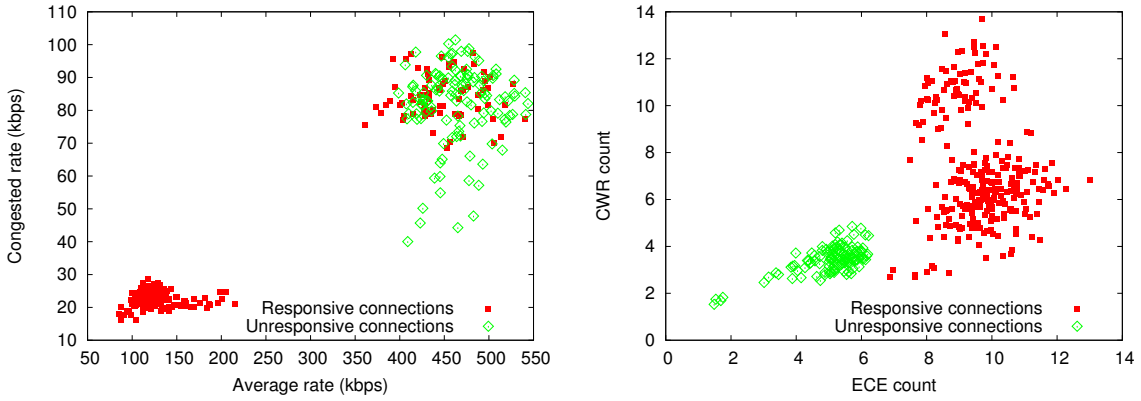
From the previous section it is clear that it is important to adequately respond to unresponsive TCP connections as they can significantly decrease the fairness between connections. To do this, it is first required to detect the connections that are being unresponsive. Once detected, specific countermeasures can be taken to decrease the throughput of these connections and improve the fairness. In this section, we propose a cognitive algorithm to perform such a detection. The cognitive algorithm classifies incoming connections as being responsive or unresponsive based on flow based statistics, collected from the traffic that passes through.

Attribute selection

In this experiment, we investigated which flow-based statistics are best suited to distinguish between responsive and unresponsive connections. We investigated 4 different statistics: the connection's rate, the connection's congested rate, the connection's CWR count and the connection's ECE count. The connection's rate denotes the average downstream (from server to client) throughput of the connection during a recent time window. The connection's congested rate denotes the average downstream throughput that was marked with an ECN flag or dropped in the queue. The connection's CWR count is also measured downstream and is the number of packets containing the CWR flag in the last time window. The connection's ECE count is measured upstream, from client to server, and is the number of packets with the ECE flag turned on in the last time window.

Figure 8.5 shows the dependency between two sets of attribute candidates: the average connection's rate and congested rate on one hand (Figure 8.5(a)) and the connection's CWR and ECE count on the other hand (Figure 8.5(b)). The experiment consisted of 30% unresponsive connections of the Ignore70 type, thus ignoring 70% of all ECN signals. We introduced only unresponsive connections in the client terminals that only have the 100 Mbps link as bottleneck. We averaged these attributes over time to avoid oscillations and because the absolute values did not yield any clear division between cluster groups.

As illustrated, both attribute candidate sets cluster the data into two groups but only the connection's CWR and ECE count are able to distinguish between responsive and unresponsive connections. The connection's average rate and congested rate clearly divide the connection's into two groups, one for each bottleneck. In this case, the lower left group corresponds with 50% of the connections (i.e. 200 out of 400) that each have an average throughput of approximately 0.1 Mbps, while the upper right group corresponds with the other half of the connections that take up the remaining 80 Mbps of the 100 Mbps bottleneck link. In this latter group, responsive and unresponsive connections are mixed. Although the unresponsive connections are more



(a) Average rate and congested rate

(b) Average CWR and ECE count

Figure 8.5: Comparison between several attribute candidates to base the clustering on. Both attribute sets cluster the data into two distinct groups, but only the average CWR and ECE count make the distinction between responsive and unresponsive connections.

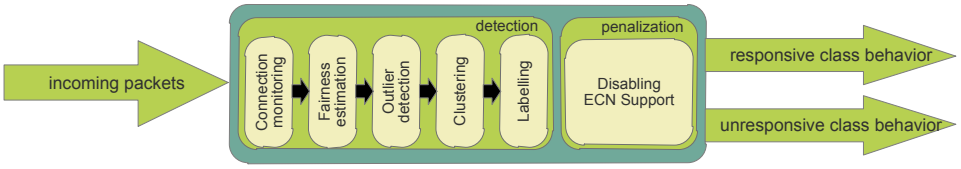


Figure 8.6: Overview of the detection algorithm and penalization component.

located at the right hand side of this group, which can be expected as - on average - they have a higher throughput, the clustering between responsive and unresponsive connections in this group is not feasible as the instances are too much interleaved.

The connection's CWR and ECE count on the other hand do have a clear division into two groups, one being responsive, the other being unresponsive. In this case, there is no visual difference between the two bottlenecks, which is an important advantage: as the clustering is independent of the existence of multiple bottlenecks, an increase in the number of bottlenecks thus not means an increase in the number of clusters. The downside of using the CWR and ECE count as attributes is that, unlike the division found in Figure 8.5(a), the distance between the responsive and unresponsive groups is rather small. Therefore, the clustering can be prone to noise on the data. As the connection's CWR and ECE count are the only two attributes that provide such a clear distinction between responsive and unresponsive ECN behavior, these attributes are used in the cognitive algorithm to base the labeling of responsive and unresponsive connections on. This is discussed in the remainder of this Section.

Detection of unresponsive connections

An overview of this detection algorithm and penalization component that uses the detection algorithm is illustrated in Figure 8.6. The goal of the cognitive detection algorithm is to classify incoming connections as being responsive or unresponsive. To do this, each connection is monitored and statistics about each connection are calculated and continuously updated. Based on a combination of clustering and outlier detection, the connections are divided into several

cluster groups, called profiles. Afterwards, these profiles are labeled as being responsive or unresponsive. The algorithm itself is an iterative process of 5 steps:

Step 1: Connection monitoring In the first step, each connection is monitored individually and 3 statistics are calculated: the connection's rate, the connection's CWR count and the connection's ECE count. To avoid oscillations of the measurements, each instantaneously measured statistic, which is in fact already averaged over 1 second, is smoothed through an exponential weighted moving average (EWMA). An EWMA is a straightforward way of calculating an average with the lowest memory constraints: only the average itself needs to be stored for future calculations. Each updated value is calculated through a linear combination of the previous one. For example, the connection's average rate at time n is calculated as

$$ConnectionRate(n) = w \times ConnectionRate(n) + (1 - w) \times Rate$$

where Rate is the instantaneous measurement of the connection's rate. In our experiments, we set the w value to 0.99, which corresponds with a moderate time window (i.e. of less than 50 seconds). The exact time window is hard to determine as the contribution of previous measurements to the current average decreases exponentially. These monitored statistics are used in the detection algorithm as follows:

We calculate an updated value for the connection's average rate once every second. The calculation of the connection's CWR count and ECE count are only executed when this is triggered by the next step, the fairness estimation. The connection's average rate is used in the next step to detect the existence of a problem in fairness between flows. If such a problem is detected, the connection's CWR count and ECE count are also calculated and used in the remaining steps.

Step 2: Fairness estimation While the connection's average rate can easily be calculated based on the size of each packet running through the monitoring point, the calculation of the connection's CWR and ECE count requires more resources. This is because the CWR and ECE signals are encoded as TCP flags in the TCP header. This thus requires finding the location of the TCP header in a packet and decoding the TCP flags. To avoid this costly calculation as much as possible, it is only triggered if there is a potential problem with the fairness between connections. Such a potential problem is estimated through the Jain's fairness index [60], which is a metric that rates the fairness between a set of values and is often used for evaluating the fairness of TCP congestion control. The Jain's fairness index for n connections is calculated as follows:

$$\text{Jain} \equiv \frac{(\sum_{i=1}^{n} r_i)^2}{n \times \sum_{i=1}^{n} r_i^2}$$

where r_i is the average rate of connection i. The Jain's fairness index produces a value between $1/n$ and 1, where the former indicates the worst case scenario, and 1 indicates the best case. In our algorithm, the execution of the next steps - and corresponding calculation of the connection's CWR and ECE count - is only triggered when the Jain's fairness index is lower

than a threshold J. This threshold defines the variance that is allowed on the fairness between connections, before a detection and corresponding penalization or reward is initiated.

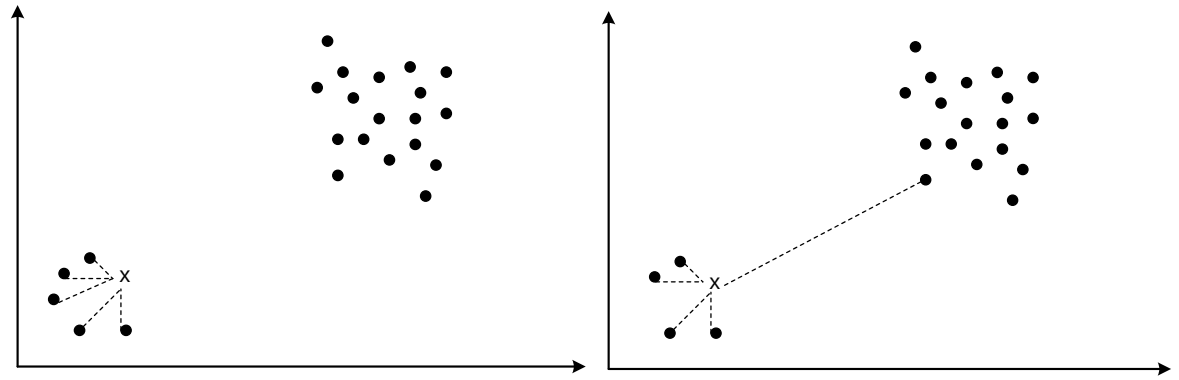
If the threshold J is exceeded, it indicates a difference in fairness between the connections. This difference can be caused either by the presence of unresponsive connections or because another bottleneck exists further in the network that results in lower throughputs for a subset of the connections. In the latter case, the outlier detection and clustering steps should not find any unresponsive profiles.

Step 3: Outlier detection The goal of this and the following step is to divide the different flows passing through the router into different groups, called profiles. These profiles represent different ECN behaviors that can be observed amongst the connections and are identified through an outlier detection mechanism and clustering mechanism. Both mechanisms perform their calculations based on two attributes: the connection's CWR and ECE count. As discussed in Section 8.3.4 these attributes are able to distinguish between profiles inflicted by unresponsive connections but do not detect differences because of multiple bottlenecks. Therefore, they avoid that a group that experiences an additional bottleneck is labeled as unresponsive.

As discussed in the previous section, a single unresponsive connection is also able to achieve a moderate gain in terms of bandwidth compared to the other responsive connections. A single unresponsive connection thus forms a profile on itself. In most clustering algorithms, a group with only one instance is considered to be an outlier and ignored in the clustering process. Even in clustering algorithms that allow to set the minimum number of instances belonging to a cluster, it is typically not advisable to set this value 1 as this makes the clustering algorithm more prone to noise. However, in this case, it is important that such a group with few instances is still taken into account, as it represents an unresponsive connection.

To ensure that a single unresponsive connection is also detected, we perform an outlier detection beforehand. This outlier detection is performed through the Local Outlier Factor (LOF) algorithm [19], which is a density-based algorithm that is also used in data mining to detect outliers. For each instance, the LOF algorithm calculates its density with its k nearest neighbors. Instances with a high LOF value are considered to be outliers: in our algorithm, each outlier is removed from the dataset for clustering and mapped to a separate profile, that is labeled in the next step.

The LOF algorithm was chosen because it has an interesting property with respect to its k parameter. Because the algorithm calculates the density with its k nearest neighbors, a cluster with $k+1$ instances will not be identified as $k+1$ outliers but as a cluster. This is illustrated in Figure 8.7, which shows the LOF calculation for an instance, marked as X, part as a group of 6 instances (Figure 8.7(a)) and a group of 5 instances (Figure 8.7(b)), respectively. In both calculations, k was set to 5. As shown, the group with 6 instances is not considered as having outliers because its 5 nearest neighbors are close to each other. If one instance is removed from that group, the LOF calculation also takes into account the next nearest neighbor which is part of a complete different group. Therefore, the LOF value is much higher and these instances are identified as outliers. In our algorithm, this characteristic is used to configure the k value in the LOF algorithm. Since an outlier can represent an unresponsive connection, it is important not to miss any outliers. Therefore, k is set to $C-1$, where C is the minimum number of instances needed in the clustering algorithm to form a cluster. The chosen clustering algorithm, allows setting this C value.



(a) Group of 6 instances (b) Group of 5 instances

Figure 8.7: Graphical overview of an illustrative outlier calculation for a group of 6 instances (a) and 5 instances (b). As k is set to 5, only the group of 5 instances is identified as outliers.

Step 4: Clustering Once the outliers are identified and removed from the dataset, the remaining instances are clustered using a regular clustering mechanism, according to the connection's CWR and ECE count. In the algorithm, each cluster is considered as a separate profile. Since it is not possible to know beforehand how many clusters remain in the dataset, after the outliers are removed, we use the DBScan clustering algorithm [39] as this is one of the few algorithms that don't require to define the number of clusters beforehand.

Step 5: Labeling of unresponsive profiles The output of the previous step is a set of profiles, where it is known that each profile represents a different behavior with respect to the ECN attributes. To goal of this last step is to label each profile as being either responsive or unresponsive. The labeling algorithm is straightforward: if only one profile is detected through the outlier detection and clustering algorithm, the Jain's fairness index is assumed to be exceeded because of multiple bottlenecks in the network, and the single profile is assumed to be responsive. If more than one profile is detected, one profile is labeled as responsive, while the others are labeled as unresponsive. Because the connection's ECN statistics (CWR and ECE count) are the only attributes the clustering is based on, different profiles can only occur because of differences in responsiveness and not because of multiple bottlenecks. Additionally, as shown in Section 8.3.4, regardless of the existence of multiple bottlenecks, responsive connections are always grouped into one cluster.

To identify the unresponsive profiles from the set of profiles, the labeling algorithm investigates the value of the connection's ECE count. From the attribute selection results in Section 8.3.4, it is clear that unresponsive connections have a considerable lower ECE count than responsive connections. While a similar trend can be observed in the connection's CWR count, the distinction between the connection's ECE count is more clear. Therefore, the algorithm labels the profile with the highest ECE count as being responsive and all the other profiles as unresponsive.

Rationale behind a cognitive approach

In essence, the cognitive algorithm classifies connections as responsive or unresponsive based on the value of the connection statistics. This may seem a behavior that can easily be imple-

mented by performing a simple threshold analysis on the connection statistics, which classifies a connection as unresponsive if a threshold is exceeded. However, we argue that there are number of reasons why a cognitive approach outperforms such a simple algorithm.

- *Robustness against network changes:* First, a cognitive approach is better suited against changes in the network itself. The derived connection statistics, connection rate, CWR count and ECE count, can easily differ depending on the context of the network scenario. For example, the connection's CWR count will typically be much higher when the link capacity increases. If in this case, unresponsive connections are present the detection algorithm still needs to distinguish between the two profiles. As such, the relative position of the different profiles is of importance and not the values themselves. The clustering algorithm takes care of this as clustering only focuses on finding groups of data and disregards the scale of the values. In an algorithm that performs threshold comparison this issue could be solved by normalizing the data. However, in this case, the existence of outliers can result in loss of information in the normalization process.

- *Robustness against connection changes:* Changes can also occur in the connections themselves. As discussed in Section 4, we varied the percentage of ECN messages that are ignored to introduce different levels of responsiveness. Ignoring only a fraction of the ECN messages can be used by unresponsive connections to mask their defectiveness. In a threshold comparison algorithm, such connections could be classified as responsive as they do not exceed the crisp border, defined by the threshold. The clustering algorithm allows for a more smooth transition as no crisp threshold is used for dividing the connections into clusters.

- *No training set required:* A common downside of a cognitive approach is that it requires to be trained to the actual scenario before deployment. Although the algorithm solves a classification problem, the algorithm uses only unsupervised machine learning techniques. This means that it does not need a training set to learn its expected behavior before deployment. Instead, a priori knowledge about the desired relative position of each profile is used to decide on the responsiveness of a profile.

8.3.5 Penalization of unresponsive connections

Once the connections have been identified, the responsive and unresponsive connections are assigned to different classes, a responsive and an unresponsive class, where each class receives a separate AQM behavior. We propose a simple but effective penalization action to support such a differentiated AQM behavior. The goal is to modify the AQM policy of those connections that violate the overall network fairness. Connections belonging to the unresponsive class are therefore not modified by the AQM system, as they already feature the desired behavior.

On the other hand, connections belonging to the unresponsive class are considered to be less reliable and therefore penalized. Since we focus on unresponsiveness with respect to ECN signals, the penalization consists of modifying the ECN support of the unresponsive class. Since connections belonging to the unresponsive class are assumed to not respond well to ECN signals, no ECN signals are sent to these connections. Instead, packets that would normally be marked with an ECN flag are immediately dropped from the queue. This has the effect that the unresponsive connections no longer need to react to ECN signals, and the responsibility

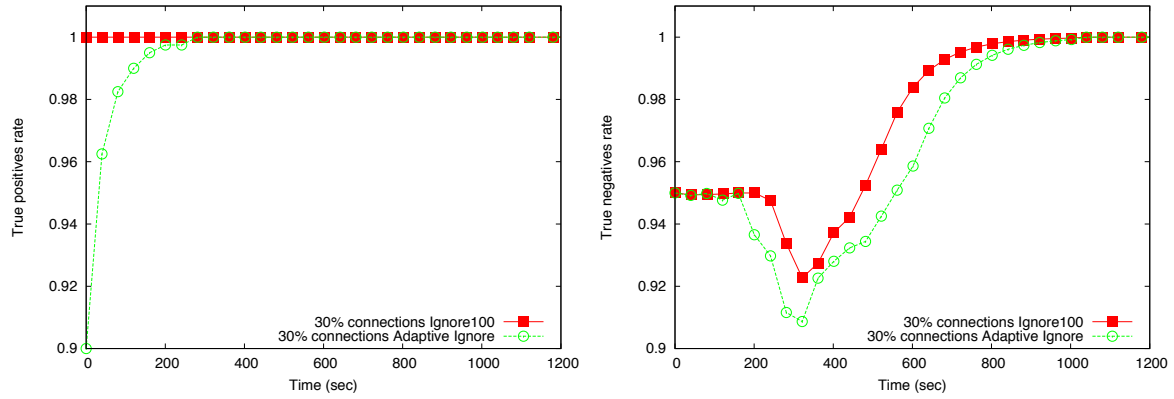
of lowering the sending rate is immediately positioned at the sender. This would eventually also happen as the unresponsiveness of a client leads to a continuous increase of the queue, and, in the end, to data loss as well. However, in this case, only connections belonging to the unresponsive class are affected by an increase in packet loss, while connections belonging to the responsive class continue to receive ECN signals. Disabling the ECN support also has an impact on the connection's CWR and ECE count attributes. As no ECN signals are received, these ECN statistics will also be zero. Connections that are mapped to the unresponsive class do not provide any useful information anymore for the detection algorithm, and are therefore removed from the set of connections on which the responsiveness is determined by this step.

To avoid the penalization of responsive connections, the decision on which connections are mapped to the different classes is not solely based on the detection algorithm, detailed in the previous section. Instead, a hysteresis is introduced to avoid the oscillation of the output of the detection algorithm. The introduced hysteresis consists of an exponential back-off algorithm: if a connection is mapped to the unresponsive class a timer starts that lasts for T seconds. After that, the connection is moved to the responsive class again. Once it is moved to the responsive class, the value for the timer T is updated. Each time the detection algorithm marks the connection as responsive, the T value is decreased with 1 second; if the detection algorithm marks the connection as responsive again, the T value is doubled and the connection is mapped to the unresponsive class again, and thus penalized, but this time for a longer period. New connections are thus given the benefit of a doubt, while connections that continue to give unresponsive detection results are penalized for increasing longer periods.

Without this exponential back-off algorithm, connections that are mistakenly mapped to the unresponsive class would continue to be penalized, since connections of the unresponsive class are no longer considered in the detection algorithm. The exponential back-off algorithm allows to periodically probe if connections were not mistakenly mapped. The frequency at which this probing occurs decreases as the number of times the connection is labeled as unresponsive from the detection algorithm increases.

8.3.6 Performance evaluation results

We evaluated the performance of the cognitive accountability mechanism by characterizing the accuracy of the detection algorithm on one hand and the overal gain of the penalization actions on the other hand. As network scenario, the network topology as presented in Section 8.3.3 was used, consisting of two bottlenecks of which only 50% of the client terminals are affected by the 20 Mbps bottleneck. We investigated the effect of the presence of both the AdaptiveIgnore and IgnoreX type of unresponsive connections and varied both the share and the level of unresponsiveness. In all experiments, the k parameter of the outlier detection algorithm was set to 5, as the DBScan clustering algorithm was configured to interpret a group of 6 instances as a cluster. The J threshold in the fairness estimation step was set to 0.95. Due to the existence of two bottlenecks this threshold was continuously exceeded: the efficiency of the detection algorithm to cope with these multiple bottlenecks is thus also evaluated. Each test was repeated 20 times: the variations between iterations were due to variations in the probabilistic models used to describe the inter-arrival time of requests and connection length.



(a) True positives

(b) True negatives

Figure 8.8: True positives and true negatives as a function over the time for different unresponsive connection types. Overall, the accuracy is always more than 90% and converges to an accuracy of 100% over time. The number of false positives is considerably lower than the number of false negatives.

Detection evaluation results

As the use of penalization has an impact on the dataset used for detecting the responsiveness of connections, we disabled the penalization algorithm in this set of experiments. This allows focusing on the accuracy of the detection algorithm avoiding any noise from errors in the detection process. We characterized the detection accuracy as a function over time as well as the impact of the share of unresponsive connections and the level of unresponsiveness in the IgnoreX case.

Detection response times The accuracy of the detection algorithm over time is illustrated in Figure 8.8, which shows the true positives rate (Figure 8.8(a)) and true negatives rate (Figure 8.8(b)) as a function over time. In two separate experiments, we introduced 30% unresponsive connections: one of type Ignore100, i.e. ignoring all ECN messages, and one of type Adaptive Ignore. As these graphs are similar to ROC curves, a true positives rate (respectively true negatives rate) of 1 means that there aren't any false positives (respectively false negatives) generated by the detection algorithm. Unlike a ROC curve, we do not plot the true positives or true negatives rate as a function of the sensitivity of the classifier but as a function of the time.

As discussed in Section 8.3.3, the first 600 seconds of this experiment define a start-up phase where the bottleneck link of 100 Mbps is gradually being filled up to 400 connections. From 600 seconds onwards, new connections start, while older connections stop, keeping the expected number of connections fixed to 400. The results in Figure 8.8(a) show that the detection algorithm generates almost no false positives: only in the first 400 seconds and in the case of AdaptiveIgnore a number of connections are marked as unresponsive, while they are in fact responsive. Here, the true positives rate increases from 90% to 100%. The other unresponsiveness type, Ignore100, does not suffer from any false positives over time.

The number of false negatives, as illustrated in Figure 8.8(b), is somewhat higher than the number of false positives. Moreover, although the true negatives rate also converges to 100%, the process goes slower, which means that there are some connections that are not detected as

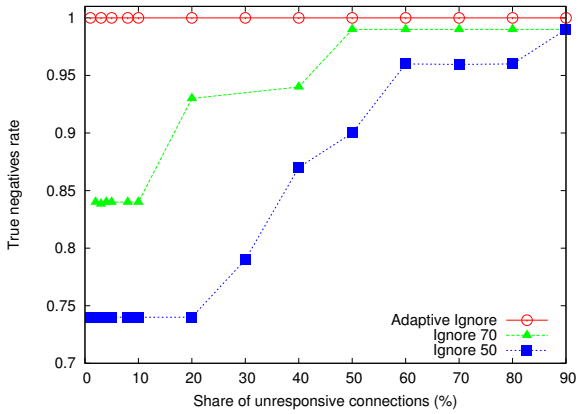


Figure 8.9: ROC Curve for the false negatives as a function of the share of unresponsive connections. The more unresponsive connections are present in the network, the higher the resulting accuracy is.

unresponsive over a longer time period. The true negatives rate also features a higher burstiness over time: around 300 Mbps the false negatives rate drops from 95% to 92.12% for the Ignore100 case and 92.12% for the AdaptiveIgnore case. This drop in accuracy was visible in each experiment and is because of the high unresponsiveness of the connections, which sometimes leads too high network loads, especially during the start-up phase. Similar drops were visible in other experiments and happened there sooner or later depending on the share of unresponsive connections.

Although the false negatives rate is lower than the false positives rate, in both types of unresponsiveness, the accuracy is still more than acceptable: the lowest experienced true negatives rate is 90.81%. Similar to the false positives rate, once the start-up phase is completed and the number of connections is stabilized, the accuracy converges to 100% and newly arriving connections can be mapped to the right label quickly. As the true negatives rate converges to 100% around 900 seconds, the impact of the EWMA algorithm, that averages the connection's CWR and ECE count, is also visible. Since we average with a weight equal to 0.99, there is a delay before the accuracy converges to 100%: lower weight values resulted in smaller delays but instable accuracies.

When comparing both types of unresponsive stacks, the results show that the Ignore100 stacks are easier to detect since the obtained true positives and true negatives rate are higher. With respect to the true positives rate, no false positives even occur in the detection of unresponsive Ignore100 stacks. This is because the AdaptiveIgnore connections are more subtle in their responsiveness: since they continuously switch between responding to ECN signals (when the queue is planning to drop packets) and not responding to ECN signals (when the queue is not planning to drop packets). As such, AdaptiveIgnore connections introduce more noise into the detection attributes, the CWR and ECE count, which complicate the clustering and impact the overall accuracy.

Influence of the share of unresponsive connections Figure 8.9 illustrates the impact of an increasing share of unresponsive connections in the network on the average true negatives rate during the steady state (i.e. after 600 seconds). Here the true negatives rate of three unresponsive

	Ignore50		Ignore70	
	1 connection	5 connections	1 connection	5 connections
True negatives (without outliers)	48.12%	68.24%	57.56%	69.01%
True negatives (with outliers)	74.62%	74.68%	84.47%	84.14%
True positives (with outliers)	97.57%	99.89%	98.37%	99.98%

Table 8.1: True negatives rate, with and without the detection of outliers, and true positives rate for a varying share of unresponsive connections and unresponsive types.

connection types, AdaptiveIgnore, Ignore70 and Ignore50, are shown as a function of the share these unresponsive connections have in the network. The share was varied by first introducing unresponsive clients in the network subset with only one bottleneck and, for a share of 50% and higher, also introducing unresponsive connections in the network subset with both bottlenecks.

While a stable accuracy of 100% can be obtained for the Adaptive Ignore case, regardless of the share of Adaptive Ignore connections, the other 2 types of unresponsive connections, Ignore50 and Ignore70, do suffer from a drop in accuracy when the share of unresponsive connections is small. The reason for this is the following: although the Adaptive Ignore already introduces more noise into the clustering attributes, its mean average accuracy still converges to 100% as long as the attributes are thoroughly averaged over time. If only a fraction of ECN messages are ignored, which is the case for the Ignore50 and Ignore70 cases, the introduced noise is higher: for example, in the Ignore50 case, the connections can be regarded as being unresponsive only 50% of the time. As a result, the cluster groups found by the detection algorithm are positioned closer to each other, complicating the clustering process. If, additionally, the number of unresponsive connections is small it is even harder for the detection algorithm to determine whether these points belong to the cluster borders, or form a new cluster.

Because of this, the true negatives rate increases as the share of unresponsive connections increases as well. If only one 1 to 10% of the connections are unresponsive the true negative rate is 84,72%, for the Ignore70 case, and 74,62%, for the Ignore50 case. This increases for a growing share to true negatives rates of 90% and higher when more than 50% of the connections are unresponsive. Also note that, the gain of only a few unresponsive connections is significantly lower than that of higher shares of unresponsive connections. This was discussed in Section 8.3.1, which described the original problem.

To illustrate the need for the outlier detection algorithm, the algorithm that is specifically intended to detect a small number of unresponsive connections, Table 8.1 illustrates the true negatives rate with and without the outlier detection algorithm. As shown, the outlier detection results in a significant gain in true negatives rate, up to 25%, because without the outlier detection, most single unresponsive connections are not detected. Table 8.1 also shows the true positives rate for the same share of unresponsive connection types. Similar to the previous results, it is shown that the number of false positives is considerably lower than the number of false negatives. Even for the lowest shares of unresponsive connections, the true positives rate is close to 100%.

Influence of the level of responsiveness The results in the previous section already indicated that the level of responsiveness has an impact on the overall detection accuracy. This is illustrated in more detail in Figure 8.10 which shows both the true positives rate and false negatives rate as a function of the level of responsiveness. In this case, 30% unresponsive connections

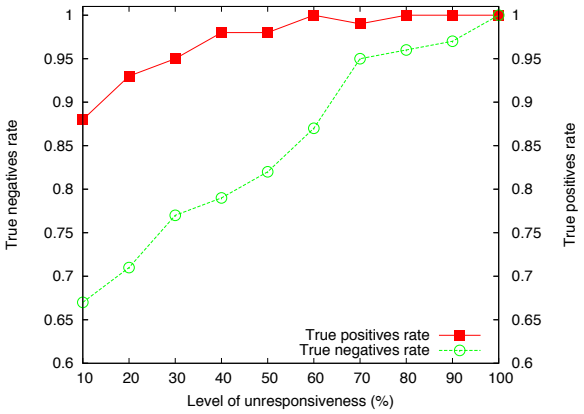


Figure 8.10: ROC Curve for the false negatives and false positives as a function of the level of unresponsiveness. The level of unresponsiveness is varied by changing the ratio of ECN messages that are ignored.

were introduced; the level of responsiveness was varied through unresponsive connections of type IgnoreX, where the X was varied. Hence, by increasing the fraction of ECN messages that are being ignored, the responsiveness is decreased.

As expected, an increasing unresponsiveness has an increasing effect on the accuracy in terms of both the false positives and false negatives rate. The less ECN signals are ignored by the unresponsive connections, the more these connections resemble regular - responsive - connections. This means that it becomes harder to adequately detect such unresponsive connections, but also for the unresponsive connections themselves to achieve an actual gain out of being unresponsive.

Similar to the previous results, the true positives rate is again considerably higher than the true negatives rate. While both feature a close to linear increase as the level of unresponsiveness increases, an unresponsiveness of 50% and more results in a false positives rate of approximately 100% and remains stable as the unresponsiveness further increases. The true negatives rate is also 100% when all ECN messages are ignored, but is considerable lower, i.e. 66.12%, when only 10% of all ECN messages are ignored. While this latter true negatives rate is low, the false negatives rate steadily increases: at 50% of unresponsiveness, the false negatives rate is 79.91%. As will be detailed in the next section, this is the moment at which the unresponsive connections start experiencing an actual gain of being unresponsive.

Penalization evaluation results While the previous section focused on an off-line detection accuracy evaluation, in this section, we characterize the gain of of the complete system, including the penalization action proposed in Section 8.3.5. Therefore, the detection algorithm was deployed on-line and its output was provided to the penalization algorithm that includes the exponential back-off algorithm with T initially set to 5 seconds. The connection monitoring step generated an updated statistics value every second and also triggered, if needed, the detection algorithm and, ultimately, the penalization action. We characterize the impact of penalization on both a limited and large connection set and investigate the influence of the level of unresponsiveness.

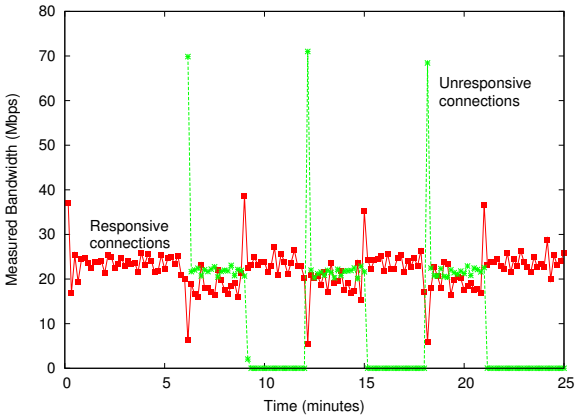


Figure 8.11: Influence of the penalization on the scenario described in Figure 8.2, representing four responsive connections and one unresponsive connection. By applying penalization to the unresponsive connection, the unfair throughput of the unresponsive connection is immediately decreased, leading to an almost perfect fairness level.

Influence of the penalization on a limited connection set

In this experiment, the same settings as described in Section 8.3.2 were used, where 5 connections were forwarded over a 100 Mbps bottleneck. In this case, we penalized the unresponsive connection that is started every 6 minutes once its unresponsiveness was detected by the cognitive accountability mechanism. Figure 8.11 shows the corresponding measured bandwidth of the unresponsive connection as well as the responsive connections. As can be seen, the introduction of an unresponsive connection into a pool of responsive connections starts with a similar behavior as experienced without penalization: the unresponsive connection is able to achieve a much higher throughput than the responsive connections. However, unlike the case without penalization discussed in Section 8.3.2, this unfairness does not last the complete length of the unresponsive connection. Immediately after its introduction (i.e., approximately 10 seconds later), the unresponsiveness of the connection is detected and penalized. This penalization has a clear impact on the throughput of the connections and corresponding fairness. After the initial unfair peak in throughput, the throughput of the unresponsive connection immediately decreases to values that are similar to those obtained from the responsive connections. Therefore, applying penalization allows to restore the fairness that should exist between connections, even when the connections are highly unresponsive; the reaction speed is lower than 10 seconds.

Influence of the penalization on a large connection set To characterize the impact of the penalization actions on a larger connection, we again use the gain metric, introduced in Section 8.3.1. We use this metric and not the Jain's fairness index introduced in Section 8.3.4 because the Jain's fairness index does not take into account the knowledge we have about unresponsive and responsive connections. Therefore, the gain metric provides a better understanding of the actual unfairness between the two classes, instead of the global unfairness, where the multiple bottlenecks also contribute to. Figure 8.12 illustrates the evolution of this gain value over time, measured during the steady state of the experiment, thus ignoring the start-up phase. In this experiment, 30% of the connections ignored all ECN messages (i.e. the Ignore100 type). Furthermore, we turned the penalization action on after 100 seconds to identify the effect of

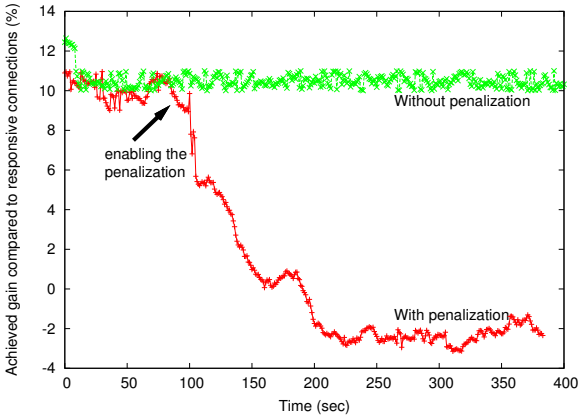


Figure 8.12: Gain in terms of measured bandwidth of unresponsive connections versus responsive connections over time, with and without penalization of unresponsive connections. The penalization is turned on after 100 seconds. Applying penalization limits the gain of unresponsive connections.

the penalization. Figure 8.12 shows both the gain with the penalization turned off completely and turned on after 100 seconds, evaluated in two separate experiments. As shown, without any penalization, the unresponsive connections achieve a gain of approximately 11.5%. This means that the average throughput of the unresponsive connections is 11.5% higher than those of the responsive connections.

By enabling the penalization of connections at 100 seconds, this gain slowly but steadily decreases over the course of 100 seconds. The ultimately achieved gain by enabling penalization is close to zero but also more bursty than when no penalization is applied. This is because of the exponential back-off algorithm that periodically probes to check if responsive connections haven't been misclassified as unresponsive. By moving these connections from the unresponsive class to the responsive class again, unresponsive connections have the chance of achieving a higher gain for a short period.

As time elapses, the achieved gain even decreases below zero with an average of approximately -2%. This is because the penalization action slightly favors the responsive connections as they can still receive ECN warnings. Once the queue starts marking ECN packets, the responsive connections receive these warnings and can send a CWR message to ask the sender for a small decrease in its transfer rate. The unresponsive connections, on the other hand, immediately experience data loss, which leads to a more drastic reduction in the sender rate (typically through a multiplicative decrease algorithm). When the unresponsive connections ignore all ECN warnings the cognitive accountability algorithm is thus able to restore the fairness between connections by decreasing the achieved gain of the unresponsive connections with more than 12%.

Influence of the level of unresponsiveness

Figure 8.13 illustrates the average achieved gain for an increasing level of unresponsiveness. Similar to the previous experiment, 30% of the connections were IgnoreX unresponsive connections, where the value of X was varied. The results show that when more than 50% of all ECN signals are ignored (i.e. the unresponsiveness is below 50%), the gain the unresponsive

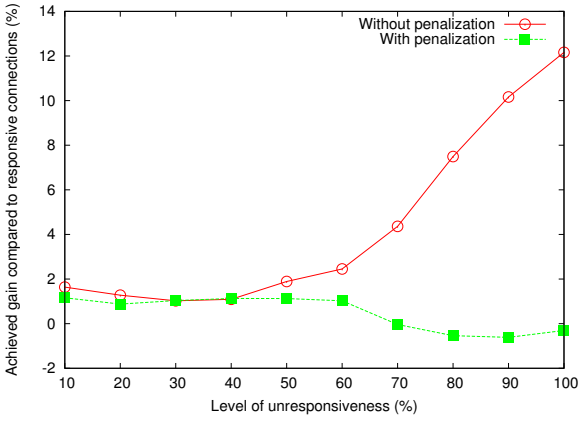


Figure 8.13: Influence of the level of unresponsiveness on the gain unresponsive connections achieve, with and without penalization of unresponsive connections.

connections can achieve is negligible, being less than 2%. Because of the smaller true negatives rates reported in Figure 8.10, the use of penalization is also not able to decrease this last 2% of gain, which makes that applying penalization has little effect for connections with an unresponsiveness lower than 50%.

As the level of unresponsiveness increases up to a point where all ECN messages are ignored, the achieved gain unresponsive connections obtain, if no penalization is applied, also increases up to 11.5%. The cognitive accountability algorithm is able to respond to this by penalizing the unresponsive connections. Instead of an increased gain, applying penalization results in a slight decrease in gain until the unresponsive connections even have a disadvantage compared to the unresponsive connections, illustrated through a negative gain value.

8.3.7 Conclusions

In this section, we presented a cognitive accountability mechanism that is able to restore the fairness between TCP connections, originally affected by the presence of unresponsive TCP stacks that (partially) ignore ECN congestion signals. The cognitive accountability mechanism is deployed on top of a normal AQM enabled router and consists of a detection algorithm and differentiated AQM mechanism. The detection algorithm monitors each connection passing through the router and performs a combination of clustering and outlier detection based on the monitored data. The detection algorithm maps the connections to different groups, called profiles, that feature similar ECN behavior and labels each profile as responsive or unresponsive. The output of the detection algorithm is used to perform a differentiated AQM behavior that divides connections into a responsive and unresponsive class and penalizes the unresponsive class by disabling the ECN support.

The cognitive accountability mechanism has been evaluated on a testbed facility emulating a tree-based network topology with multiple bottlenecks and 400 client terminals. Different evaluation scenarios were introduced by varying the type of unresponsive connections, the share of unresponsive connections and the level of unresponsiveness, featured by the unresponsive connections. Both the accuracy of the detection algorithm and the achieved fairness gain of the overall accountability mechanism has been characterized. The results show that, the detection accuracy depends on both the level and share of unresponsiveness. Moreover, a drop in

accuracy typically has an impact on the number of false negatives and not on the number of false positives. As such, lower accuracy values only result in unresponsive connections being classified as responsive connections and not the other way around. Overall, the detection accuracy is higher than 80% and often close to 100% in cases where the unresponsive connections achieve an actual gain from being unresponsive. Furthermore, the results showed that penalizing detected unresponsive connections can successfully limit the gain unresponsive connections achieve compared to responsive connections to 2% and less. The more gain unresponsive connections achieve by being unresponsive, the better unresponsive connections are penalized, leading to better fairness values. Our experiments showed that unresponsive connections that experience throughputs that are 5 times higher than their responsive counterparts, fall back to the same throughputs of responsive connections by applying penalizations. Thus, the results show that the clustering and outlier detection of ECN statistics of a flow provide a timely and accurate detection of unresponsive ECN based TCP stacks. Moreover, the penalization of the unresponsive connections does indeed lead to a better overall network fairness.

8.4 Unresponsive senders

In this section, we focus on the problem of unresponsive TCP senders. Different levels of responsiveness in terms of congestion warnings can cause different throughputs between the TCP stacks, when all other network environmental settings are the same. As such, these differences in throughput can violate the network fairness principle TCP tries to achieve.

Often, different responsiveness levels between TCP senders are a side effect of differences in TCP dialects, which incorporate slight modifications to the original TCP congestion avoidance algorithm. These modifications are often motivated because the different TCP dialects are optimized for specific use cases (e.g., lossy links, delay tolerant networks). For example, an Additive Increase Multiplicative Decrease (AIMD) TCP dialect such as TCP CUBIC will be less aggressive in increasing the throughput than a Multiplicative Increase Multiplicative Decrease (MIMD) TCP dialect such as Scalable TCP.

The goal of this section is to investigate how different responsiveness levels of different TCP dialects have an effect on the flow characteristics. By investigating different attribute sets of flow characteristic, a distinguishing attribute set can be selected between these different TCP dialects. As such, this research serves as the basis for a metric that distinguishes between different levels of responsiveness.

8.4.1 Experimental setup

Topology

In our experiments, we use two versions of the modeled topology: a virtual and physical topology. The virtual topology is the one we wish for our emulation tests and maps to the physical topology by grouping similar node functionality on one single physical node. The physical topology is what we use to emulate it: the rationale behind these two different topologies is a severe node reduction required for the experiments.

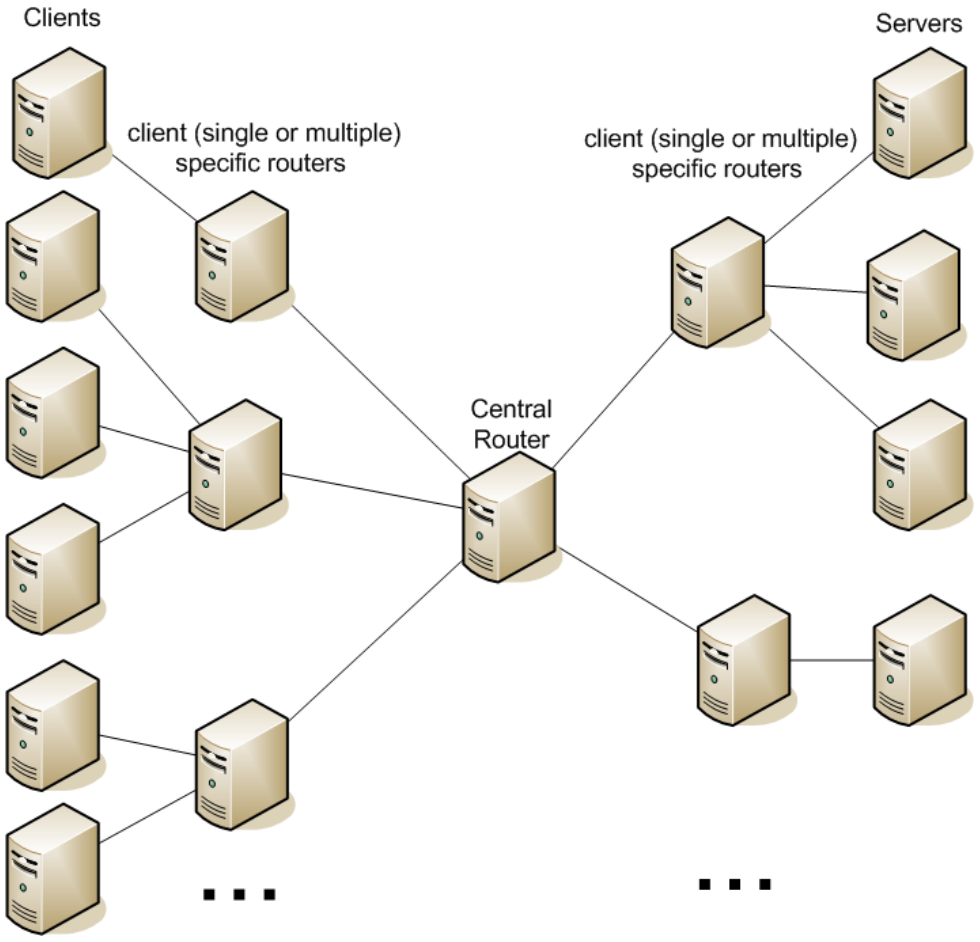


Figure 8.14: Example virtual network

Virtual topology

Figure 8.15 illustrates the used virtual network topology. The topology allows depicts a butterfly topology, where both wings represent servers or clients and are connected through one central router. Each connection goes from a server to a client through this router, and all servers and clients can be distinct (but need not be for all tests). There are bottlenecks possible that affect a number of connections together (i.e. they go through the same router), upstream and downstream from the router. Network delay both upstream and downstream from the router can be distinct for each connection. The TCP stacks used on each server and client can be configured.

Physical topology The physical topology as illustrated in Figure 8.14 is much simpler as various virtual nodes are mapped to a physical node. We have a physical server for each TCP stack, representing multiple servers, and a client for each TCP stack, representing multiple clients. In between them, we have a Click Modular Router, which emulates the rest of the network, including the central router.

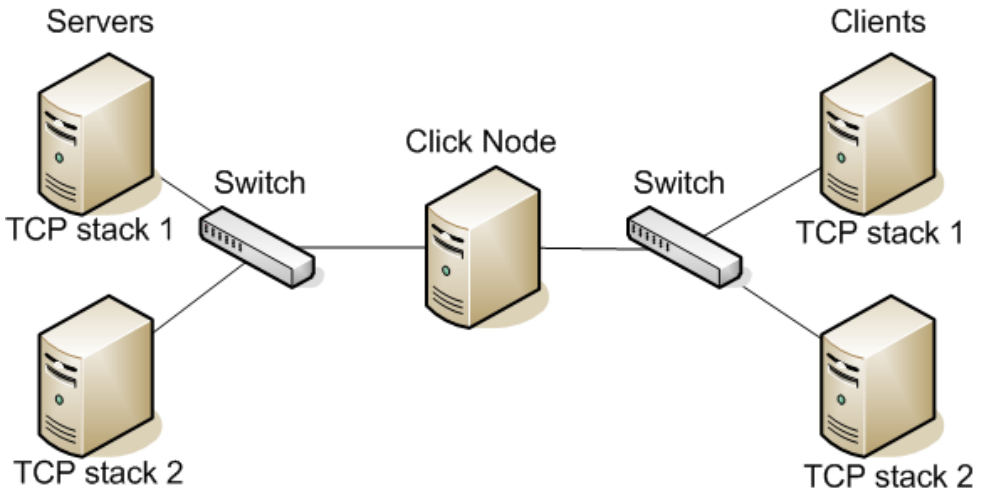


Figure 8.15: Example physical network setup

Software Details

Client and Server Software To emulate client and servers, a simple custom software application is used. This software creates a TCP connection between 2 hosts, and sends data at the maximum speed possible over this connection. The software can be configured to exit when a number of bytes have been sent, or a certain time has passed. This byte and time limit is configurable.

Network Monitoring Software Network traffic is monitored using software that uses the PCAP library. This software monitors the bandwidth consumed per user over specified periods of time. It also distinguishes and can filter on certain traffic parameters. The monitor node is connected directly to the Click Router, and receives a copy of all packets from it. This includes dropped packets, which are marked (by setting IP TTL to zero). This allows us to measure the exact amount of lost packets with the monitoring tool.

TCP Stacks In our experiments, we have used two different stacks, which correspond with two families of TCP stacks:

- Additive Increase, Multiplicative Decrease (AIMD) stacks is a family of TCP stacks that features a linear growth of the congestion window when no congestion warnings are observed. For this family of stacks, we used the TCP CUBIC stack implementation, which is the default stack used by the Linux kernels 2.6.19 and above.
- Multiplicative Increase, Multiplicative Decrease (MIMD) stacks is a family of TCP stacks that features an exponential growth of the congestion window when no congestion warnings are observed. For this family of stacks, we used the Scalable TCP stack implementation.

Both CUBIC TCP as Scalable TCP are explicitly designed for use in high speed networks. However, Scalable TCP is far less aggressive in increasing its congestion window, which allows it to increase a higher throughput. The goal of the experiments is to find a metric that allows

detecting this different behavior only if the difference in throughput is due to the higher aggressiveness of Scalable TCP. Both TCP stacks used are the Linux implementation versions of these stacks.

Click: Router with RED queue Emulation The Click Router emulates a RED queue using a modification of the default click RED element. This element is configured with the following RED parameters:

Minimum Threshold 300 packets

Maximum Threshold 100 packets

Queue Length 600 packets

Maximum Probability 0.02

Stability 9

Click: Network Emulation The Click Router emulates a complex network, with bottlenecks and delays before and after the RED queue. A combination of packet filters (to separate packets per client and server), delay adders and bandwidth limiters is used to implement this.

Virtual wall and WExO Experiments where performed on the IBBT virtual wall, an infrastructure to create network testbeds. We used WExO, a tool set to create and manage virtual wall experiments. WExO was explained in detail previously in deliverable D3.4.

Experimental settings

The goal of the tests is to discover a method to detect ”unfair” connections on an intermediate router. We will refer to this router as the ”ECODE router”. The term ”unfair connections” refers to any TCP connection that does not act ”fair” in the given network situation, and thus uses more bandwidth then it should. A number of typical network situations can cause traffic patterns similar to those of unresponsive stacks, so we need to distinguish those patterns from the ones caused by unresponsive stacks.

We will focus mostly on bandwidth consumption per connection. An unfair connection generally uses more bandwidth on average than a responsive one. Network situations such as bottlenecks and delay will also impact bandwidth per connection.

Our basic tests are:

- First test a scenario with a chosen number of unfair connections.
- Secondly, test the same scenario, without unresponsive stacks, but with a network situation that causes the same average bandwidth usage.

Then, examine the measured data and search a metric that can distinguish these 2 cases. For this, we focus on the introduction of server bottlenecks that cause the same average bandwidth usage. Note that there are other aspects that may cause these differences in bandwidth usage. We will discuss these in the remainder of this section, as part of future work.

Server Bottlenecks We will try to distinguish bottlenecks between the server and the ECODE router, and unresponsive stacks by enabling or disabling a bandwidth limitation (bottleneck) on the servers. This process works as follows: First we test without a bandwidth limitation. We start with an equal amount of CUBIC and Scalable TCP connections. We will notice that that the total average bandwidth of the Scalable TCP connections is higher than the total average bandwidth of the CUBIC TCP connections. In the second test, we use only CUBIC TCP connections, but we apply a server bottleneck to half of them, so they use only as much bandwidth as the fair connections in the previous case.

Parameters to vary as part of future work The results described in this deliverable focus mainly on the mechanism of varying the server bottleneck. As part of future work, we will do variations on the previous tests. The idea is to create a large number of tests, in which there are either Scalable TCP connections, or not, following a similar approach as explained above. Various combinations of parameters will be tested, such as connection duration, connection startup, share of Scalable TCP connections, etc. The chosen metric should be robust against these parameter variations. Some important parameter variations that should be investigated in future work are:

- Network delay: as the throughput of TCP is known to be inverse proportional with the Round Trip Time (RTT), the network delay is one of the parameters that should be varied. For this, we will use a similar approach as the one described in the previous section.First we test without a difference in network delay. We start with an equal amount of fair and unfair connections. We will notice that that the total average bandwidth of the unfair connection is higher then the total average bandwidth of the fair connections. In the second test, we use only fair connections, but we apply a network delay to half of them, so they use only as much bandwidth as the fair connections in the previous case. To do this we will evaluate a number of delay settings.

- Unfair connection mechanism: besides using the CUBIC TCP and Scalable TCP connections, other unresponsive and aggressive stacks can exist. For example, similar defective stacks as the ones discussed in the previous section can be used as well.

- RED settings: RED settings can also have a severe impact on the congestion behavior. Differences in these settings should also be investigated.

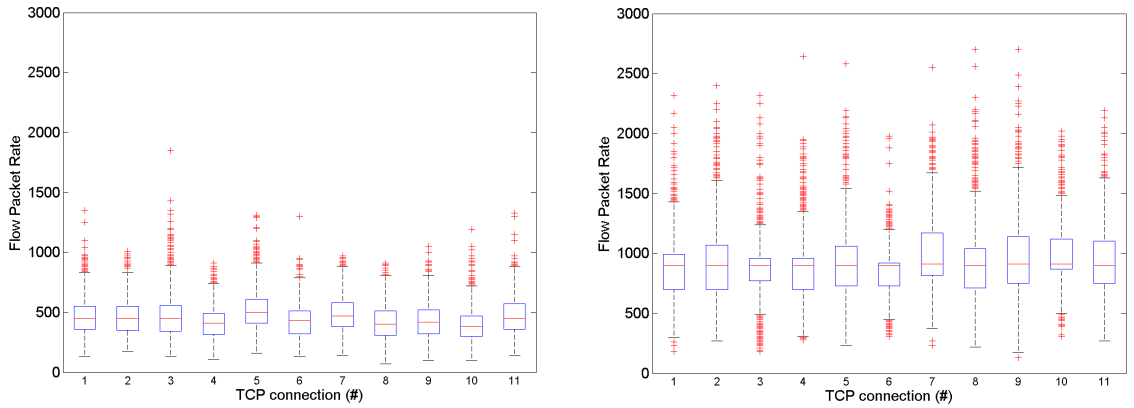
- Connection length: Currently, the connections are homogeneous. A study of heterogeneous connections is also required.

- The current study focuses on downstream connections from server to clients. However, two-way traffic may also influence the connection's throughput.

8.4.2 Results description

In order to distinguish the two scenarios, the following attributes per connection are monitored during the experimental network tests over a time period of 10 minutes.

- Flow packet (FP) rate (# flow packets / 100 ms)



(a) Box and whisker diagram (CUBIC TCP)

(b) Box and whisker diagram (Scalable TCP)

Figure 8.16: Box and whisker diagram for the different FP rates for CUBIC and Scalable TCP.

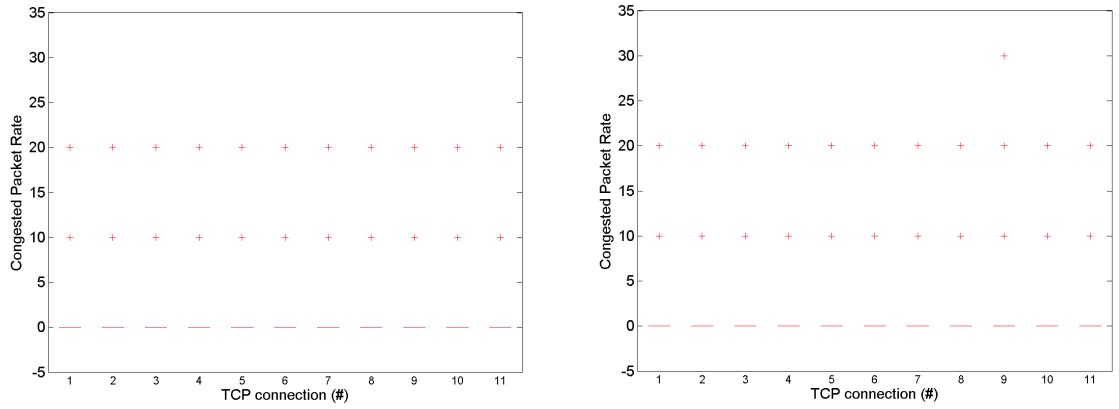
- Congested packet (CP) rate (# congested packets (i.e., packets that are ECN-marked) / 100 ms)

This 10 minutes time frame is subdivided into 4 separate time intervals, resulting in 4 different phases.

- Interval 1 [0 – 2.5 min] : interval when TCP connections are started (start-up phase)
- Interval 2 [2.5 – 5.0 min] : interval when TCP connections are generating traffic (stationary phase)
- Interval 3 [5.0 – 7.5 min] : interval when some TCP connections are added & stopped (non-stationary phase)
- Interval 4 [7.5 – 10.0 min] : interval when TCP connections are stopped (cool-down phase)

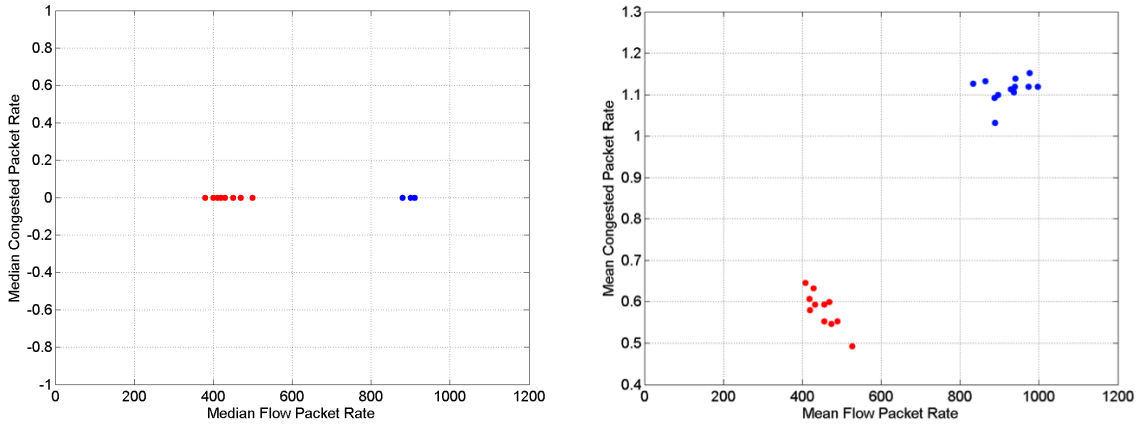
Figures 8.16(a) and 8.16(b) show a box-and-whisker diagram of the different FP rates for all the TCP connections in the case without bottleneck. Such a diagram depicts the sample minimum, lower quartile, median, upper quartile and sample maximum of TCP connections that are active in the stable phase (interval 2). Fig. 8.16(a) shows a diagram for TCP CUBIC connections, whereas Fig. 8.16(b) shows the result for the Scalable TCP connections. It is seen from the data that the CUBIC and Scalable TCP connections behave differently, and that they can easily be distinguished e.g. by comparing the skewness of the data, the length of the whiskers (dispersion of the data), and/or the median and mean of the sample set.

Figs 8.17(a) and 8.17(b) show similar diagrams for the CP rate. It is shown that this attribute provides less information, since most of the time, there is little or no congestion. Also, the CP rate varies in discrete steps of 10 CP/100 ms. Note however that the Scalable TCP connections have a higher average CP rate, due to the occurrence of a higher number of outliers (not visible in the figures, due to superposition of the markers). Based on these diagrams, it is expected that the FP rate is probably a more insightful attribute than the CP rate.



(a) Box and whisker diagram (CUBIC TCP) (b) Box and whisker diagram (Scalable TCP)

Figure 8.17: Box and whisker diagram for the different CP rates for CUBIC and Scalable TCP.

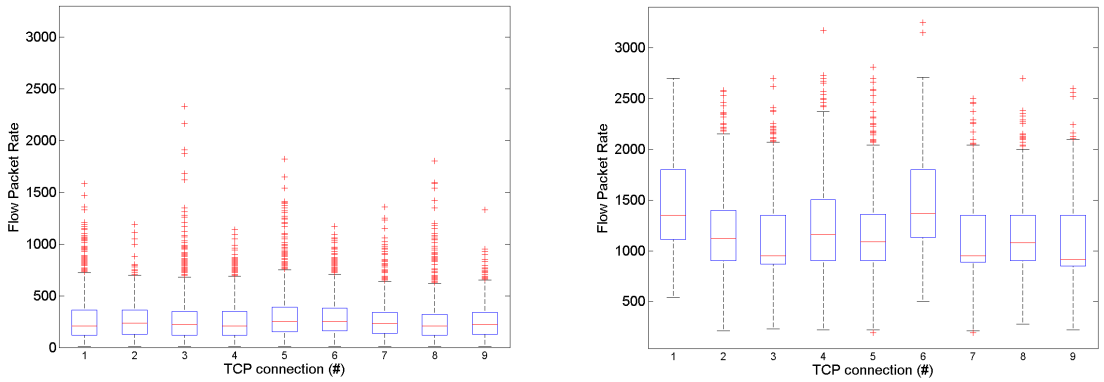


(a) Median (b) Mean

Figure 8.18: Median and mean FP rate vs. CG rate.

Since the median of the packet rates is substantially different for the CUBIC and Scalable TCP connections, it seems natural to select these variables as valid attributes for a potential clustering algorithm. As shown in Fig. 8.18(a) , there are two separate clusters that correspond exactly to the set of CUBIC and Scalable TCP connections. Due to the low congestion, it suffices to consider only the median FP rate, because the median CG rate is 0 for all TCP connections. One can take this variable explicitly into account by considering the mean FP rates and the (non-zero) mean CP rates as shown in Fig. 8.18(b) , as this leads to similar conclusions.

Note however that the goal is not to make a distinction between CUBIC and Scalable TCP connections, but to distinguish the case of unresponsive stacks, where more unresponsive Scalable TCP connections receive an actual unfair bandwidth compared to their more responsive CUBIC counterparts. In case, that a bottleneck has been placed between the server and the ECODE router a similar difference in throughput can be observed. The goal of the experiments is to find these attributes that can distinguish between these two scenarios. As an example, sim-



(a) Box and whisker diagram (CUBIC TCP)

(b) Box and whisker diagram (Scalable TCP)

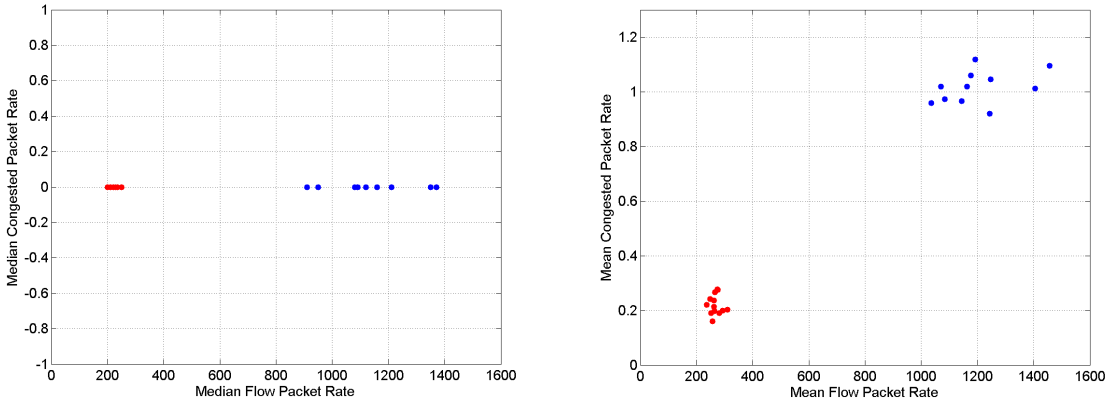
Figure 8.19: Box and whisker diagram for the different FP rates of CUBIC TCP and Scalable TCP. In this case, an additional bottleneck has been placed on the server, which diminishes the differences in unresponsiveness between the connections.

ilar box-and-whisker diagrams are made for a scenario in where the Scalable TCP connections are behind an additional bottleneck of 160Mbps and the CUBIC TCP connections are behind a bottleneck of 40Mbps. This difference in throughput (160 Mbps vs 40 Mbps) is the same throughput difference one can observe without such a bottleneck and only the interaction between the unresponsive Scalable TCP and the more responsive CUBIC TCP connections. Fig. 8.19(a) and 8.19(b) illustrate that the shape of the distributions is similar. Using the median or mean FP and CG rates accurately separates the CUBIC and Scalable TCP connections, however it also leads to a set of 2 distinct clusters with a comparable mutual positioning as shown in Fig. 8.20(a) and 8.20(b) . **This is not the desired behavior for an accurate responsiveness metric**: as in this case, the differences in throughput are due to an server bottleneck, the attributes should **not** be able to distinguish between these two type of connections. Experimental results indicate that actual differences between the results (e.g. size of clusters, distance between clusters, . . .) are mainly caused by changes in the network topology. A further analysis is given in the next section.

Approach 1 : Clustering TCP connections based on values of packet rates In order to quantify any statistically relevant information between the two difference cases, the following approach is considered. Each attribute constitutes a time-dependent trace of real values, describing flow packet (FP) rate or congested packet (CP) rate of the individual TCP connections. In order to deal with the temporal behavior of the data, it is investigated if the time-dependent behavior of the flows can be characterized by a statistical distribution that denotes the probability that one of the attributes (acting as a random variable) takes certain values over time. The goal is then to replace the time-dependence of the attribute by estimated parameters of the distribution, and then to use a cluster-based analysis to distinguish the two cases using these parameters of the distribution.

Step 1 : Fit statistical distribution that best describes the data

A closer inspection of Fig. 8.16(b) indicates that the statistical distribution of FP rate for the Scalable TCP connections appear to be substantially different. Therefore, the probability that



(a) Median (b) Mean

Figure 8.20: Median and mean FP rate vs CG rate.

certain packet rates occur over time is unlikely to be the same for each individual connection. The FP rates of the CUBIC TCP connections appear to behave in a more consistent way, so it will first be investigated which statistical distribution would be the most appropriate one to fit the data. A wide range of 60 statistical distributions is considered and their parameters are optimized in such a way that the distribution of the FP rates over time is matched as closely as possible. Note that this requires an accurate estimation of distribution parameters. For this purpose, different methods are used, such as the maximum-likelihood estimation, the methods of moments and a least-squares approach. An overview of the considered statistical distributions is given in Table 8.2, as well as their estimated parameters.

Table 8.2: Fitting results

#	Distribution	Parameters	KS test (D)	KS test (Rank)	AD test (A^2)	AD test (rank)
1	Beta	α1=3.0774 α2=8.448 a=130.0 b=1350.0	0.06765	16	8.117	23
2	Burr	k=1.5113 α=4.6837 β=492.29	0.0552	4	2.3412	1
3	Burr (4P)	k=0.65494 α=0.7031 β=13.733 γ=130.0	0.62131	57	891.87	56
4	Cauchy	θ=81.522 μ=432.21	0.09062	28	21.768	38
5	Chi-Squared	ν=455	0.36835	50	3141.7	57
6	Chi-Squared (2P)	ν=11445 γ=-10989.0	0.10431	34	12.047	27
7	Dagum	k=134.97 α=3.4883 β=16.61	0.95562	59	7209.6	59
8	Dagum (4P)	k=2.3996 α=0.28607 β=0.48911 γ=130.0	0.58527	56	706.23	55
9	Erlang	m=8 β=51.03	0.17437	43	80.767	41
10	Erlang (3P)	m=8 β=52.22 γ=19.396	0.09893	31	14.762	33
11	Error	k=1.1792 θ=152.5 μ=455.75	0.10135	32	14.735	32
12	Error Function	h=0.00464	0.89759	58	6235.0	58
13	Exponential	λ=0.00219	0.37641	51	322.32	49
14	Exponential (2P)	λ=0.00307 γ=130	0.29149	48	205.69	46
15	Fatigue Life	α=0.34351 β=430.33	0.07011	18	5.2589	19
16	Fatigue Life (3P)	α=0.24459 β=596.19 γ=-158.27	0.05872	8	2.9723	8
17	Frechet	α=3.5954 β=367.14	0.13789	42	39.693	39
18	Frechet (3P)	α=1.9700E+8 β=2.5363E+10 γ=-2.5363E+10	0.06236	12	4.6541	17
19	Gamma	α=8.931 β=51.03	0.06274	13	3.5198	13
20	Gamma (3P)	α=8.3561 β=52.22 γ=19.396	0.06091	10	3.2126	9
21	Gen. Extreme Value	k=-0.06823 θ=127.33 μ=390.35	0.05422	2	2.7718	5
22	Gen. Gamma	k=1.006 α=9.0541 β=51.03	0.06312	14	3.2279	10
23	Gen. Gamma (4P)	k=0.91854 α=10.313 β=35.108 γ=8.679	0.06099	11	3.2413	12
24	Gen. Pareto	k=-0.5498 θ=328.71 μ=243.65	0.08223	26	341.1	50
25	Gumbel Max	θ=118.91 μ=387.12	0.07108	19	4.7059	18
26	Gumbel Min	θ=118.91 μ=524.38	0.17643	45	88.156	43
27	Hypersecant	θ=152.5 μ=455.75	0.10366	33	13.462	29
28	Inv. Gaussian	λ=4070.3 μ=455.75	0.0671	15	4.5374	15
29	Inv. Gaussian (3P)	λ=10333.0 μ=618.2 γ=-162.45	0.10645	38	13.713	31
30	Johnson SU	γ=-2.3472 δ=2.6444 λ=259.24 ξ=174.84	0.05468	3	2.4079	2
31	Kumaraswamy	α1=2.2425 α2=312.4 a=126.18 b=4948.1	0.07603	24	7.5506	22
32	Laplace	λ=0.00927 μ=455.75	0.10533	36	19.861	37
33	Levy	θ=406.38	0.50071	55	504.13	54
34	Levy (2P)	θ=239.72 γ=124.58	0.4278	54	356.46	51
35	Log-Gamma	α=322.07 β=0.01884	0.07558	23	5.913	20
36	Log-Logistic	α=5.3116 β=430.84	0.0722	20	3.6273	14
37	Log-Logistic (3P)	α=6.657 β=543.35 γ=-106.39	0.05356	1	2.4428	3
38	Log-Pearson 3	α=45.925 β=-0.04988 γ=8.3573	0.0608	9	3.2399	11
39	Logistic	θ=84.079 μ=455.75	0.10495	35	11.603	26
40	Lognormal	θ=0.33793 μ=6.0665	0.06822	17	4.581	16
41	Lognormal (3P)	θ=0.24616 μ=6.3761 γ=-150.02	0.05734	7	2.8286	7
42	Nakagami	m=1.9589 &=2.3095E+5	0.07292	21	13.486	30
43	Normal	θ=152.5 μ=455.75	0.10647	39	12.972	28
44	Pareto	α=0.83405 β=130	0.39128	52	359.67	52
45	Pareto 2	α=105.99 β=43680.0	0.40939	53	373.06	53
46	Pearson 5	α=8.6075 β=3497.9	0.09028	27	10.812	25
47	Pearson 5 (3P)	α=29.277 β=22387.0 γ=-336.0	0.05638	6	2.7148	4
48	Pearson 6	α1=204.92 α2=9.0156 β=17.925	0.09143	29	10.643	24
49	Pearson 6 (4P)	α1=25.733 α2=36.039 β=782.93 γ=-119.7	0.05626	5	2.8059	6
50	Pert	m=346.7 a=130.0 b=1350.0	0.11596	40	55.1	40
51	Power Function	α=0.65383 a=130.0 b=1350.1	0.33609	49	279.08	47
52	Rayleigh	θ=363.64	0.1758	44	87.72	42
53	Rayleigh (2P)	θ=254.83 γ=129.2	0.09328	30	17.156	35
54	Reciprocal	a=130.0 b=1350.0	0.24623	47	199.65	45
55	Rice	ν=424.44 θ=159.37	0.07465	22	14.774	34
56	StudentÕs t	ν=2	0.99997	60	17187.0	60
57	Triangular	m=156.0 a=130.0 b=1350.0	0.24023	46	160.57	44
58	Uniform	a=191.61 b=719.89	0.1166	41	311.28	48
59	Weibull	α=3.7174 β=502.95	0.10552	37	17.779	36
60	Weibull (3P)	α=2.2622 β=371.51 γ=126.38	0.07608	25	7.1075	21

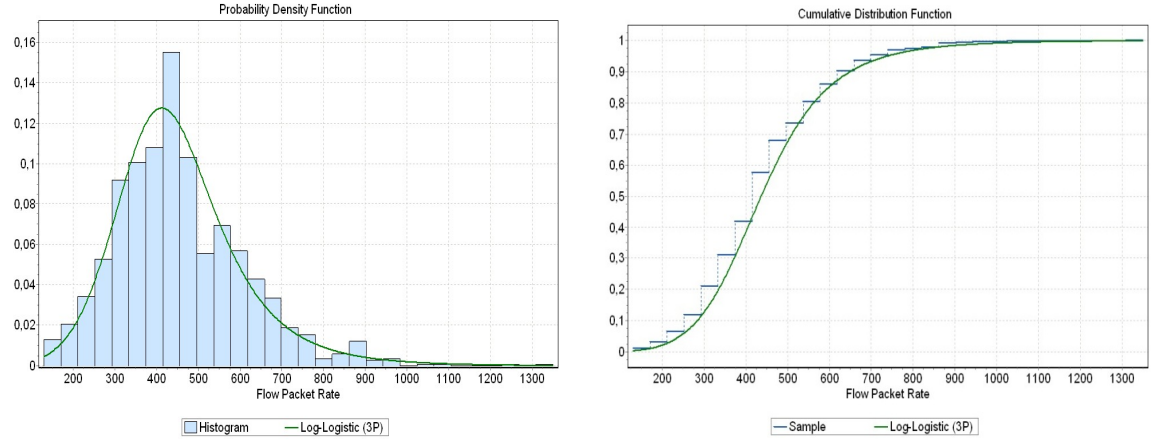
Step 2 : Use a statistical test to determine the goodness-of-fit

The goodness of fit tests measures how well a hypothesized distribution function matches the population of packet rate samples that were monitored over time. In this case two measures were considered to assess the goodness of fit : the Kolmogorov-Smirnov test statistic (D) and Anderson-Darling test statistic (A^2).

1. The Kolmogorov-Smirnov test statistic computes the largest vertical difference between a theoretical cumulative distribution $F(x)$ and the empirical cumulative distribution function $F_n(x)$ derived from data

$$D = \sup_x |F_n(x) - F(x)|$$

2. The Anderson-Darling test statistic compares the fit of an observed cumulative distribution function to an expected cumulative distribution function. This test gives more weight



(a) Histogram of empirical data & PDF of Log-Logistic (3P) distribution

(b) Comparison of cumulative distribution function (Log-Logistic 3P)

Figure 8.21: Histogram and cumulative distribution functions.

to the tails than the Kolmogorov-Smirnov test statistic. If the data samples $x_1 < ... < x_n$ are ordered, then the Anderson-Darling test statistic (A^2) is defined as follows

$$A^2 = -n - \sum_{k=1}^{n} \frac{2k-1}{n} [\ln F(x_k) + \ln(1 - F(x_{n+1-k}))]$$

Table 8.2 shows the results of both the D and A^2 test statistics and the corresponding ranking for an arbitrary CUBIC TCP connection. It turns out that both statistical tests yield somewhat different results, however some distributions are obviously more suitable than others (see e.g. the three-parameter Log-Logistic distribution, the Johnson SU distribution, or a Burr distribution). By comparing a histogram of the empirical data with the probability density function of the Log-Logistic distribution (Fig. 8.21(a)) and by inspecting the cumulative distribution functions (Fig. 8.21(b)), it is seen that a fair agreement is obtained, showing that this distributions is suitable to fit empirical data that is "nearly Gaussian".

In order to determine the statistical relevance of this outcome, the null hypothesis (H_0) and alternative hypothesis (H_A) are defined are follows :

- H_0 : the data follows the specified distributional form
- H_A : the data does not follow the specified distributional form

The null hypothesis H_0 regarding the distributional form is rejected if the test statistic, D or A^2, is greater than the critical value obtained from a table at a certain significance level α. (Note that there are several variations of these tables in the literature that use somewhat different scalings for the Kolmogorov-Smirnov test statistic and critical regions. These alternative formulations should be equivalent, but it is ensured that the test statistic is calculated in a way that is consistent with how the critical values were tabulated).

Table 8.3 lists the critical values for the statistics calculated for various significance levels (α), as well as the acceptance of the null hypothesis for each of the level values. It turns out that

Table 8.3: Goodness of fit statistics (Log-Logistic 3P)

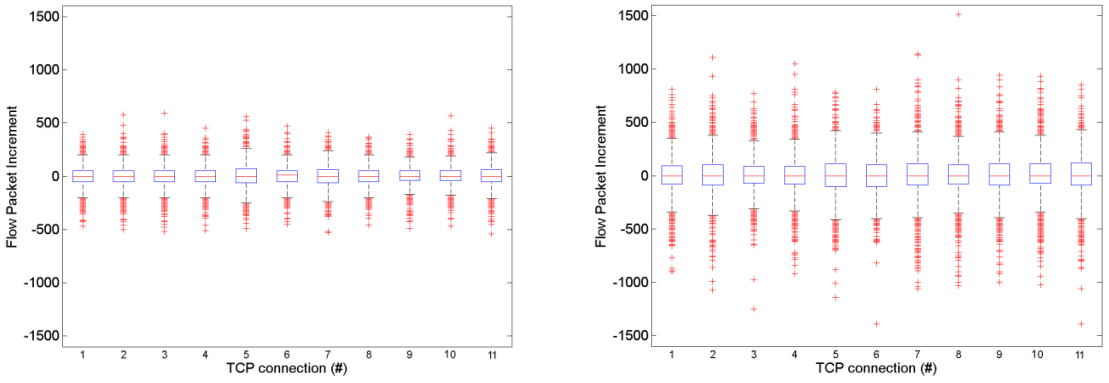
Log-Logistic (3P) [#37]					
Kolmogorov-Smirnov					
Sample Size	1501				
Statistic	0.05359				
P-Value	3.4608E-4				
Rank	1				
a	0.2	0.1	0.05	0.02	0.01
Critical Value	0.0277	0.03157	0.03505	0.03918	0.04205
Reject?	Yes	Yes	Yes	Yes	Yes
Anderson-Darling					
Sample Size	1501				
Statistic	2.4428				
Rank	3				
a	0.2	0.1	0.05	0.02	0.01
Critical Value	1.3749	1.9286	2.5018	3.2892	3.9074
Reject?	Yes	Yes	No	No	No

the null hypothesis H_0 is rejected at the 5% significance level using the Kolmogorov-Smirnov test for all distributions that were considered, even the ones that are highest ranked. Since it cannot be rejected at the 5% significance level using the Anderson-Darling test statistic, a similar analysis is performed on the other CUBIC TCP connections. It turns out that the ranking of distributions according to the goodness-of-fit statistics is substantially different for each TCP connection, even if only the CUBIC TCP connections are considered. Therefore, it is concluded that the packet rates which are gathered for the different TCP connections do not follow a single distributional form with acceptable statistical relevance.

Approach 2 : Clustering TCP connections based on increments of packet rates A limitation of the previous approach is that it relies on the probability that one of the attributes takes certain values over time, without taking the order of these values over time into account. Rather than analyzing the individual values of the FP rate and the CP rate, the increment or decrement (ΔFP(t) and ΔCP(t)) between successive time frames is considered. This concept is closely related to the responsiveness and aggressiveness metrics of the TCP connections.

$$\begin{aligned} \Delta\text{FP(t)} &= \frac{\text{FP}(t) - \text{FP}(t-1)}{\Delta t} \\ \Delta\text{CP(t)} &= \frac{\text{CP}(t) - \text{CP}(t-1)}{\Delta t} \end{aligned}$$

Note that the increments can also take negative values (i.e. the increment is a so-called decrement). Figs. 8.22(a) and 8.22(b) show a box-and-whisker diagram of the different FP increments for all the TCP connections in the case without bottleneck. It shows that the interquartile range of the Scalable TCP connections is much larger than the interquartile range of the CUBIC ones. This corresponds to the intuition that the Scalable TCP connections are more aggressive



(a) CUBIC TCP

(b) Scalable TCP

Figure 8.22: Box and whisker diagrams for FP increments (without server bottleneck).

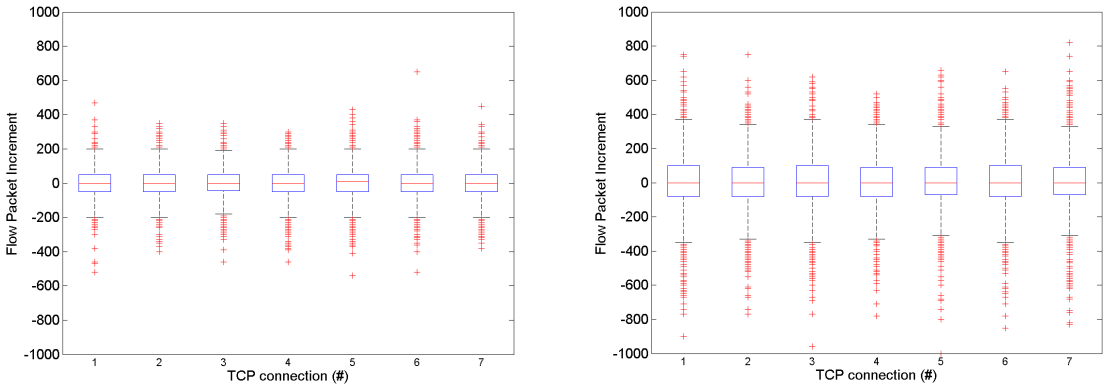
than CUBIC TCP connections. Also here, the CP increments appear to be less meaningful, since the interquartile range was 0 for all connections.

The distribution of the increments was also compared with the case of various bottleneck scenarios. For a case where the bottleneck was perfectly aligned with the observed throughput values. The distribution of the increments is similar in the bottleneck case and completely different in the non-bottleneck case. This indicates that the average length increment seems a potential good candidate for distinguishing between the two considered families of TCP stacks. However, in other cases, where the number of connections differs, see e.g. Figs. 8.23(a) and 8.23(b) for a scenario where 12 CUBIC connections are behind a server bottleneck of 40 Mbps and the other 36 Scalable TCP connections are behind a 170 Mbps server bottleneck, it was found that the connections behave again in a similar way as in the case without a bottleneck. Using a similar procedure along the lines of Approach 1, it was found that the increments also do not provide a conclusive answer to distinguish all the bottleneck cases in a consistent way.

Further study is needed to draw conclusions out of these results. More specifically, in future work it is needed to gain insights in the level of differences in throughput a different share of one connection over the other entails. If this difference is small, i.e. the connections are rather fair, than the observed similarity in the distribution between increments in Figures 8.22(a), 8.22(b), 8.23(a) and 8.23(b) are less of an issue and the distribution of increments can be used as a potential metric candidate. If not, the metric should be rejected.

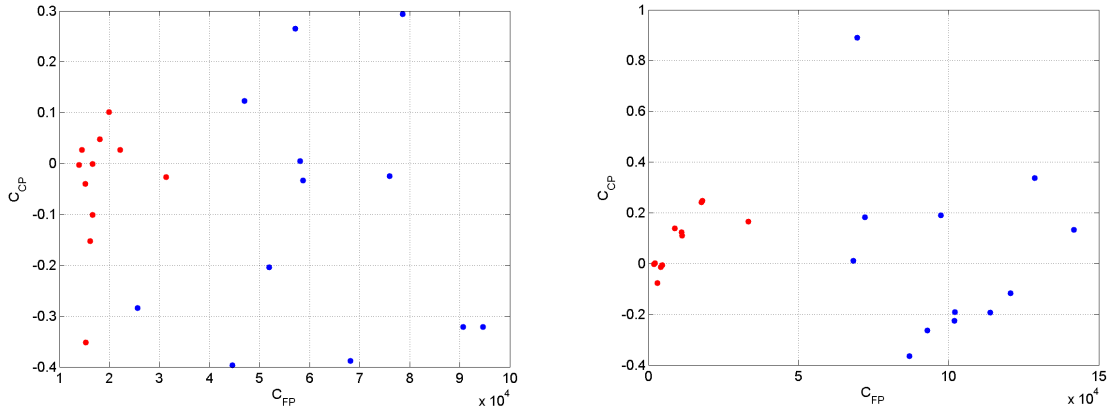
Approach 3 : Auto-covariance of the packet rates It has been investigated if the auto-covariance of the packet rates can be used to distinguish both cases. Given a FP of CP rate, the auto-covariance is defined as the covariance of the attribute with itself, i.e. the variance of the attribute against a time-shifted version of itself. If the attribute has mean $E[\mathrm{FP}(t)] = \mu_t$, then the auto-covariance $C_{FP}(t,s)$ with $s = t + \tau$ is given by the formula

$$\begin{aligned} C_{FP}(t,s) &= E[(\mathrm{FP}(t) - \mu_t)(\mathrm{FP}(s) - \mu_s)] \\ &= E[\mathrm{FP}(t)\mathrm{FP}(s)] - \mu_t \mu_s \end{aligned}$$



(a) CUBIC TCP (b) Scalable TCP

Figure 8.23: Box and whisker diagrams for FP increments (with server bottleneck).



(a) Without server bottleneck (b) With server bottleneck

Figure 8.24: Auto-covariance CFP versus CCP with and without server bottleneck.

Since the probability distribution does not change when shifted in time, the process is stationary and $\mu_t = \mu_s = \mu$. Hence,

$$C_{FP}(\tau) = E[\mathrm{FP}(t)\mathrm{FP}(t+\tau)] - \mu^2$$

A calculation of the auto-covariance $C_{FP}(\tau)$ and $C_{CP}(\tau)$ for all the TCP connections in the case of a bottleneck results in a set of data points that are scattered without a clear formation of clusters (see Fig. 8.24(a) and 8.24(b) for an illustration with $\tau = 1$). In the case of a bottleneck scenario (i.e., Scalable TCP connections experience a server bottleneck of 180 Mbps and CUBIC TCP connections experience a server bottleneck of 40 Mbps), a comparable set of scattered data points is obtained which does not provide a comprehensive pattern to distinguish both cases.

Approach 4 : Fourier transforms of the packet rates A set of N equispaced time domain packet rate samples $\{t_n, \mathrm{FP}(t_n)\}_{n=1}^{N}$ is transformed into a sequence of N equispaced frequency

domain packet rate samples $\{\omega_k, \mathrm{FP}(\omega_n)\}_{n=1}^{N}$ using the discrete Fourier transform.

$$\mathrm{FP}(\omega_k) = \sum_{v=0}^{N} \mathrm{FP}(t_v) e^{-(j2\pi/N)kv} \text{ for k} = 0, ..., \mathrm{N}-1$$

The frequency domain packet rate samples $\mathrm{FP}(\omega_n)$ are located at the discrete frequencies $\omega_k = k/(N\Delta t)$ where Δt denotes the time step. Therefore, they are equally spaced over a given frequency range $[0, \omega_{\max}]$, where $\omega_{\max}$ corresponds to the maximum frequency $(N-1)/(N\Delta t)$. Since the $\mathrm{FP}(\omega_n)$ rates are complex-valued, it offers various potential attributes that can be used in a clustering algorithm, such as the real or imaginary part of $\mathrm{FP}(\omega_k)$, as well as the amplitudes and phase component. Experimental results have shown that some of these attributes lead to a clustering of CUBIC and Scalable TCP connections in the frequency domain. Nevertheless, an application of the Fourier transform involves a lot of intricacies, e.g. the Fourier transform may require a long record of the packet rates in the time domain to ensure a sufficient frequency resolution, as zero-padding can lead to inaccurate results. This means that the length of the time interval plays an important role, and also the time step Δt has an influence on the results as it is directly related to the bandwidth of the $\mathrm{FP}(\omega_k)$ rate. Therefore, further analyses are needed to assess the potential of this approach.

8.5 Conclusions

We investigated several approaches to find a good attribute set that (i) is able to distinguish between AIMD and MIMD TCP stacks which are known to have different levels of aggressiveness and thus lead to significant throughput differences and (ii) does not distinguish between these stacks when there is not a difference in throughput or if the throughput difference is not due to the difference in aggressiveness but due to other factors such as the existence of a bottleneck. While tens of attribute sets were studied during this project, this deliverable focuses on the best 4 candidates found so far: a mapping on a statistical distribution, a distribution of increments, the auto-covariance and a translation to the frequency domain. We have shown that these attribute sets show promising results in achieving the desired behavior but are - at the same time - very prone the parameter fluctuations.

Due to the large problem domain, further study is needed to investigate the performance of these 4 candidate attribute sets. Therefore, future work will focus on a better exploration of the problem domain by taking two research directions. In the first research direction, it is necessary to better characterize the throughput differences between the stacks and to come up with a metrics that can characterize whether or not the differences in throughput are tolerable or not. In the second direction, the various parameters that can have an effect on the throughput (round trip time, heterogeneity of the connections, etc.) should be varied in more detail to investigate the robustness of the candidate attribute sets.

Chapter 9

Conclusion

This deliverable reports on the experimental evaluation of the machine learning engine by means of representative use cases and execution scenarios. Each of the experimental scenarios is used to perform functional and performance validation thanks to the definition of validation criteria and metrics defined in deliverable D4.1.

As conclusion, we summarize the properties of the machine learning engine (whose design is detailed in deliverable D2.3) as validated by experimentation:

Accuracy Accuracy appears as both monitoring accuracy (monitoring data matches actual traffic and events, or matches third-party monitoring traces) as well as detection and prediction accuracy (detection rates, number of false positives, etc.).

- Monitoring accuracy
  - Running monitoring applications based on adaptive sampling provides network status estimation accuracy measures, i.e., adaptive sampling lead to accurate estimation of actual network status.
  - Validation of the accuracy of the monitoring underlying the Anomaly Detection System, namely the NEWNADA system is achieved by means various datasets (MAWI, METROSEC, etc.) and its comparison against actual traffic where a monitoring card is used as base-line monitoring tool.
- Detection accuracy
  - Evaluation of the anomaly detection accuracy has been performed by means of the NEWNADA system by examining its resulting detection rates, false positive/negative rates, and rate of undetected attacks and their comparison with traffic traces.
  - Detection accuracy of path exploration events, as well as false positives and false negatives can be observed from the results obtained when applying sequential learning to on-line BGP data.
- Prediction accuracy
  - The path availability and performance service (IDIPS) does not lead to prediction accuracy issues (as IDIPS itself does not perform actual predictions). Nevertheless,

the way the measurement module is implemented can have an impact on the quality of the measurement and, subsequently on the prediction accuracy.

- DMFSGD (Decentralized Matrix Factorization based on Stochastic Gradient Descent), a novel Network Coordinate Systems (NCS) enables to predict unknown network performance metrics (typically delay, available bandwidth) from a relatively small number of measurements between some pairs of nodes. Using different criteria, the proposed DMFSGD algorithm either outperforms or is competitive to Vivaldi (the only coordinate system that has been adopted in a real Internet application).
- Prediction accuracy of shared risk groups, defined as rate of correct predictions, false positives and false negatives has been reported for SRGs containing up to 10 elements (depending on the overlapping of links between SRGs).

Correctness The usage of a machine learning algorithms impacts the correctness of the outcome and actions of networking techniques. The machine learning engine can be used to provide faster, more scalable solutions, in which case the outcome may suffer in correctness. Alternatively, the objective of using the machine learning engine to run machine learning algorithms may lie in reaching functionality that performs better than traditional techniques; in this case correctness is a performance metric.

- Correctness as requirement: the correctness of selected path and AS-paths may be impacted by the removal of BGP path exploration sequences.
- Correctness as performance gain: in the context of the evaluation of the Anomaly Detection System, correctness refers to the ability of the ADS to detect unknown (0-day) anomalies.
- Higher correctness (shorter paths) is reached by finding routing shortcuts which can be found from the NCS using a triangular inequality violation detector.

Timing Timeliness of actions triggered by the machine learning engine output generally has a direct impact on network user experience. Reaction times, recovery times and detection times are the main performance metrics considered. Timing may be related to stability.

- For monitoring applications based on adaptive sampling, this time refers to the reaction time needed to readjust configuration of the monitors.
- A proof-of-concept has been setup to compare recovery times for normal OSPF and SRG inference enhanced OSPF. The timing behavior has been examined using packet traces, in order to determine the duration of interruption of the video streams. In the latter case, when locally detecting one of the failing links, all SRG links (and thus all streams) benefit from fast detection, allowing up to sub 25 ms recovery of the video streams in the demonstration setup.
- Detection time of path exploration events in intimately related to instability as demonstrated using RouteView BGP datasets.

Stability Stability can be expressed against a number of performance metrics.

- IDIPS is stable over transactions, as no drift is observed. As expected, the larger the number of destinations, the less stable the service time as suggested by the service time distribution amplitude.
- Off-line experiments have shown the stability of the inferred SRG sets in terms of prediction accuracy over time (i.e., total number of recorded failures).

Scalability Scalability can be expressed against a number of performance metrics. Also scalability is expressed against one or more scenario parameters (e.g., topology, input traffic).

- Adaptive sampling is scalable by definition as the system automatically adapts itself to the resource constraints. The experimented system is also scalable in terms of number of monitoring tasks and network topology.
- IDIPS experiments show that the time spent in the whole ranking process increases linearly with the number of paths from the requests. As expected, the capacity of IDIPS to process requests decreases with the request size. It is expected behavior as large requests require more processing time, in terms of IDIPS (typically more cost function to evaluate and, thus, more lookups into the predicted values storage) and internal XORP processing.
- For a network comprising n nodes, the size of the SRG table (in bytes) is $O(n^4)$. Generally however, only SRGs that have been observed will be stored in the SRG table.
- Run-time memory cost in BGP updates processing scalability in terms of memory usage, number of prefixes and attributes can be improved by means of BGP messages filtering.
- In the ALFA fast re-routing technique, the alternate FIB (aFIB) stored at each node is initially a copy of the primary FIB (pFIB) which by default doubles the size of the FIB. Several improvements can be applied: i) after configuration, the forwarding entries for which the primary and the alternate next-hop for the same destination are identical can be removed from the aFIB, ii) aFIB entries can be aggregated (at the expense of additional processing to perform aFIB updates).
- The clustering algorithm of the Unsupervised Analysis module performs multiple clusterings in $N = m(m-1)/2$ low-dimensional sub-spaces $X_i \subset X$. This multiple computation imposes scalability issues for on-line detection of attacks in very high-speed networks. Two key features of the algorithm are exploited to reduce scalability problems in number of features m and the number of aggregated flows n to analyze: i) clustering in very low dimensional sub-spaces, $X_i \in R^2$, which is faster than clustering in high-dimensional spaces, and ii) each sub-space can be clustered independently of the other sub-spaces, which is perfectly adapted for parallel computing architectures. Moreover, parallelization can be considered to improve NEWNADA does not use parallelization (each sub-space is sequentially analyzed, one after the other). Indeed, parallelization can significantly improve the computational time of NEWNADA, and therefore increases the volume of traffic that can be monitored in an on-line basis.

Overall, the experimentation of the learning modules by means of the machine learning engine has been successfully conducted by means of the ECODE Unified Architecture (EUA) proposed in deliverable D2.2. Indeed, the experimental results obtained show that such a platform can host different learning modules performing different learning tasks.

Appendix A

Path availability and coordinate system

In large-scale distributed systems a full-mesh active probing of end-to-end performance metrics is very costly. One way to make it more scalable consists in measuring a small set of pairs only, and infer the non measured ones. This approach has been used to infer Round Trip Times (RTTs) using coordinate systems, but it does not extend to other metrics like the Available Bandwidth (ABW). An alternative approach used in this deliverable consists in formulating this problem as a matrix completion problem, whereby a matrix representing all the pairwise performance values is only very partially known by actual measurements and all other entries have to be inferred. This problem turns out to be feasible because the matrix can be approximated by a low-rank matrix, thanks to correlations among all the measurements. Moreover, it can be solved in a fully distributed way without building the huge matrix, and without relying on special nodes such as landmarks or central calculation servers.

In Section A.1 a fully decentralized algorithm based on Stochastic Gradient Descent (SGD) is proposed to solve the matrix completion problem. By letting network nodes exchange messages with each other, the algorithm only requires each node to collect and process local measurements, with no need for explicit constructions of matrices, nor for any special nodes such as landmarks or central servers. In addition, we compared comprehensively matrix factorization and Euclidean embedding to demonstrate the suitability of the former on network distance prediction. Extensions by incorporating robust loss functions and the non-negativity constraints were also studied. We justify the simplicity, the flexibility and the superiority of our approach by extensive experiments on various publicly available datasets of network delays of both static measurements and dynamic measurements collected from a real application. This work has been submitted for publication in a journal. An earlier version appeared in [78].

This approach is also applicable when the performance values are not measured exactly, but are only known qualitatively as binary values, namely whether they are "good enough" or "too bad" wrt a threshold. This is particularly important for a metric like the ABW which is much more costly to be determined exactly than to be assigned to a binary class. The new matrix completion problem now requires to predict the class of the missing matrix entries knowing only the classes of the measured entries. We have extended our decentralized matrix factorization algorithm to solve this new problem and shown its performance on three (static and dynamic) datasets of RTTs and ABWs, and its robustness against erroneous measurements. We have also highlighted its usability on a peer selection application. This class-based variant of our approach is not described in this deliverable by lack of place, but the reader can refer to [77].

In Section A.2, we rely on this Network Coordinate System (NCS) to discover appropriate

routing shortcuts in the network, namely paths through intermediate nodes that turn out to have smaller delays than the direct paths. Indeed, the knowledge of estimated delays between nodes can be useful to select better paths for real-time applications (e.g. IDIPS). 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* (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 NCS to detect useful routing shortcuts in networks. An earlier version of this work appeared in [26].

Finally, in Section A.2.2 we explain how these algorithms (NCS and Shortcut detector) have been implemented and combined to operate as standard peer-to-peer applications. Moreover, the Vivaldi NCS was also implemented within the ECODE XORP architecture (see deliverable D3.5).

A.1 Network Distance Prediction by Decentralized Matrix Factorization

A.1.1 Introduction

On large networks such as the Internet, many applications require the knowledge of end-to-end network distances in order to achieve Quality of Service (QoS) objectives. In the networking community, the distance between two network nodes is defined as the delay between them, in the form of either one-way delay or more often round-trip time (RTT). Examples include peer-to-peer file sharing and content distribution systems where peers preferably access nodes or servers that are likely to respond fast [30, 114, 99, 100, 118, 33, 46].

Clearly, it is unfeasible to probe actively end-to-end distances among all pairs of nodes in large networks as the demanded measurements grow quadratically with the scale of the network. A natural idea is to probe a small set of pairs and then predict the distances between other pairs where there are no direct measurements. This understanding has motivated numerous research on Network Coordinate System (NCS) [89, 31, 83, 78, 36]. For instance, approaches based on Euclidean embedding have been widely studied and achieved good performance in interesting scenarios [31, 72]. Realizing that the assumption of Euclidean distance properties (symmetry and triangle inequality) are often violated in practice, as observed in various studies [119, 75, 115, 31, 14, 81], matrix factorization has recently drawn increasing attention of the networking community [83, 78].

In this section, we investigate matrix factorization for network distance prediction. In particular, we formulate the problem of network distance prediction as a matrix completion problem

where a partially observed matrix is to be completed [25, 24, 65]. Here, the matrix contains distance measurements such as RTTs between network nodes with some of them known and the others unknown thus to be filled. Matrix completion is only possible if matrix entries are largely correlated, which certainly holds for network distances because Internet paths with nearby end nodes often overlap and share common bottleneck links. These redundancies among network paths cause the constructed distance matrix to be low rank, which will be empirically demonstrated for various RTT datasets.

The low-rank nature of network distance matrices enables their completion by many matrix factorization techniques. In this section, we propose a novel approach based on Stochastic Gradient Descent (SGD) which has two distinct features. First, it is fully decentralized with no requirement of explicit constructions of matrices and of special nodes such as landmarks and central servers where measurements are collected and processed. Instead, by letting network nodes exchange messages with each other, matrix factorization is collaboratively and iteratively achieved at all nodes, with each node equally retrieving a number of distance measurements. Second, the algorithm is simple, with no infrastructure, and is computationally lightweight, containing only vector operations. These features make it suitable for dealing with practical problems, when deployed in real applications, such as measurement dynamics where network measurements vary largely over time and network churn where nodes join and leave a network frequently. Extensive experiments on various publicly-available RTT datasets show not only the scalability and the accuracy of our approach but also the usability by real Internet applications.

A short version of this section was published in [78]. Here, we make the following distinct contributions:

- Our previous algorithm in [78] was based on Alternating Least Squares (ALS), requiring each node to probe all local measurements simultaneously. In contrast, the new SGD-based algorithm allows each node to probe one measurement at a time, making the system more flexible. The new algorithm also addresses practical issues including the sensitivity to an important parameter of SGD, the learning rate, and the passive acquisition and the dynamics of the measurements.

- We compare comprehensively matrix factorization and Euclidean embedding to reveal the suitability of matrix factorization. A unified view is provided which leads to a unified optimization framework to solve both of them.

- Two extensions to the current matrix factorization model are proposed, including the incorporation of a robust loss function and a non-negativity constraint to preserve the non-negativity of the distances. These extensions are found helpful in improving the accuracy of the prediction and require little modification to the algorithm with no additional computational cost.

- In addition, more extensive evaluations have been carried out to study not only the impacts of the parameters but also the accuracy of our approach. In particular, we highlight the usability of our approach by evaluations using real data containing dynamic measurements collected from a real Internet application, Azureus [114, 72].

The rest of the section is organized as follows. Section A.1.2 summarizes the related work on network distance prediction based on Euclidean embedding and matrix factorization. Section A.1.3 introduces the formulation of network performance prediction as matrix completion

and its resolution by low-rank matrix factorization. Section A.1.4 describes the decentralized matrix factorization algorithm based on Stochastic Gradient Descent. Conclusions and future work are given in Section A.1.6.

A.1.2 Related Work

Among numerous work on network distance prediction, we only discuss and compare approaches based on Euclidean embedding and on matrix factorization due to their simplicity and generality. We refer the interested readers to [36] for a more detailed review of this field.

Euclidean Embedding

A straightforward approach to network distance prediction is to embed network nodes into a metric space where each node is assigned a coordinate from which distances can be directly computed. Two representatives are Globe Network Positioning (GNP) [89] and Vivaldi [31].

GNP firstly proposed the idea of network embedding that relies on a small number of landmarks. Based on inter-landmark distance measurements, the landmarks are first embedded into a metric space such as Euclidean or spherical coordinate systems. Then, the ordinary nodes calculate their coordinates with respect to the landmarks. Vivaldi extended GNP in a decentralized manner by eliminating the landmarks. It simulates the network by a physical system of spring and minimizes its energy according to the Hooke's law to find an optimal embedding.

In all metric spaces, distances undergo two important properties:

- Symmetry: $\mathrm{d}(A,B) = \mathrm{d}(B,A)$;
- Triangle Inequality: $\mathrm{d}(A,B) + \mathrm{d}(B,C) \geqslant \mathrm{d}(A,C)$.

However, network distances are not necessarily symmetric especially when represented by one-way delays [92, 52]. The bigger issue is the property of triangle inequality. Many studies have shown that the violations of triangle inequality (TIV) are widespread and persistent in current Internet [119, 75, 115, 31, 14, 81]. In the presence of TIVs, metric space embedding shrinks the long edges and stretches the short ones, degrading heavily the accuracy of the embedding. Figure A.1 illustrates the idea of Euclidean embedding for network distance prediction and the impact of TIVs on the accuracy.

Without loss of generality, we focus on the simplest metric space, namely Euclidean coordinate systems, in the rest of this section. Other metric spaces are in principle the same.

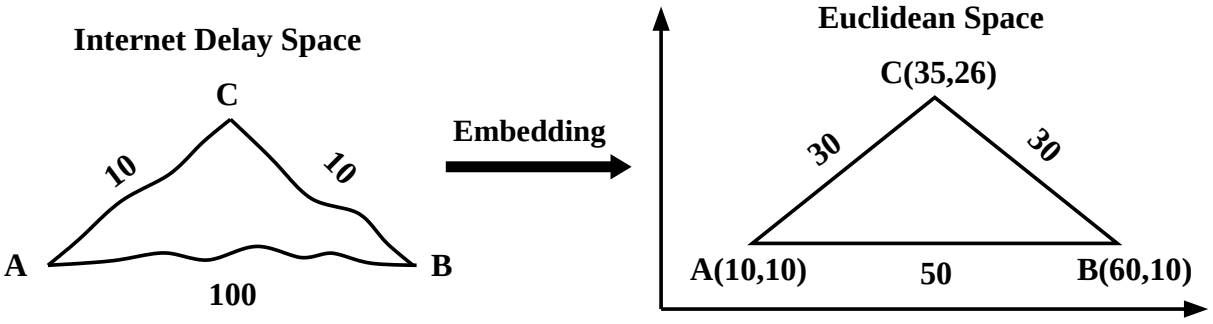


Figure A.1: Euclidean Embedding.

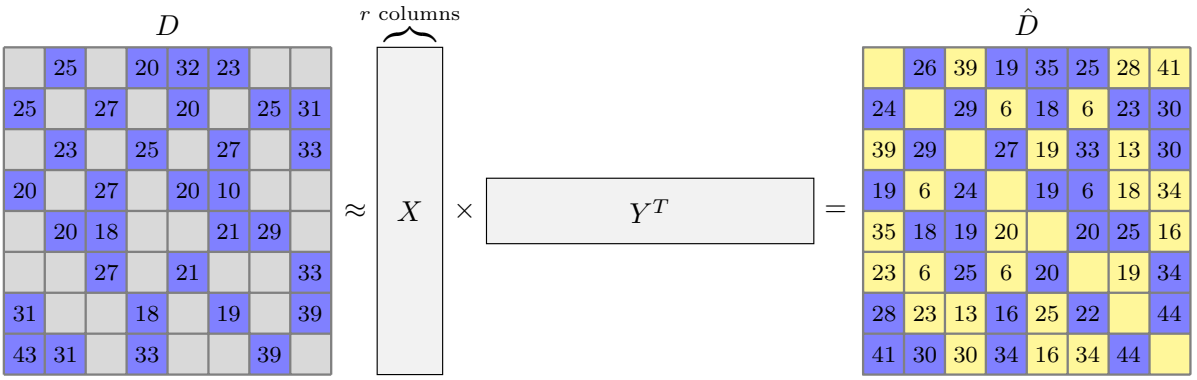


Figure A.2: Matrix Factorization. Note that the diagonal entries of D and $\hat{D}$ are empty.

Matrix Factorization

Alternatively, matrix factorization has also been used for network distance prediction, illustrated in Figure A.2. The biggest advantage of matrix factorization is that it makes no assumption of Euclidean distance properties and thus can tolerate the widespread TIVs and the possible asymmetry in network distance spaces.

The first system based on matrix factorization was Internet Distance Estimation Service (IDES) [83] which has the same landmark-based architecture as GNP. The difference is that IDES factorizes a small but full inter-landmark distance matrix, at a so-called information server, by using Singular Value Decomposition (SVD). Similarly, Phoenix treated the early-entered nodes as landmarks and allowed an ordinary node to select any existing nodes in the system which already have coordinates assigned [27]. These landmark-based systems suffers from common drawbacks including single-point failures, landmark overloads and potential security problems. The selection of landmarks can also affect the accuracy of the prediction. Moreover, in Section A.1.3, we will show that these landmark-based approaches are actually special cases of a general decentralized matrix factorization model and thus can also be solved by our approach.

Our main contribution is the formulation of the distance prediction problem as matrix completion. Although numerous approaches to matrix completion have been proposed, many of which are based on low-rank matrix factorization [87, 23, 105], few are applicable to network applications where decentralized processing of data is appreciated. Thus, we developed a fully decentralized algorithm based on Stochastic Gradient Descent (SGD) which is founded on the stochastic optimization theory with nice convergence guarantees [16]. We then studied empirically the sensitivity of the algorithms to the parameters and compared it with state-of-the-art approaches to demonstrate its accuracy.

A.1.3 Network Distance Prediction by Matrix Factorization

This section formulates the problem of network distance prediction as matrix completion and describes its resolution by matrix factorization. We also provide a unified view of different approaches to network distance prediction, the insights of which lead to a unified optimization framework.

Problem Formulation

Assuming n nodes in the network, a $n \times n$ distance matrix is constructed with some distances between nodes measured and the others unmeasured. Denote D the measured distance matrix with d_{ij} the measured distance from node i to node j and $\hat{D}$ the predicted distance matrix with $\hat{d}_{ij}$ the predicted distance computed from some function.

Given the above notations, network distance prediction can be viewed as a matrix completion problem that estimates the missing entries in D from a small number of known entries [24]. Its resolution generally amounts to minimizing a loss function of the following form

$$L(D,\hat{D},W) = \sum_{i,j=1}^{n} w_{ij} l(d_{ij},\hat{d}_{ij}), \tag{A.1}$$

where W is a weight matrix with w_{ij} taking values between 0 and 1. In a simple case, $w_{ij} = 1$ if d_{ij} is measured and 0 otherwise. l is a loss function that penalizes the difference between an estimate and its desired or true value. The most commonly-used loss function is the L_2 or square loss function,

$$l(d,\hat{d}) = (d-\hat{d})^2. \tag{A.2}$$

We will discuss other loss functions in Section A.1.5.

Low-Rank Approximation and Matrix Factorization

Additional constraints are needed to solve the matrix completion problem in Eq. A.1. A common approach is to constrain the rank of the approximate matrix $\hat{D}$ so that

$$Rank(\hat{D}) = r, \tag{A.3}$$

where $r \ll n$ for D of size $n \times n$

The assumption in this low-rank approximation is that the entries of D are largely correlated, which causes D to have a low effective rank. To show that it holds for our problem, Figure A.3 plots the singular values of two RTT matrices. It can be seen that the singular values of both matrices decrease fast as the 10th singular values are 5.7% and 2.9% of the largest ones respectively, indicating strong correlations in them. The low-rank nature of many other RTT datasets have been previously reported in [112].

Thus, to find $\hat{D}$, we need to minimize Eq. A.1 subject to Eq. A.3, which is considerably difficult due to the rank constraint. However, as $\hat{D}$ is of low rank, we can factorize it into the product of two smaller matrices, i.e.,

$$\hat{D} = XY^T, \tag{A.4}$$

where X and Y are of size $n \times r$. Therefore, we can get rid of the rank constraint by replacing $\hat{D}$ by XY^T in Eq. A.1, and then look for X and Y instead by minimizing

$$L(D,X,Y,W) = \sum_{i,j=1}^{n} w_{ij} l(d_{ij},x_i y_j^T), \tag{A.5}$$

where x_i is the ith row of X, y_i is the ith row of Y, and $x_i y_j^T = \hat{d}_{ij}$ is the estimate of d_{ij}. Note that the factorization in Eq. A.4 has no unique solution as

$$\hat{D} = XY^T = XGG^{-1}Y^T, \tag{A.6}$$

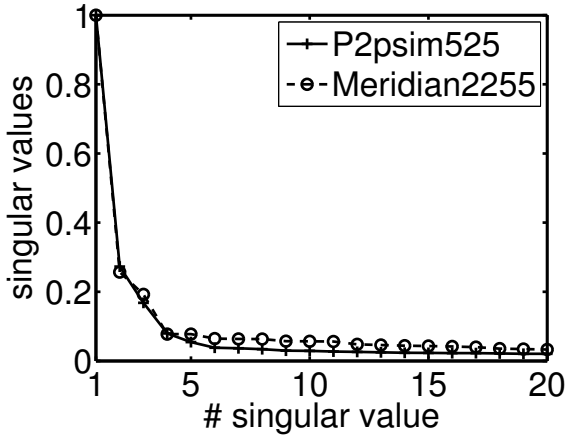


Figure A.3: The singular values of a RTT matrix of 2255×2255, extracted from the Meridian dataset [117] and called "Meridian2255", and of a RTT matrix of 525×525, extracted from the P2psim dataset [9] and called "P2psim525". The singular values are normalized so that the largest singular values of both matrices are equal to 1.

where G is any arbitrary $r \times r$ invertible matrix. Thus, replacing X by XG and Y^T by $G^{-1}Y^T$ results in the same $\hat{D}$.

Generally, the class of techniques to solve the low-rank approximation is matrix factorization. When D is complete, analytic solutions can be found by using singular value decomposition (SVD) [47]. With missing entries, the factorization can only be done by iterative optimization methods such as Gradient Descent and Newton algorithms [23]. Note that additional constraints can be imposed in Eq. A.5. For instance, the entries of X and Y can be required to be non-negative in order to recover a non-negative matrix, leading to the non-negative matrix factorization (NMF) [73].

Incorporation of the Regularization

Matrix completion by matrix factorization suffers from a well-known problem called overfitting in the field of machine learning [111]. In words, directly optimizing Eq. A.5 often leads to a "perfect" model with no or small errors on the training data while having large errors on the unseen data which are not used in learning. The problem is more severe when D is sparse or when r is large.

A common way to avoid overfitting is through regularization that penalizes the norms of the solutions, resulting in the following regularized loss function,

$$L(D,X,Y,W,\lambda) = \sum_{i,j=1}^{n} w_{ij} l(d_{ij}, x_i y_j^T) + \lambda \sum_{i=1}^{n} x_i x_i^T + \lambda \sum_{i=1}^{n} y_i y_i^T, \tag{A.7}$$

where λ is the regularization coefficient that controls the extent of regularization.

Besides avoiding overfitting, the regularization also helps overcome the drift of the solutions due to the non-uniqueness in Eq. A.6, which often leads to the overflows of the solutions. Among the infinite number of pairs of X and Y which produce the same $\hat{D}$, the incorporation of the regularization will force to choose the pair with the smallest norm.

A Unified View of Approaches to Network Distance Prediction

Although popular approaches to network distance prediction vary by adopting various models including Euclidean embedding and matrix factorization and by adopting different architectures of either landmark-based or landmark-less and thus decentralized, these seemingly different approaches all optimize the same function in Eq. A.1 but differ only in the setting of w_{ij} and in the associated distance functions to calculate $\hat{d}_{ij}$.

- Setting of w_{ij}: For landmark-based methods, as all paths between landmarks are measured and ordinary nodes probe only the landmarks,

$$w_{ij} = \begin{cases} 1 & \text{if node } j \text{ is a landmark} \\ 0 & \text{otherwise} \end{cases}.$$

  For decentralized methods, as each node equally probes a number of nodes,

$$w_{ij} = \begin{cases} 1 & \text{if node } i \text{ probes node } j \\ 0 & \text{otherwise} \end{cases}.$$

  Figure A.4 illustrates the architectures of landmark-based and decentralized systems.

- Distance functions to calculate $\hat{d}_{ij}$: For matrix factorization, as described above,

$$\hat{d}_{ij} = x_i y_j^T, \tag{A.8}$$

  For Euclidean embedding, the Euclidean distance is defined as

$$\hat{d}_{ij} = \sqrt{(x_i - x_j)^T (x_i - x_j)}, \tag{A.9}$$

  where x_i and x_j are the Euclidean coordinates of node i and node j.

The above insights suggest a unified framework to treat and to solve equally network distance prediction under different models and different architectures. For instance, the decentralized matrix factorization algorithms proposed in the following sections can be used to solve both Euclidean embedding and landmark-based systems with little modification.

A.1.4 Decentralized Matrix Factorization for Network Distance Prediction

The goal is to find X and Y such that XY^T best approximates D by minimizing Eq. A.7. Commonly, it requires to collect and to process a number of distance measurements at a central node, which is a major obstacle to network applications. Below, we introduce algorithms to minimize Eq. A.7 in a fully decentralized manner.

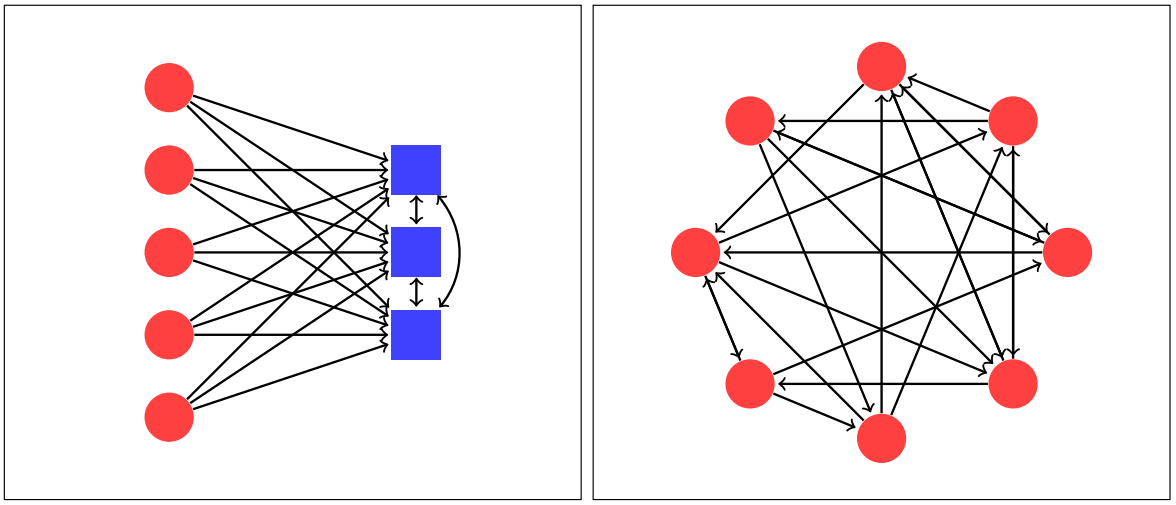


Figure A.4: Architectures of landmark-based, the left plot, and decentralized, the right plot, systems for network distance prediction. The squares are landmarks and the circles are ordinary nodes. The directed path from node i to node j means that node i probes node j and therefore $w_{ij} = 1$.

Problem Formulation

A decentralized resolution of Eq. A.7 forbids the explicit constructions of matrices. Thus, each row of X and of Y, denoted by x_i and y_i and called the x and y coordinates of node i in the sequel, are stored distributively at each node. To calculate x_i and y_i locally, Eq. A.7 is decomposed into a number of sub-problems, defined as

$$l_i = \sum_{j=1}^{n} w_{ij} l(d_{ij}, x_i y_j^T) + \lambda x_i x_i^T, \tag{A.10}$$

$$l^i = \sum_{j=1}^{n} w_{ji} l(d_{ji}, x_j y_i^T) + \lambda y_i y_i^T. \tag{A.11}$$

Essentially, l_i is the regularized loss of the edges from node i to other nodes and l^i is that of the edges from other nodes to node i. Clearly, optimizing l_i and l^i requires only distance measurements related to node i. Note that if the distance measurements are RTTs, then $d_{ji} = d_{ij}$ as RTTs are approximately symmetric. Consequently, $w_{ji} = w_{ij}$ as d_{ji} and d_{ij} are either both known or both unknown.

Eqs. A.10 and A.11 provide a natural decomposition of the large-scale optimization problem in Eq. A.7, which enables a decentralized resolution. In seeing that Eq. A.10 and Eq. A.11 are standard least-squares problems where analytical solutions exist, our previous work in [78] solved the matrix factorization problem by Alternating Least Squares (ALS), which alternatively and iteratively solves a number of least-squares sub-problems in the forms of Eqs. A.10 and A.11.

While the ALS-based algorithm performed well in simulations on datasets containing static measurements, it requires each node to probe measurements with a number of nodes simultaneously, which is impractical when deployed in real applications. Below, we propose a different

algorithm based on Stochastic Gradient Descent (SGD) that processes, at each node, measurements one by one and one at a time.

Stochastic Gradient Descent (SGD)

SGD is a variation of traditional Batch Gradient Descent which is often used for online machine learning [16]. Instead of collecting all training samples beforehand and computing the gradients over them, each iteration of SGD chooses one training sample at random and updates the parameters being estimated along the negative gradients computed over that chosen sample. SGD is particularly suitable for network applications, as measurements can be acquired on demand and processed locally at each node. It also has simple update rules that involve only vector operations and is able to deal with large-scale dynamic measurements.

Stochastic Updates In using SGD, at each time, each node probes one node in the system and retrieves the distance measurements between them and the coordinates of that node. Let node j be the chosen node by node i at the current time. Then, the regularized losses that node i seeks to reduce with respect to node j are

$$l_{ij} = l(d_{ij}, x_i y_j^T) + \lambda x_i {x_i}^T, \tag{A.12}$$
$$l_{ji} = l(d_{ji}, x_j y_i^T) + \lambda y_i {y_i}^T. \tag{A.13}$$

The gradients of l_{ij} and l_{ji} are

$$\frac{\partial l_{ij}}{\partial x_i} = \frac{\partial l(d_{ij}, x_i y_j^T)}{\partial x_i} + \lambda x_i, \tag{A.14}$$
$$\frac{\partial l_{ji}}{\partial y_i} = \frac{\partial l(d_{ji}, x_j y_i^T)}{\partial y_i} + \lambda y_i. \tag{A.15}$$

In particular, the gradients of the L_2 loss function are

$$\frac{\partial l}{\partial x_i} = -(d_{ij} - x_i y_j^T) y_j, \tag{A.16}$$
$$\frac{\partial l}{\partial y_i} = -(d_{ji} - x_j y_i^T) x_j. \tag{A.17}$$

Note that we drop the factor of 2 in the gradients for mathematical convenience.

Then, node i updates its coordinates along the negative gradient directions, given by

$$x_i = (1 - \eta\lambda) x_i + \eta (d_{ij} - x_i y_j^T) y_j, \tag{A.18}$$
$$y_i = (1 - \eta\lambda) y_i + \eta (d_{ji} - x_j y_i^T) x_j, \tag{A.19}$$

where η, called *learning rate* or *step size*, controls the speed of the updates.

Minibatch and Line Search The SGD algorithm is sensitive to the learning rate η, where a too large η results in large steps of updates and may overflow the solution, whereas a too small

Algorithm 2 Line Search (for updating x_i)

1: compute l_i^0 by Eq. A.10;
2: initialize η by a large value;
3: **while** true **do**
4: compute x_i by Eq. A.20;
5: compute l_i by Eq. A.10;
6: **if** $l_i + \delta < l_i^0$ **then**
7: **return**
8: **end if**
9: $\eta \longleftarrow \eta/2$;
10: **end while**

η makes the convergence slow. This sensitivity can be relieved by using more training samples at the same time, leading to minibatch SGD with the following update rules

$$x_i = (1 - \eta\lambda)x_i + \eta \sum_{j=1}^{n} w_{ij}(d_{ij} - x_i y_j^T)y_j, \tag{A.20}$$

$$y_i = (1 - \eta\lambda)y_i + \eta \sum_{j=1}^{n} w_{ji}(d_{ji} - x_j y_i^T)x_j. \tag{A.21}$$

To completely get rid of η, a line search strategy can be incorporated to determine η adaptively [15]. In particular, in each update, η starts with a large initial value and is gradually decreased until the losses in Eq. A.10 or Eq. A.11 are reduced after the update. The line search algorithm for updating x_i is given in Algorithm 2. The same algorithm can be used for updating y_i by replacing Eq. A.10 by Eq. A.11 and Eq. A.20 by Eq. A.21. Note that δ in Line 6 is a constant of some small positive value to ensure that the algorithm stops. It is also effective to adapt η by line search.

Neighbor Decay and Neighbor Selection

As mentioned earlier, it is preferable to have a system that probes and processes measurements one by one. Thus, we let each node maintain the information (distance measurements and coordinates) of the nodes with which it communicates, called neighbors in the sequel. In minibatch SGD, each node probes one neighbor at a time but updates its coordinates with respect to all neighbors in the neighbor set using their recorded historical information.

A neighbor decay strategy is incorporated that scales the weight of each node in the neighbor set by its age so that older information receives less weight, i.e.,

$$w_{ij} = \frac{a_{max} - a_j}{\sum_{j \in \text{NeighborSet}(i)}(a_{max} - a_j)}, \tag{A.22}$$

where a_j is the age of the information of node j and a_{max} is the age of the oldest information in the neighbor set. Note that this neighbor decay strategy was firstly proposed by [72] to overcome the problem of skewed neighbor update in Vivaldi. In words, some nodes may be probed at far greater frequency than others due simply to their longer life cycles and a direct consequence is that the optimization will become skewed toward these nodes.

Algorithm 3 DMFSGD(i, j)

1: node i retrieves d_{ij}, d_{ji}, x_j, y_j actively or passively;
2: node i updates the weights of its neighbors by Eq. A.22;
3: update x_i by Eq. A.20 with η set by line search;
4: update y_i by Eq. A.21 with η set by line search;

Conventionally, the neighbors of a node are selected randomly and the distances between a node and its neighbors are probed by active measurements [32]. However, in practice, it is more attractive to perform the updates of the coordinates passively without generating any extra traffic. In some applications such as Azureus, the passivity is forced, as we have no control over the selection of neighbors with which a node communicates and when it communicates with them [72].

Therefore, we differentiate the situations where distances are probed by active and passive measurements. For the former, the conventional random neighbor selection procedure is adopted that each node randomly selects k nodes as its neighbors and actively probes one of them from time to time. For the latter, no neighbor selection is performed and each node maintains a relatively small set of active neighbors with which it recently communicates and updates its coordinates whenever a measurement is made available. Note that this difference has no impact to the update rules in eqs A.18 and A.19 or in eqs A.20 and A.21.

Algorithm

We denote the SGD-based decentralized matrix factorization algorithm as DMFSGD, given in Algorithm 3. Like Vivaldi [32], our DMFSGD algorithm has no infrastructure and employs the same process at all nodes. It is simple, with update rules containing only vector operations.

In the implementation, the coordinates of each node are initialized with random numbers uniformly distributed between 0 and 1. Empirically, the algorithm is insensitive to the random initialization of the coordinates. We would like to point out that the algorithm is one of those randomized gossip algorithms where each node exchanges messages with a number of other nodes randomly [17].

As mentioned earlier, the algorithm is generic and can also deal with landmark-based architectures, by letting each node only select landmarks as its neighbors, and with Euclidean embedding, by adopting the Euclidean distance defined as Eq. A.9 when optimizing Eq. A.1, which leads to the update rule of Vivaldi [32], given by

$$x_i = x_i - \eta \frac{\partial l(d_{ij}, \hat{d}_{ij})}{\partial x_i} = x_i + \eta (d_{ij} - \hat{d}_{ij}) \frac{x_i - x_j}{\hat{d}_{ij}}.$$

Note that Vivaldi adopted the L_2 loss function in Eq. A.1 with no regularization incorporated, and the learning rate η, termed differently as *timestep*, was adapted by taking into account some confidence measure of each node to its coordinate. Thus, Vivaldi can be viewed as a SGD-based decentralized Euclidean embedding algorithm, instead of the simulation of a spring system in [32].

A.1.5 Extended Matrix Factorization Models

This section discusses possible ways to extend the common matrix factorization model.

Robust Matrix Factorization

The widely-used L_2 loss function is known to be sensitive to outliers which often occur in network measurements due to network anomaly such as sudden traffic bursts and attacks from malicious nodes. Other loss functions such as the L_1 loss function, the ε-insensitive loss function and the Huber loss function are more robust and can tolerate outliers [53, 63]. For example, the L_1 loss function is defined as

$$l(d,\hat{d}) = |d-\hat{d}|. \tag{A.23}$$

Thus, we can enhance the robustness of matrix factorization by replacing the L_2 loss function by e.g. the L_1 loss function, and the same SGD procedure can be applied to solve the robust matrix factorization problem. Note that the L_1 loss function is non-differentiable and the gradients have to be approximated by the subgradients [1], given by

$$\frac{\partial l}{\partial x_i} = -\text{sign}(d_{ij} - x_i y_j^T) y_j, \tag{A.24}$$

$$\frac{\partial l}{\partial y_i} = -\text{sign}(d_{ji} - x_j y_i^T) x_j. \tag{A.25}$$

Replacing the gradient functions of Eq. A.14 and Eq. A.15 by Eq. A.24 and Eq. A.25, the update rules of minibatch SGD become

$$x_i = (1-\eta\lambda)x_i + \eta \sum_{j=1}^{n} \text{sign}(d_{ij} - x_i y_j^T) w_{ij} y_j, \tag{A.26}$$

$$y_i = (1-\eta\lambda)y_i + \eta \sum_{j=1}^{n} \text{sign}(d_{ji} - x_j y_i^T) w_{ji} x_j, \tag{A.27}$$

Comparing Eq. A.26 and Eq. A.27 with Eqs. A.20 and Eq. A.21, the only difference is that for the L_2 loss function, the magnitudes of the updates are proportional to the fitting errors $(d - xy^T)$, whereas for the L_1 loss function, only the signs of the fitting errors are taken into consideration and decide the directions of the updates.

Non-Negativity Constraint

Conventional matrix factorization techniques do not preserve the non-negativity of the distances. Empirically, only a very small portion of the predicted distances were found negative by our DMFSGD algorithm, and a direct solution is to turn $\hat{d}_{ij}$ into a small positive value if $\hat{d}_{ij} = x_i y_j^T < 0$.

A systematic solution is to incorporate the non-negativity constraint in matrix factorization, leading to the non-negative matrix factorization (NMF) that optimizes

$$\sum_{i,j=1}^{n} w_{ij} l(d_{ij}, x_i y_j^T) + \lambda \sum_{i=1}^{n} x_i x_i^T + \lambda \sum_{i=1}^{n} y_i y_i^T, \tag{A.28}$$

$$\text{subject to } x_i \geqslant 0,\ y_i \geqslant 0,\ i = 1,\ldots,n.$$

[1] Analogously, the subgradient-based technique that optimizes non-differentiable functions is called subgradient descent [15]. Following the convention in [16], we use the term SGD to refer to both Stochastic Gradient and SubGradient Descent.

The optimization of NMF is not fundamentally different from that of the unconstrained matrix factorization, adding only one projection step that turns the negative entries in x_i and y_i into zero after each SGD update which causes no noticeable impact on the speed of the algorithm. The technique is also known as projected gradient descent [80].

Note that the non-negativity constraint has been previously studied in [83, 27], both of which adopted a more heavyweight non-negative least-squares solver.

Symmetric Distance Matrix Factorization

Also note that network distances are symmetric if represented by RTT and that this symmetry is not preserved either. A direct solution is to turn the predicted distances symmetric by defining a symmetric distance function as

$$\hat{d}^s_{ij} = \frac{\hat{d}_{ij} + \hat{d}_{ji}}{2} = \frac{x_i y_j^T + x_j y_i^T}{2}. \tag{A.29}$$

As distances are defined as in Eq. A.29, a systematic solution is to factorize D by optimizing

$$\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} l(d_{ij}, \hat{d}^s_{ij}) + \lambda \sum_{i=1}^{n} x_i x_i^T + \lambda \sum_{i=1}^{n} y_i y_i^T. \tag{A.30}$$

Similar SGD update rules can be derived.

Height Model

The height model in Vivaldi [32] can also be incorporated that augments the x and y coordinates of a node with a height. Similarly, the x and y coordinates model the high-speed Internet core, while the height models the time packets take to travel the access link from the node to the Internet core. The cause of the access link distance includes queuing delay and low bandwidth [32]. The height augmented symmetric distance is defined as

$$\hat{d}^{hs}_{ij} = \frac{x_i y_j^T + x_j y_i^T}{2} + h_i + h_j. \tag{A.31}$$

Correspondingly, the loss function to be optimized becomes

$$\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} l(d_{ij}, \hat{d}^{hs}_{ij}) + \lambda \sum_{i=1}^{n} x_i x_i^T + \lambda \sum_{i=1}^{n} y_i y_i^T. \tag{A.32}$$

Similar SGD update rules can be derived.

Extended DMFSGD Algorithm

Empirically, we found no or little improvements by incorporating the symmetric or height-augmented symmetric distance function in Eq. A.29 or A.31, thus include neither of them in our system. However, the incorporation of the non-negativity constraint and the robust loss function not only improved the accuracy but also made the results more stable and less sensitive to parameter settings. The extended DMFSGD algorithm is given in Algorithm 4. Note that as Algorithm 3 is a special case of Algorithm 4, we will refer to the extended version in Algorithm 4 simply as DMFSGD in the sequel.

Algorithm 4 Extended_DMFSGD(i, j)

node i retrieves d_{ij}, d_{ji}, x_j, y_j actively or passively;
node i updates the weights of its neighbors by Eq. A.22;
if use L_2 loss function **then**
    update x_i by Eq. A.20 with η set by line search;
    update y_i by Eq. A.21 with η set by line search;
else {use L_1 loss function}
    update x_i by Eq. A.26 with η set by line search;
    update y_i by Eq. A.27 with η set by line search;
end if
if force non-negativity **then**
    turn the negative entries in x_i and y_i into 0;
end if

A.1.6 Conclusions

We have presented a novel approach to network distance prediction. The success of the approach roots both in the exploitation of the dependencies across distance measurements between network nodes and in the stochastic optimization which enables a fully decentralized architecture. A so-called Decentralized Matrix Factorization based on Stochastic Gradient Descent (DMFSGD) algorithm is proposed to solve the distance prediction problem. The algorithm is simple, with the same architecture as Vivaldi, scalable, able to deal with dynamic measurements in large-scale networks, and accurate, generally superior to Vivaldi. In particular, experiments on real data collected from Azureus demonstrates the potential of the algorithm being utilized by Internet applications, which we would like to study in the future.

A.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 deliverables, we have proposed some methods that rely on the nodes running an ICS to detect routing shortcuts in networks. In this section we essentially show that combining the methods proposed in the previous delivrable provides better detection results.

A.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. In such case, using C as a relay node to go from A to B instead of sending the data directly from A to B reduce the delay experienced between A and B and is called overlay routing. The second sub-problem we address then consists in finding C nodes that are routing shortcuts for a given path AB.

Related work

Overlay routing is attractive because deploying an overlay requires only the cooperation of the nodes participating in the overlay. It has already been proposed to use overlay routing to improve the performance (and the reliability) of a network: it has been observed in [103] that indirect routing can significantly improve the performance for many paths and, on the basis of these observations, Detour [102] and RON [11] were proposed. The idea of RON is to build a fully connected mesh between the nodes participating in the overlay and to monitor all the paths. If a direct path between two nodes is broken or if it has poor performance it proposes to use relay nodes to reach the destination.

Even if overlay networks seem to be an easy way to improve the performance of the Internet, they are not often used by applications. The main problem is the scalability. Indeed, to obtain the results proposed by RON, it is necessary to measure all the paths and distribute measurements results among the overlay nodes in order to apply a routing algorithm. These operations become costly if a large number of nodes are members of the overlay.

A first approach to improve the scalability of overlay networks consists in eliminating redundant overlay paths. For example, Nakao et al. [85, 86] proposed to do that by using topological information (AS-level topology, etc.).

Another solution consists in reducing the communication overhead generated by the exchanges of the measurements results between all the nodes. In RON this overhead is $O(n^2)$ where n is the number of nodes in the overlay. Sontag et al. [109] proposed a protocol allowing each node to find an optimal one-hop path to any other node with a communication overhead $O(n^{1.5})$.

The last way to improve the scalability of overlay routing is to reduce the measurement overhead. Since the objective is to change traffic routes, we suppose that measurements must be done quite frequently to have accurate information about the state of the network. To circumvent this problem Gummadi et al. [48] proposed to route through random relay nodes instead of doing measurements and they observed that it is sufficient to ensure reliability. However, Sontag et al. [109] observed that it is not sufficient to find good alternative paths considering particular metrics like latency. Recently, Lumezanu et al. [82] proposed to use an ICS to detect one-hop shortcuts in a network. Since these situations cannot be reproduced by the estimations provided by the ICS, their idea consists in using estimation errors to find paths that are edges of one-hop shortcuts. We have also explored that solution at the beginning of our work but we gave up because our observations showed that the impact of one-hop shortcuts on an ICS vary according to which shortcut's edges are measured by the ICS (to compute the estimations) and according to the existence of other shortcuts in the network. We have investigated other ways [62, 79] to detect one-hop shortcuts by observing the behavior of the ICS over time instead of computing the estimation errors at a given time. The results obtained are satisfactory but these detection methods are quite heavy to deploy: they need constant collection of data and the processing of these data has a cost. Since the detection of one-hop shortcuts by analyzing the behavior of an ICS seems difficult we propose to use the estimations provided by the ICS to estimate whether or not some node C is a shortcut for a given path AB.

Contribution

Like Lumezanu et al. [82], we propose to use an ICS (Vivaldi) in order to reduce the measurement overhead of an overlay routing mechanism. We know that 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

$$EST(A,B) > EST(A,C) + EST(C,B) \tag{A.33}$$

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.

A.2.2 Implementation

Without loss of generality we consider a classical ICS algorithm, Vivaldi [32], and we have proposed two basic shortcut detection criteria in the previous delivrable: EDC and ADC. In this section, we will describe a third criterion that combine the two basic ones in order to obtain better detection results.

Detection criteria definitions

Lumezanu et al. stated in [82] that, if a node C violates the triangle inequality with a path AB, $EST(A,B)$ is an under-estimation of $RTT(A,B)$. We observed this too: generally, more than 80% of the paths for which there exists at least one (significant) shortcut are under-estimated by the ICS. Our detection criteria are based on that observation. Indeed, if the estimation of the alternative path is reliable and if the path AB is significantly under-estimated, we will restore the TIV by replacing $EST(A,B)$ by $RTT(A,B)$ in (A.33).

Estimation Detection Criterion (EDC) is our first criterion. To decide if a node C is a shortcut for a path AB, this criterion compares the measured RTT of the direct path between A and B and the estimated RTT of the alternative path using C as relay. Formally, a node C is considered as a shortcut for the path AB if

$$RTT(A,B) > EST(A,C) + EST(C,B) \tag{A.34}$$

The potential problem with that criterion is that it uses the values of the estimations provided by the ICS as if there were no estimation error. However, we know that there are estimation errors and, in particular, that these errors cannot be avoided if node C is a shortcut for the path AB. So, using the exact values of the estimated RTTs to find shortcuts is not necessarily a good idea.

Approximation Detection Criterion (ADC) is our second criterion. For a path AB and a node C, we define C_A (resp. C_B) as C's nearest node among A's (resp. B's) Vivaldi neighbors 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,

$$RTT(A,B) > RTT(A,C_A) + RTT(C_B,B) \tag{A.35}$$

There is still a potential problem with ADC. Indeed, if it is impossible to find some A's (resp. B's) Vivaldi neighbors near C, the approximation of $RTT(A,C)$ (resp. $RTT(C,B)$) by $RTT(A,C_A)$ (resp. $RTT(C_B,B)$) can be very bad. In such case, using the EDC criterion can provide more reliable detection results even if there are estimation errors. So, we define a *Hybrid Detection Criterion (HDC)* by combining our two basic criteria in order to exploit their advantages. Formally, let C_A (resp. C_B) be C's nearest node among A's (resp. B's) Vivaldi neighbors according to the estimated RTTs. We define

$$VAL(A,C) = \begin{cases} RTT(A,C_A) & \text{if } EST(C_A,C) < threshold \\ EST(A,C) & \text{otherwise} \end{cases}$$

$$VAL(C,B) = \begin{cases} RTT(C_B,B) & \text{if } EST(C_B,C) < threshold \\ EST(C,B) & \text{otherwise} \end{cases}$$

where *threshold* is a value used to decide if C_A (resp. C_B) is sufficiently near C to obtain a quite good approximation of $RTT(A,C)$ (resp. $RTT(C,B)$) by using $RTT(A,C_A)$ (resp. $RTT(C_B,B)$). During our experiments, we observed that using a *threshold* equal to 10% of $RTT(A,B)$ when we search a shortcut for the path AB is a good choice. Using these definitions, a node C is considered as a shortcut for the path AB if

$$RTT(A,B) > VAL(A,C) + VAL(C,B) \tag{A.36}$$

Ranking of the detected C nodes

We have three criteria which, for a given path AB, are able to return a set of C nodes that are probably shortcuts for that path. 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, nodes 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

$$G_a = RTT(A,B) - (RTT(A,C) + RTT(C,B)) \qquad G_r = \frac{G_a}{RTT(A,B)} \tag{A.37}$$

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 highest value for these parameters. However, we cannot compute G_a and G_r for all C nodes. 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 on the shortcut detection criterion used to obtain the set of C nodes:

$$EG_a = RTT(A,B) - (V(A,C) + V(C,B)) \qquad EG_r = \frac{EG_a}{RTT(A,B)} \tag{A.38}$$

where $V(X,Y)$ is the value used by the detection criterion to estimate the RTT of the path XY.

For a path AB, we will rank the C nodes of the set selected by a shortcut detection criterion in decreasing order of their estimated gain. If the nodes with the highest estimated gains are also those with the highest (real) gains then we will find the nodes providing the most interesting shortcuts in the top of the ranking.

A.2.3 Conclusion

We showed that, for any given path AB, using only the RTT of that path and the information available in an ICS, it is possible to select a small set of nodes containing very likely an interesting one-hop shortcut (but not necessarily the best one) when shortcuts exist for that path. We obtained the best results with our shortcut detection criterion called HDC. With that criterion we are able to limit the number of potential shortcuts for any given path AB to about one or two percent of the total number of nodes in the network. So, to improve significantly the latency between A and B, we will only have to do measurements between A, B and these few candidate nodes to know if they are really shortcuts and which of them is the best shortcut.

Our final goal consists in designing a distributed self-organized one-hop routing mechanism based on the observations reported so far. Compared to a solution like RON [11] the traffic generated by the measurements will be significantly reduced with our approach. With RON, since each node has to do RTT measurements with all the other nodes this traffic is $O(n^2)$. With our approach, each node measures only with its m neighbors and this traffic is $O(n \times m)$ (with $m \ll n$). Our approach will also reduce the communication overhead generated by the exchanges of the measurements results. With RON, each node has to send its measurement results to each other nodes. So, the number of messages is $O(n^2)$ and the size of each message is $O(n)$. Consequently, the traffic generated is $O(n^3)$. With our approach this traffic is only $O(n^2)$: each node will have to send a message to each other nodes (n^2 messages) but each message contains only one coordinate vector. Finally, even if our approach is less efficient than RON in terms of quality of the shortcuts proposed (RON provides the guarantee to find the best shortcut for any path and we don't), our approach will generate a lot less traffic in the network.

Bibliography

[1] Geant.

[2] JumpGen Network-Processor Cards. http://www.jumpgen.com.

[3] *matlab mdscale function*. http://www.mathworks.de/access/helpdesk/help/toolbox/stats/mdscale.html.

[4] Mawi working group traffic archive.

[5] METROlogy for SECurity and QoS. http://laas.fr/METROSEC.

[6] TCPReplay, Pcap Editing and Replay Tools for *NIX. http://tcpreplay.synfin.net/.

[7] The KDD Cup 1999 Dataset. http://kdd.ics.uci.edu/databases/kddcup99.

[8] Softflowd flow-based network traffic analyser, 2010.

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

[10] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.

[11] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. *SIGOPS Oper. Syst. Rev.*, 35(5):131–145, 2001.

[12] A. Atlas et al. U-turn alternates for ip/ldp fast-reroute. *IETF Draft, Feb*, 2006.

[13] A. Atlas and A. Zinin. Basic Specification for IP Fast Reroute: Loop-Free Alternates. RFC 5286 (Proposed Standard), September 2008.

[14] Suman Banerjee, Timothy G. Griffin, and Marcelo Pias. The interdomain connectivity of PlanetLab nodes. In *Proc. of the Passive and Active Measurement*, Antibes Juan-les-Pins, France, April 2004.

[15] D.P. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[16] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, 1998.

[17] Stephen Boyd, Arpita Ghosh, Student Member, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52:2508–2530, 2006.

[18] D Brauckhoff, K Salamatian, and M May. A signal processing view on packet sampling and anomaly detection. *IEEE Infocom*, Mar 2010.

[19] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 93–104, New York, NY, USA, 2000. ACM.

[20] B. Briscoe. Tunnelling of Explicit Congestion Notification. RFC 6040 (Proposed Standard), November 2010.

[21] S. Bryant, C. Filsfils, S. Previdi, and M. Shand. Ip fast reroute using tunnels. *Work in Progress in IETF*, 2005.

[22] S. Bryant, M. Shand, and S. Previdi. Ip fast reroute using not-via addresses. *draft-bryant-shand-ipfrr-notvia-addresses-03. txt*, 2006.

[23] A. M. Buchanan and A. W. Fitzgibbon. Damped newton algorithms for matrix factorization with missing data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 316–322, 2005.

[24] E. J. CandÃ¨s and Y. Plan. Matrix completion with noise. *Proceedings of the IEEE*, 98(6), 2010.

[25] E. J. CandÃ¨s and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

[26] F. Cantin and G. Leduc. Finding routing shortcuts using an internet coordinate system. In *Proc. of IWSoS 2011*, Karlsruhe, Germany, Feb. 2011.

[27] 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.

[28] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1 – 14, 1989.

[29] K. Cho, K. Mitsuya, and A. Kato. Traffic data repository at the wide project. In *Proc. of USENIX 2000 Annual Technical Conference: FREENIX Track*, pages 263–270, 2000.

[30] Mark Crovella and Balachander Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications*. John Wiley & Sons, Inc., New York, NY, USA, 2006.

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

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

[33] Frank Dabek. *A Distributed Hash Table*. PhD thesis, Massachusetts Institute of Technology, November 2005.

[34] Gustavo de Veciana, Takis Konstantopoulos, and Tae-Jin Lee. Stability and performance analysis of networks supporting elastic services. *IEEE/ACM Trans. Netw.*, 9:2–14, February 2001.

[35] G. Dewaele, K. Fukuda, P. Borgnat, P. Abry, and K. Cho. Extracting Hidden Anomalies using Sketch and non Gaussian Multi-resolution Statistical Detection Procedures. In *Proc. of ACM Workshop on Large-Scale Attack Defense*, 2007.

[36] Benoit Donnet, Bamba Gueye, and Mohamed Ali Kaafar. A survey on network coordinates systems, design, and security. *IEEE Communication Surveys and Tutorial*.

[37] Ibtissam El Khayat, Pierre Geurts, and Guy Leduc. Enhancement of tcp over wired/wireless networks with packet loss classifiers inferred by supervised learning. *Wirel. Netw.*, 16:273–290, February 2010.

[38] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. *Applications of Data Mining in Computer Security*, 2002.

[39] Martin Ester, Hans-Peter Kriegel, J Sander, and X Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Editors, editor, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, volume 96, pages 226–231. AAAI Press, 1996.

[40] Wu-chang Feng, Kang G. Shin, Dilip D. Kandlur, and Debanjan Saha. The blue active queue management algorithms. *IEEE/ACM Transactions on Networking*, 10:513–528, August 2002.

[41] G. Fernandes and P. Owezarski. Automated Classification of Network Traffic Anomalies. In *Proc. of 5th International ICST Conference on Security and Privacy in Communication Networks*, 2009.

[42] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782 (Proposed Standard), April 2004.

[43] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397 –413, August 1993.

[44] P. Francois, O. Bonaventure, M. Shand, S. Bryant, and S. Previdi. Loop-free convergence using ofib. *Work in Progress*, 2008.

[45] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second igp convergence in large ip networks. *ACM SIGCOMM Computer Communication Review*, 35(3):35–44, 2005.

[46] Michael J. Freedman, Karthik Laskhminarayanan, and David Mazières. OASIS: Anycast for any service. In *POT3rd Symposium on Networked Systems Design and Implementation*, San Jose, CA, May 2006.

[47] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[48] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of internet paths with one-hop source routing. In *Proceedings of OSDI*, Berkeley, CA, USA, 2004. USENIX Association.

[49] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proc. of the ACM/SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[50] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, IMC '05, pages 4–4, Berkeley, CA, USA, 2005. USENIX Association.

[51] Sun Hanlin, Jin Yuehui, Cui Yidong, Wang Hongbo, and Cheng Shiduan. Improving fairness of red aided by lightweight flow information. In *2nd IEEE International Conference on Broadband Network Multimedia Technology, 2009. IC-BNMT '09.*, pages 335 –339, 2009.

[52] Yihua He, Michalis Faloutsos, Srikanth Krishnamurthy, Bradley Huffaker, Yihua He, Michalis Faloutsos, Srikanth Krishnamurthy, and Bradley Huffaker. On routing asymmetry in the Internet. In *Proceedings of IEEE Globecom*, 2005.

[53] Christian Hennig and Mahmut Kutlukaya. Some thoughts about the design of loss functions. *REVSTAT–Statistical Journal*, 5(1), 2007.

[54] C.E. Hopps. Analysis of an equal-cost multi-path algorithm. 2000.

[55] N. Hu and P. Steenkiste. Quantifying internet end-to-end route similarity. In *Passive and Active Measurement Conference*, volume 2006, page 76. Citeseer, 2006.

[56] Tiansi Hu and Yunsi Fei. Qelar: A machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks. *IEEE Transactions on Mobile Computing*, 9(6):796 –809, 2010.

[57] Xiao Huang, Jiangxing Wu, Guibin Sun, and Jing Jing. A new fair active queue management algorithm. In *Future Networks, 2009 International Conference on*, pages 191 –195, 2009.

[58] V. Jacobson. Congestion avoidance and control. *SIGCOMM Computer Communication Review*, 18:314–329, August 1988.

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

[60] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical report, Digital Equipment Corporation, September 1984.

[61] A. Jayaraj, T. Venkatesh, and C.S.R. Murthy. Loss classification in optical burst switching networks using machine learning techniques: improving the performance of tcp. *IEEE Journal on Selected Areas in Communications*, 26(6):45 –54, 2008.

[62] M. Kaafar, F. Cantin, B. Gueye, and G. Leduc. Detecting triangle inequality violations for internet coordinate systems. In *Proc. of Future Networks 2009 workshop*, Dresden, Germany, June 2009.

[63] Qifa Ke and Takeo Kanade. Robust l1 norm factorization in the presence of outliers and missing data by alternative convex programming. In *Computer Vision and Pattern Recognition*, pages 592–599, 2005.

[64] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. In *Journal of the Operational Research Society*, volume 49, 1998.

[65] Raghunandan H. Keshavan, Sewoong Oh, and Andrea Montanari. Matrix completion from a few entries. *CoRR*, abs/0901.3150, 2009.

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

[67] C. Kruegel and T. Toth. Using decision trees to improve signature-based intrusion detection. In *Proc. of the 6th International Workshop on Recent Advances in Intrusion Detection (RAID)*, 2003.

[68] Chamil Kulatunga and Gorry Fairhurst. Enforcing layered multicast congestion control using ecn-nonce. *Computer Networks*, 54(3):489 – 505, 2010.

[69] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proc. of ACM SIGCOMM*, 2004.

[70] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc. of ACM SIGCOMM*, 2005.

[71] Steven Latré, Wim Van de Meerssche, Stijn Melis, Dimitri Papadimitriou, Filip De Turck, and Piet Demeester. Automated management of network experiments and user behaviour emulation on large scale testbed facilities. In *Proceedings of the 6th International Conference on Network and Service Management (CNSM 2010)*, 2010.

[72] J. Ledlie, P. Gardner, and M. I. Seltzer. Network coordinates in the wild. In *Proc. of USENIX NSDI*, Cambridge, April 2007.

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

[74] J. Lee, H. Lee, S. Sohn, J. Ryu, and T. Chung. Effective value of decision tree with kdd99 intrusion detection datasets for ids. In *Proc. of the 10th International Conference on Advanced Communication Technology*, 2008.

[75] S. Lee, Z. Zhang, S. Sahu, and D. Saha. On suitability of Euclidean embedding of Internet hosts. *SIGMETRICS*, 34(1):157–168, 2006.

[76] K. Leung and C. Leckie. Unsupervised anomaly detection in network intrusion detection using clustering. In *Proc. of the 28th Australasian Conference on Computer Science*, 2005.

[77] Y. Liao, W. Du, P. Geurts, and G. Leduc. Decentralized prediction of end-to-end network performance classes. In *ACM CoNext*, Tokyo, Japan, December 2011.

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

[79] Y. Liao, M. Kaafar, B. Gueye, F. Cantin, P. Geurts, and G. Leduc. Detecting triangle inequality violations in internet coordinate systems by supervised learning - work in progress. In *Proc. of Networking 2009*, Aachen, Germany, May 2009.

[80] Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19:2756–2779, Oct 2007.

[81] E. K. Lua, X. Zhou, J. Crowcroft, and P. V. Mieghem. Scalable multicasting with network-aware geometric overlay. *Comput. Commun.*, 31(3):464–488, 2008.

[82] C. Lumezanu, R. Baden, D. Levin, N. Spring, and B. Bhattacharjee. Symbiotic relationships in internet routing overlays. In *Proceedings of NSDI*, pages 467–480, Berkeley, CA, USA, 2009. USENIX Association.

[83] Yun Mao, Lawrence Saul, and Jonathan M. Smith. IDES: An internet distance estimation service for large networks. *IEEE Journal On Selected Areas in Communications*, 24(12):2273–2284, December 2006.

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

[85] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *Proceedings of SIGCOMM*, pages 11–18, New York, NY, USA, 2003. ACM.

[86] A. Nakao, L. Peterson, and A. Bavier. Scalable routing overlay networks. *SIGOPS Oper. Syst. Rev.*, 40(1):49–61, 2006.

[87] Nathan Srebro Nati and Tommi Jaakkola. Weighted low-rank approximations. In *International Conference on Machine Learning*, pages 720–727, 2003.

[88] 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.

[89] T. S. E. Ng and H. Zhang. A network positioning system for the Internet. In *Proc. of USENIX Annual Technical Conference*, June 2004.

[90] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski. Sndlib 1.0 - survivable network design library. *Networks*, 55(3):276–286, 2010.

[91] D. Papadimitriou, M. Welzl, M. Scharf, and B. Briscoe. Open Research Issues in Internet Congestion Control . RFC 6077, February 2011.

[92] Abhinav Pathak, Himabindu Pucha, Ying Zhang, Y. Charlie Hu, and Z. Morley Mao. A measurement study of internet delay asymmetry. In *Proc. of the Passive and Active Measurement*, Cleveland, OH, USA, April 2008.

[93] *PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services*, 2002. `http://www.planet-lab.org`.

[94] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering. In *Proc. of ACM CSS Workshop on Data Mining Applied to Security*, 2001.

[95] M. Pustisek, I. Humar, and J. Bester. Empirical analysis and modeling of peer-to-peer traffic flows. In *The 14th IEEE Mediterranean Electrotechnical Conference, 2008. MELECON 2008*, pages 169 –175, May 2008.

[96] B. Puype, D. Papadimitriou, G. Das, D. Colle, M. Pickavet, and P. Demeester. Srlg inference in ospf for improved reconvergence after failures. *Towards a Service-Based Internet*, pages 233–234, 2010.

[97] B. Puype, D. Papadimitriou, G. Das, D. Colle, M. Pickavet, and P. Demeester. Ospf failure reconvergence through srg inference and prediction of link state advertisements. In *Proceedings of the ACM SIGCOMM 2011 conference on SIGCOMM*, pages 468–469. ACM, 2011.

[98] F. Raineri and G. Verticale. Early internet application identification with machine learning techniques. In *First International Conference on Evolving Internet, 2009. INTERNET '09*, pages 60 –64, 2009.

[99] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, San Diego, CA, USA, August 2001.

[100] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *Proc. IEEE INFOCOM*, New York, NY, USA, June 2002.

[101] Ramin Sadre and Boudewijn Haverkort. Changes in the web from 2000 to 2007. In *Managing Large-Scale Service Deployment, Proceedings of the International Workshop on Distributed Systems: Operations and Management (DSOM)*, volume 5273 of *Lecture Notes in Computer Science*, pages 136–148. Springer Berlin / Heidelberg, 2008.

[102] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: Informed internet routing and transport. *IEEE Micro*, 19(1):50–59, 1999.

[103] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of internet path selection. *SIGCOMM Comput. Commun. Rev.*, 29(4):289–299, 1999.

[104] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. Tcp congestion control with a misbehaving receiver. *SIGCOMM Comput. Commun. Rev.*, 29:71–78, October 1999.

[105] Shai Shalev-Shwartz, Alon Gonen, and Ohad Shamir. Large-Scale Convex Minimization with a Low-Rank Constraint. In *International Conference on Machine Learning*, 2011.

[106] M. Shand and S. Bryant. IP Fast Reroute Framework. RFC 5714 (Informational), January 2010.

[107] Rob Sherwood, Bobby Bhattacharjee, and Ryan Braud. Misbehaving tcp receivers can cause internet-wide congestion collapse. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS '05, pages 383–392, New York, NY, USA, 2005. ACM.

[108] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Daisuke Inoue, Masashi Eto, and Koji Nakao. A comparative study of unsupervised anomaly detection techniques using honeypot data. *IEICE Transactions on Information and Systems*, E93.D(9):2544–2554, 2010.

[109] D. Sontag, Y. Zhang, A. Phanishayee, D. G. Andersen, and Karger D. Scaling all-pairs overlay routing. In *Proceedings of CoNEXT*, Rome, Italy, December 2009.

[110] N. Spring, D. Wetherall, and D. Ely. Robust Explicit Congestion Notification (ECN) Signaling with Nonces. RFC 3540 (Experimental), June 2003.

[111] GÃ¡bor TakÃ¡cs, IstvÃ¡n PilÃ¡szy, BottyÃ¡n NÃ©meth, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, June 2009.

[112] Liying Tang and Mark Crovella. Virtual landmarks for the Internet. In *Proc. of ACM/SIGCOMM Internet Measurement Conference*, Miami, FL, USA, October 2003.

[113] Eric van den Berg, Mariusz A. Fecko, Sunil Samtani, Catalin Lacatus, and Mitesh Patel. Distributed game-theoretic topology control in cognitive networks. volume 7707, page 77070E. SPIE, 2010.

[114] Vuze Bittorrent. `http://www.vuze.com/`.

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

[116] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, 2002.

[117] B. Wong, A. Slivkins, and E. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proc. of ACM SIGCOMM*, August 2005.

[118] Rongmei Zhang, Chunqiang Tang, Y. Charlie Hu, Sonia Fahmy, and Xiaojun Lin. Impact of the inaccuracy of distance prediction algorithms on Internet applications: an analytical and comparative study. In *Proc. of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[119] H. Zheng, E. K. Lua, M. Pias, and T. Griffin. Internet Routing Policies and Round-Trip-Times. In *Proc. of the Passive and Active Measurement*, Boston, MA, USA, April 2005.

[120] Xiaoyun Zhu, Jie Yu, and J. Doyle. Heavy tails, generalized coding, and optimal web layout. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1617 –1626 vol.3, 2001.