Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Network Architectures and Services

Dragon-Lab, Network States Detection and Identification Framework: Performance Investigation

Alexandru Calu
(4040384)

Committee members:

Supervisor: Dr.ir. F.A. Kuipers
Member: Dr. A. Iosup
Member: Dr. C. Doerr

August 31, 2011
M.Sc. Thesis No: PVM 2011 – 070.

Abstract

As IP networks have become the support of an increasingly varied range of applications, from games, video streaming, to e-commerce and online banking, it is critical to understand the functioning and performance bounds of the network in order to provide a certain QoS.

To be able to achieve the understanding of the whole network huge amounts of data have to be acquired and processed. Various methods of analyzing the network have been created, such as: analysis of point-to-point packet delay, delay tomography from end-to-end unicast measurements, network unreachabilities troubleshooting using end-to-end probes and routing data, analysis of link failures in an IP backbone etc.

In this research, a new approach on Internet analysis was adopted. Dragon-Lab is capable of detection, identification, and temporal and spacial localization of Internet backbone anomalous states based on two end-to-end metrics, packet delay and packet loss, and traceroutes information of the measured Internet paths. The anomalous states or instabilities are: congestion, queue building up and link failure.

This is a new approach in the field of network troubleshooting and management. It is a bridge between methods focusing mainly on delay analysis and root cause analysis methods focusing on harvesting huge amounts of routing data in order to identify and localize network problems. Dragon-Lab is based on a three months measurement study conducted over the Internet between Norway, China and New Zealand.

Relying on the Principal Component Pursuit processed data to identify anomalous delay changes and on general network knowledge, the unstable network states are identified and implicitly temporally localized.

Further analysis is performed by processing instabilities into cumulative distributed functions of duration of instabilities, time between instabilities, charts of the distributions of instabilities per path, and per type of instability, in order to study instabilities impact on Internet paths. Moreover, general metrics have been defined in order to give an overview of the impact of the combination of instabilities on paths. The metrics are: availability, fatigue or stability of the Internet path.

Further Dragon-Lab is improved by geo-localization of the instabilities. The research aims to use available data to pinpoint the source location of instabilities on the Internet paths, in essence to find the problem hop on the path by IP. This is achieved by combining reconstructions algorithms from the Compressive Sensing domain with Dragon-Lab knowledge on instabilities. The approach presents provides a solution bounded by certain conditions and having a few limitations and also a spacial localization algorithm.

Contents

List of Figures

List of Tables

Acronyms

AS	Autonomous System
BGP	Border Gateway Protocol
BP	Basis Pursuit
CDF	Cumulative Distribution Function
CRUDE	Collector for RUDE receiver
CS	Compressive Sensing
DC	Digital California
DDM	Delay Distribution Measurement
DNS	Domain Name System
eBGP	External BGP
GMT	Greenwich Mean Time
GPS	Global Positioning System
GPSR	Gradient Projection Sparse Recovery
HPR	High-performance Research
IALM	Inexact Augmented Lagrange Multiplier
iBGP	Internal BGP
IGP	Interior Gateway Protocol
IS-IS	Intermediate System To Intermediate System protocol
ISP	Internet Service Provider
K	Sparsity
LAN	Local Area network
MIB	Management Information Base
NTP	Network Time Protocol
PCA	Principal Component Analysis
PCP	Principal Component Pursuit
QoS	Quality of Service
QVPN	Virtual Private Networks with QoS
RIP	Restricted Isometry Property
RTT	Round-Trip Time
RUDE	Real-time User datagram protocol Data Emitter
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SNR	Signal to Noise Ratio
SVD	Singular Value Decomposition
TIV	Triangle Inequality Violations

Chapter 1

Introduction

1.1 Introduction

Telecommunications play an ever increasing role in today's highly interconnected world. One of the key roles is held by the Internet. Being of great importance as communications infrastructure, it is essential to be able to measure its performance, detect, isolate and prevent problems.

However, since the size and complexity of the Internet continuously increased, a thorough measurement of all its characteristics would be impossible. Regularly, portions of the Internet are measured, typically within an ISP, a research network or measurements between research networks are conducted.

In all cases metrics have to be defined in order to measure the network performance. They can be classified into: 1) packet loss; 2) packet delay; 3) bandwidth. They are the metrics ISPs use to define their Service Level Agreements (SLAs). Packet loss characterizes the congestion level of the network, measuring the packets lost in the network due to buffer overflows. A second important parameter for high QoS in the network and essential parameter in real time applications is packet delay; it characterizes the level of congestion and additionally the effect of routing changes; ISPs compute packet delay over the entire network and average it over a month. Bandwidth shows how much data can be transferred within a time unit, dependent or not on the traffic conditions within the network.

Each of the metrics described requires specialized methods of measuring. To be able to obtain a general picture of the whole network huge amounts of data have to be acquired and processed. Consequently, there has been an increasing need for practical and efficient procedures to achieve this goal. Researchers have tried various methods of analyzing the network such as: Internet mapping, analysis of point-to-point packet delay, delay distribution measurement, delay tomography from end-to-end unicast measurements, network unreachabilities troubleshooting using end-to-end probes and routing data or analysis of link

failures in an IP backbone etc. They all try to give a better understanding of the Internet's and large networks' behavior in the most efficient way possible.

In this research, a new approach on Internet analysis was adopted. A network troubleshooting framework called Dragon-Lab capable of identification, temporal and spacial localization of Internet backbone states based on two end-to-end metrics, packet delay and packet loss, and traceroutes information of the measured Internet paths, has been developed. The possible network states considered for identification in this work are: stable state, congestion, queue building up and link failure. Three of them represent instabilities in the network; Dragon-Lab will be used to analyze their behavior and geo-localize them.

This is a new approach in the field of network troubleshooting and management. It is a bridge between methods focusing mainly on delay analysis and root cause analysis methods focusing on harvesting huge amounts of routing data in order to identify and localize network instabilities.

The way Dragon-Lab is able to achieve the previously mentioned goals is briefly presented in the next section of the Introduction and more thoroughly in the rest of the report.

1.2 Problem Description

As IP networks have become the support of an increasingly varied range of applications, from games, video streaming, to e-commerce and online banking, it is critical to understand the functioning and performance bounds of the network in order to provide a certain QoS. This understanding is based on measurements of various metrics and parameters for being able to control and manage the network. Although many studies have been conducted on Internet measurement the problem of the performance of its paths is still little understood.

Therefore the new approach of Dragon-Lab will be used for further investigation of Internet paths behavior and to find a solution for identifying and localizing Internet paths instabilities. Dragon-Lab is based on a measurement study conducted over the Internet between Norway, China and New Zealand. Three months of packet delay, packet loss and traceroute data measurements was the only information available for analysis.

The analysis was based on detecting and identifying significant unusual changes in the packet delay and packet loss. In this way there was no need for further information regarding the structure, operations or events in the Internet backbone from the ISPs' side. The problem is however challenging given the fact that huge amounts of noisy data have to be processed in order to extract meaningful information.

The dimensions problem was handled in two stages: aggregation of delay data and analysis of aggregate data with the use a convex program called Principal Component Pursuit (PCP) in order to identify abrupt variations in the aggregated delay. Extra knowledge about abnormal delay events detected using PCP has been used in order to filter out physically impossible anomalies which might have been "detected".

Based on the PCP analysis data and general network knowledge, the detected anomalous delay events are classified using three metrics, which are defined as backbone features space, in essence: aggregated delay, aggregated losses and average delay. It is proven that by combining these three metrics the four network states named - that is stable state, congestion, queue building up and link failure - can be identified.

So far network instabilities can be identified and temporarily localized. However, this is still raw information about the Internet paths. For a better understanding of the performance of these paths, the available information has to be combined into a logical, tractable analysis. This is achieved by presenting the detected instabilities in expressive forms, in order to offer a statistical analysis of the problems. For this reason a performance evaluation of the four measured Internet paths between Norway, China and New Zealand is performed by creating statistics representing: cumulative distributed functions of duration of instabilities, time between instabilities, charts of the distributions of instabilities per path, and per type of instability. Moreover general metrics have been defined in order to give a better overview of the impact of the combination of instabilities per path. Such metrics are: availability, fatigue or stability of the Internet path.

The temporal localization of Dragon-Lab is improved by a geo-localization of the instabilities. This part of the research focuses on using the Dragon-Lab available data in order to pinpoint the source of instabilities on the Internet paths, in essence to find the problem hop on the path by IP. This is achieved by combining algorithms from the Compressive Sensing domain with Dragon-Lab knowledge on instabilities. The approach provides a feasible solution and presents its limitations and boundaries for applying it. Moreover a localization algorithm is designed based on the results of previous analysis.

1.3 Scope of Research

The research was started at the proposal of an Internet company in Norway, UNINETT AS., to analyze the end-to-end Internet measurements data they have collected over a period of three months, particularly packet delay, packet loss and traceroute data.

The high level goal of this research is to be able to diagnose Internet paths behavior. This diagnosis consists of several composing targets:

- detection and identification of network states: stable states, link failures, congestions and queues build, based on end-to-end noisy measurements;
- analysis of instabilities impact on network performance, per type and in general;
- spacial localization of instabilities.

Generally, techniques that aim to diagnose network paths performance have two directions:

- either analyze end-to-end delay in order to achieve as accurate measurements as possible and create delay distributions [1]; or they use end-to-end delay measurements to infer the delay distribution within the network [2] or for bandwidth estimation, admission control and other decision making schemes [3]; these are generally Internet wide measurements;
- or analyze failures within the network based on protocols routing information [4], [5], syslog messages, administrator e-mails, router configurations [6], [8], to be able to localize the failure; these techniques are conducted on large scale networks and mainly suffer from scalability problems if applied Internet wide.

Dragon-Lab aims to combine these two approaches, the input data of the delay analysis techniques, namely delay, and part of the input data from the network troubleshooting methods, particularly traceroutes, in order to offer instabilities characteristics and spacial localization which could be used further on in quality management of large IP networks; in this sense Dragon-Lab situates itself in between the two types of techniques for network analysis.

1.4 Report Structure

Introduction: In this chapter a short description of challenges present in today's Internet is given, as well as the approach to cope with them. Moreover, a new approach of instabilities troubleshooting, namely Dragon-Lab framework, is shortly presented, together with its capabilities and the motivation behind its development.

Background: This chapter offers background information on previously developed methods of delay analysis and network troubleshooting; approaches on analysis of network available data are presented for a better positioning of the Dragon-Lab framework in the wide range of network troubleshooting techniques.

The Dragon-Lab Framework: This chapter gives a detailed description of the Dragon-Lab framework. First an overview of the method is provided; secondly the measurement methodology for obtaining the data the framework is based on and the methodology of analyzing the data are offered; further on, the mathematic and algorithmic foundation of the framework and finally the output structure of the data Dragon-Lab provides are presented.

Dragon-Lab Performance Evaluation: In this chapter the performance of detection and classification into unstable network states of the detected instabilities is evaluated. This performance is essentially depending on one tuning parameter used by the PCP algorithm. The effect of the tuning parameter on the framework is studied by analyzing detection rates false positive rates and receiver operator characteristic curves.

Dragon-Lab Internet Paths States Analysis: This chapter is dedicated to one of the applications of the framework, namely analysis of Internet paths. Four Internet paths have been monitored and measured over a three months period. They provide the necessary data for the Dragon-Lab analysis. The chapter comes with a methodology for Dragon-Lab output information analysis in order to give a suggestive overview of the behavior of the detected instabilities. Moreover, future simulations may be based on results provided in this chapter.

Dragon-Lab Instabilities Localization: This chapter presents the second important feature of Dragon-Lab: geo-localization of the detected instabilities. Localization of instabilities is a challenge in Internet paths analysis based on only end-to-end delay, packet loss information and traceroutes. The results, performance, conditions and limits for achieving this goal are

given in this chapter. Moreover a localization algorithm is proposed. A presentation of the Compressive Sensing techniques, used for solving the localization problem, is also given.

Conclusion and Future Work: This chapter summarizes the thesis work and suggests the future direction in which the work could be carried forward.

Chapter 2

Background

In order to emphasize the value of Dragon-Lab, and to better position it within the Internet/delay analysis research, an overview over the work that has already been done related to the topic is given.

This chapter is structured in two parts: network delay analysis and network troubleshooting techniques. The reason is that, generally, techniques that aim to diagnose network paths performance focus on two directions:

- either they analyze end-to-end delay in order to achieve as accurate measurements as possible and create delay distributions [1]; or they use end-to-end delay measurements to infer the delay distribution within the network [2] or for bandwidth estimation, admission control and other decision making schemes [3]; these are generally Internet wide measurements;
- or they analyze failures within the network based on protocols routing information [4], [5], syslog messages, administrator e-mails, router configurations [6], [7], to be able to localize the failure; these techniques are conducted on large scale networks and mainly suffer from scalability problems if applied Internet wide.

Dragon-Lab is a network troubleshooting framework based on delay analysis, thus it combines the above directions, but has rather different features from the other method. To be able to notice what the differences are, a few of the techniques will be presented further on.

2.1 Existing Approaches on Network Delay Analysis

There are many results in the literature regarding delay measurement. The focus has been laid on designing approaches to accurately measure delay or create delay distributions [1], to infer link delay distribution from end-to-end measurements [2] or to make use of the delay measurement in bandwidth estimation, admission control and other decision making schemes [3]. This section is dedicated to understanding the principles and reasoning behind the existing approaches for network delay analysis.

2.1.1 Probabilistic Delay Guarantees using Delay Measurement

To start with, a technique of using delay measurement for the creation of probabilistic delay guarantees that will be used for admission control and bandwidth estimation is presented. This technique is thoroughly discussed in reference [3].

Service providers always try to differentiate their offering by means of customized services, such as virtual private networks (VPN) with QoS guarantees – QVPNs. The incentive of this research initiative is that carriers try to maximize the number of admitted QVPNs based of the statistical multiplexing of the input traffic along the bandwidth dimension, but do not exploit the statistical multiplexing along the delay dimension in order to offer different per-QVPN delay bounds. Thus, the challenge is to maximize the utilization efficiency of the infrastructure while still supporting the QoS requirements for each QVPN. This can be achieved by using an admission control algorithm that admits the maximum number of QVPNs while allocating the least resources to satisfy their QoS needs. Deterministic admission allocates the necessary resources in order to never violate the QoS requirements. Regarding the delay guarantees, a deterministic admission algorithm would ensure that the worst case delay would never be exceeded by any of the packets involved. However, in practice, these worst case delays are rarely happening and consequently network resources remain unutilized.

The practical experience shows that QVPN's aggregate real-time traffic on the long term has a stable nature and therefore two statistical effects can be exploited:

- tolerance to delay violations: most real-time applications tolerate some excess delay or small packet loss in the network traffic. If 99.9% of the packets experience at most 50% of the worst case delay then an admission control mechanism would allocate only 50% of the resources a deterministic admission control algorithm would;
- statistical multiplexing along delay dimension: due to statistical multiplexing the peak traffic bursts of different QVPNs do not coincide, consequently packets delay will rarely reach the worst case delay on all QVPNs simultaneously. To prove the multiplexing effect they aggregated the ON-OFF packet traces for different number of VoIP sessions. It can be observed from figure 2.1 that there is almost never the case when more than 40% of the sessions are simultaneously active.

Relying on the two statistical assumptions previously mentioned the authors come up with a new measurement-based admission control algorithm called Delay Distribution Measurement (DDM) based admission control, which will maximize the number of admitted QVPNs with QoS requirements. The requirements include delay bound, delay violation probability bound and long-term average bandwidth.

DDM provides the following features using admission control based on delay statistical distribution:

- statistical multiplexing along delay dimension; previous deterministic algorithms only exploited the bandwidth dimension, which translates into an over-allocation of resources, since QVPNs rarely transmit at rates stated in the long-term bandwidth requirement.
- distinct per-QVPN probabilistic delay bounds: DDM can differentiate between QVPN's tolerance to delay bound violations. More sensitive QVPNs will have allocated more resources.
- unified support for probabilistic and deterministic delay bounds: DDM is one admission framework which can work with deterministic or probabilistic bounds. Deterministic delay bounds mean that the tolerance to delay violations is zero.

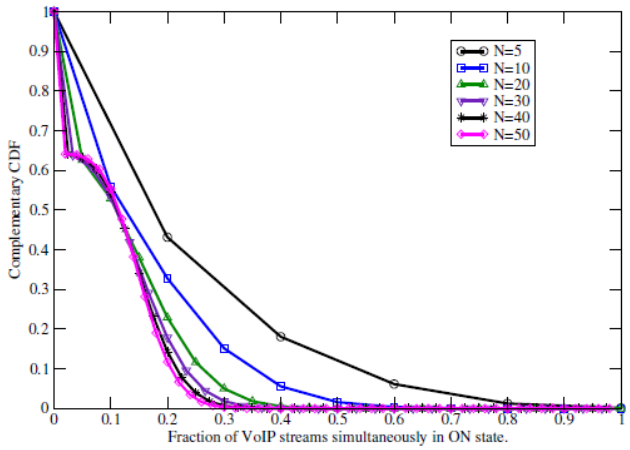


Fig. 2.1 Complementary CDF of the fraction of VoIP sessions in ON state simultaneously as the number of VoIP sessions (N) in aggregate QVPN is varied [3]

DDM works in the following way: it measures the delay of each packet, calculates the ratio between the measures packet delay and the worst case delay that the packet could experience and creates a delay ratio distribution; this is shown in figure 2.2 out of a simulation for 39 VoIP QVPNs. This is used to infer the needed bandwidth to assure a given probabilistic delay bound. Thus the novelties introduced by DDM are the construction of the CDF of the ratio between measured packet delay and the worst case delay and the resource mapping based on the CDF curve.

The formula which gives the delay-derived bandwidth requirement $\rho_{i,l}^{delay}$ of QVPN i at link l is:

$$\rho_{i,l}^{delay} = \frac{\sigma_i + L_{max}}{\frac{D_{i,l}}{Prob^{-1}(1-P_{i,l})} - \frac{L_{max}}{C_l}}$$, where L_{max} is the maximum packet size, $D_{i,l}$ is a certain delay bound, $P_{i,l}$ is the delay violation probability bound, σ_i is the $QVPN_i$'s burst size and C_l is the total capacity of link l.

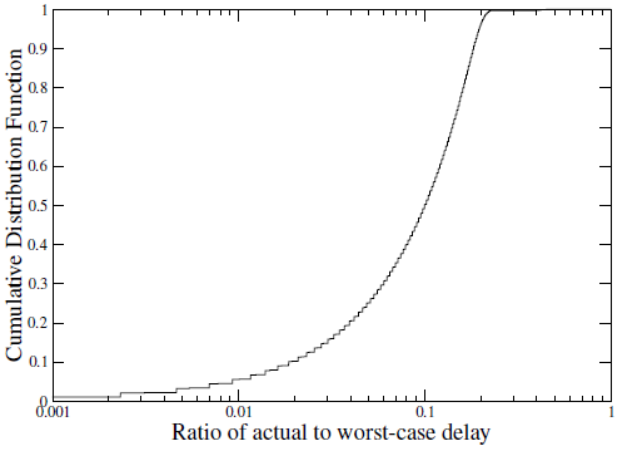


Fig. 2.2 CDF of the ratio of actual delay to worst case delay experienced by packets [3]

The expression clearly shows that the bandwidth is allocated by exploiting the statistical multiplexing along the delay dimension.

This initiative proves that the use of dynamic delay measurement can result in less resources allocated to the same number of simultaneously used QVPNs in this case, and therefore a maximization of the utilization of network resources.

2.1.2 Network Delay Tomography

Another approach on analyzing and using the network delay to achieve more value from it is presented in reference [2]. In this section it will be briefly introduced. The research paper is called Network Delay Tomography from End-to-end Unicast Measurements. Its goal is to reconstruct the network internal performance based on the end-to-end traffic behavior. It makes use of the fact that correlation seen on intersecting end-to-end paths can be exploited to draw inferences about the performance of the common portion without any other information. The technique is better suited for multicast traffic. However, if the unicast one has similar characteristics then the results can also be used.

The idea of multicast based delay inference is illustrated using figure 2.3. The source node O sends multicast packets to the leaf nodes L and R. End-to-end delay measurements are conducted. If one assumes that the delay on the received packet on the L branch is zero, then the additional delay on the R branch will be attributed to the C-R link; consequently an estimate of the delay distribution will be created. The technique works with unicast packets also. However, they should have the correlation resembling the one of the multicast packets. If for example two unicast packets are sent from the source, and one of them will be considered as having zero delay, the estimate of the delay from C to R will contain some error because it is possible that the packet experience slightly different delay on the link from O to C. Therefore a systematic error will be introduced because they consider that same paths have the same delay.

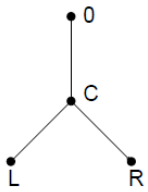


Fig. 2.3 2-Leaf Tree [2]

The authors come with techniques to estimate the delay distribution on links using end-to-end packet delay measurements. For doing this they model link delay by non-parametric discrete distributions. They use time bins, discretized versions of the true delay distribution, in order to reduce computation costs. In order to avoid losing accuracy or excessively increasing computational cost, a new technique in delay modeling is being used: variable size time bins according to the concentrations of probability mass. Then they also produce a method to estimate the per link delay variance.

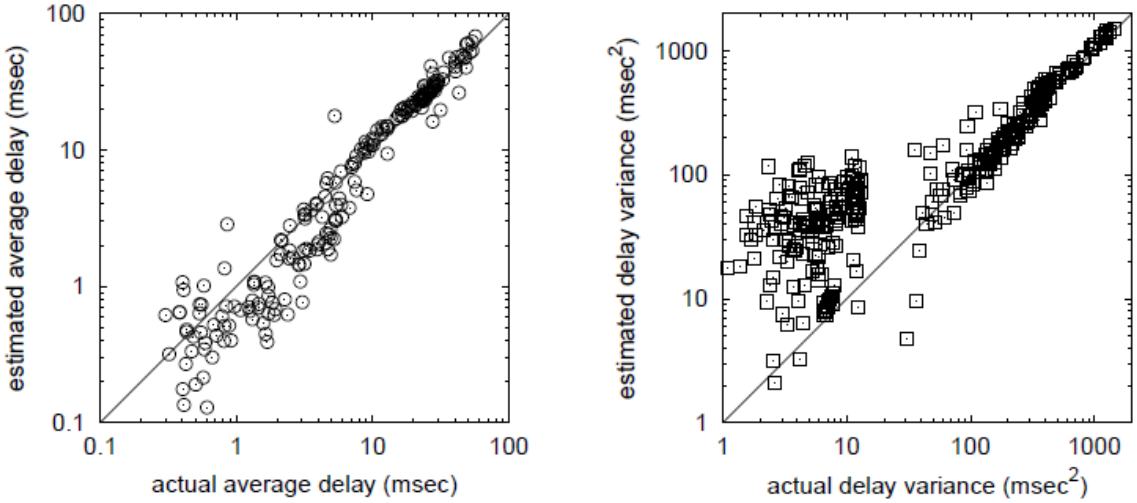


Fig. 2.4 Inferred vs. actual average and variance of link delay in simulations [2]

The delay analysis, namely estimation of the probability distribution of per link variable delay D_k, can be done using two concepts: a fixed bin size discrete model and a variable bin

size discrete model. Thus for the analysis they model the link delay by a non-parametric discrete distribution.

The results of this technique are shown in figure 2.4. As it can be observed accuracy increases for higher values of delays and delay variance. Estimates are more accurate for example if delay's mean is larger than 10ms and the variance larger than $10ms^2$, then the error will be 10% and 11.57%.

The technique performs fairly well in estimating the delay on the link within a network based on end-to-end delay measurements; no other information is used; however, there are some limitations. It is based on the assumption of using multicast measurements, which might be a problem because large portions of the Internet do not support network-level multicast, but the largest part of the traffic is unicast. Moreover, the technique might suffer from lack of accuracy. In the need for low computational cost a trade-off has to be made between accuracy and complexity.

2.1.3 Measurement-based Analysis of Internet Delay Space

A third approach regarding the network delay analysis is introduced in Measurement-based Analysis, Modeling, and Synthesis of the Internet Delay Space [1]. The goal of this study is to understand the characteristics of the Internet delay space, design a model of the delay space, and synthesize delay data for simulations using a tool called DS^2.

Their motivation was that no current delay space model for large-scale simulations captures the characteristics of the Internet space delay. To prove this they conducted measurements on the round-trip static propagation delay over globally distributed DNS servers using the King tool. Afterwards they created artificially generated delay matrices based on already existing topology models for large scale networks. The generators they used were Inet and GT-ITM. In order to match the King tool measurements only degree 1 nodes from the generated topologies were selected. For generating the delays with Inet, Euclidean distance between a pair of connected nodes was used as link delay; for generating the delay with GT-ITM artificial link delays were used for the paths selected using shortest path routing performed over topology with routing policy weights as link costs. Consequently, triangle inequalities were artificially introduced, as opposed to Inet where there are no triangle inequality violations. The delays in the delay matrices are then scaled so that the average delay matches the average delay in the measured data. The result is as in figure 2.5. It is clear that the generators do not create a realistic model of the delay space in the Internet. The delay distribution of the real measured data presents three peaks, which suggest that nodes form clusters in the data.

Therefore, the authors define new metrics, such as: global clustering, local clustering, growth metrics and triangle inequality violations (TIV), in order to capture important

characteristics in the design of distributed systems and in their evaluation. The algorithms used for defining them can be seen in [1].

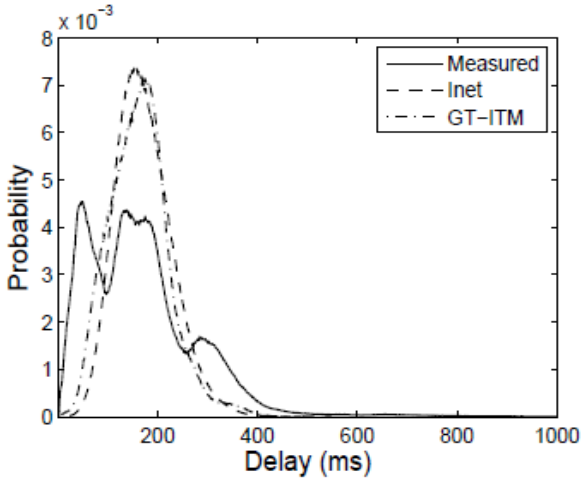


Fig. 2.5 Delay distribution [1]

Using these new metrics the research proceeds to modeling and synthesizing realistic delay spaces based on the characteristics of the measured Internet delay space. In order to achieve this several techniques are designed, which will be combined to develop a delay space synthesizer, named DS^2. Its architecture can be seen in figure 2.6. It consists of a static analysis tool, which analyzes the input measured delay space for generating a synthetic delay space, and a runtime distortion tool which generates synthetic delays upon requests.

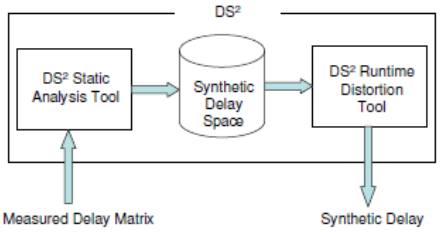


Fig. 2.6 DS^2 architecture [1]

This research adds great value to delay analysis by realistically reproducing the delay space on the Internet. However as the authors convey in the paper there are limitations of the method which have to be considered. Firstly there are limitations generated by the data set used to creating the Internet delay space. The measurements were made among DNS servers, thus the data represents delay space among edges in the Internet. Therefore, the study addresses only edge networks in a wide area network. Secondly, in order to have confidence in the measured data, questionable measurements have been eliminated. They might have different properties from the selected data and consequently important features might be lost.

2.1.4 Network Delay Analysis Conclusion

The three studies presented above give different perspectives on delay analysis: using delay measurements in admission control, inferring the estimated delay over internal links from the end-to-end delay and analyzing large delay date in order to create an Internet delay model.

They focus on delay analysis in different ways, but the goal is to be able to understand better the behavior of delay in order to be able to use it later to achieve better network performance in one way or another.

Dragon-Lab uses also delay measurements for the final goal of improving the performance of the network. However, within this framework the delay measurement is used in a completely different way; Dragon-Lab does not provide a further analysis of the measured delay, but uses its variation in order to achieve a more important goal: identifying and locating the instabilities that determine the characteristics of the delay; in other words it aim for the root cause of delay variations.

A different perspective on network performance analysis is given by the networks troubleshooting techniques. They are presented in the following section.

2.2 Existing Approaches on Network Troubleshooting

Since the Internet is becoming ever more important to various services, its dependability is essential. Consequently, there have been numerous studies on network troubleshooting, in essence instabilities detection, identification and localization.

Since Dragon-Lab framework is itself a network troubleshooting technique, it is worth presenting what other approaches have to offer. Three examples to illustrate the work done in this matter were chosen: Iannacone et al come with a characterization of the failures in the Sprint IP backbone based on the IS-IS routing updates [4]; Turner et al characterize the failures in the CENIC network making use of syslogs, emails logs and router configurations [5] and Huang et al studied network disruptions based of network-wide analysis of BGP routing updates in Abilene network [6].

2.2.1 Characterization of Failures in an Operational IP Backbone Network

Iannacone et al [4] have conducted a thorough study of the failures in the Sprint network IP backbone, by analyzing the IS-IS routing updates of the network's backbone, collected over a seven months period. The study is also related to [7]. IS-IS is the routing protocol in the Sprint network. Sprint uses IP level restoration, therefore if the failure occurs below the IP-layer (router failures, fiber cuts, optical equipment failure, protocol misconfigurations), there will be loss of connectivity between routers, thus a link failure. Their analysis consists of two stages. The first one is to classify failures according to their underlying cause, in essence the network component involved. The classification is done as follows: failures due to maintenance are separated; from the remaining ones, the shared failures are separated; further on, failures that have IP routers in common and the ones that have optical equipment in common are separated. The rest are individual link failures which are also classified based on the number of failures. The classification methodology and its results can be seen in figures 2.7 and 2.8.

The second part of their analysis is to characterize the identified types of link failures. For each class the distributions of the number of failures per link, time between failure and time-to-repair are given. These distributions may afterwards be used to provide a probabilistic failure model which can be used to generate realistic failure scenarios.

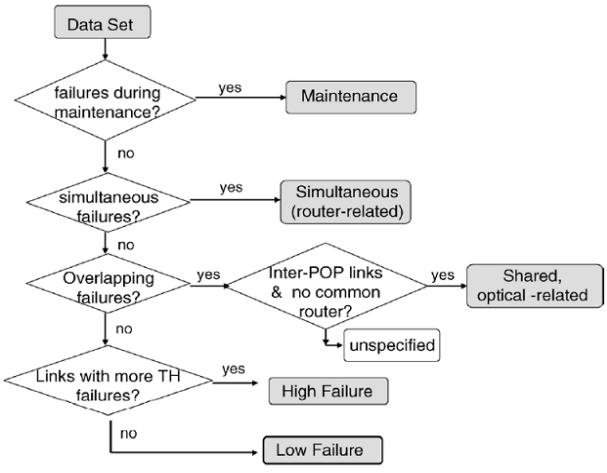


Fig. 2.7 Classification methodology [4]

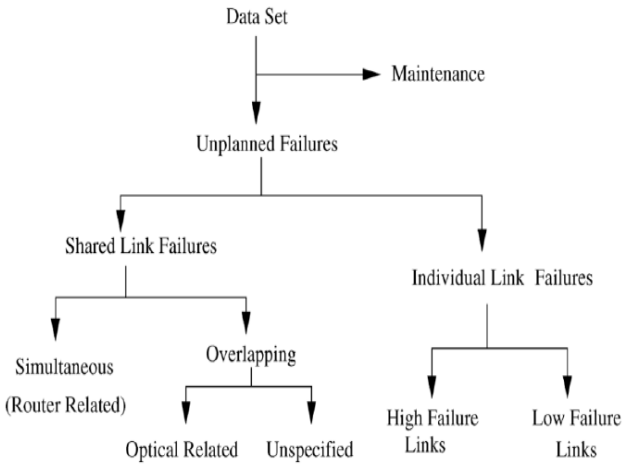


Fig. 2.8 Classification results [4]

The numerical results of their classification methodology are given in table 2.1. It is worth noticing the important percentage of maintenance due failures; also the large number of

high failure links, which are the ones that create most problem for applications. The distributions of the number of failures per class, time between certain types of failures, and time to repair for each class can be seen in the paper.

Failure class		% of all	% of unplanned
Data set		100%	
Maintenance		20%	
Unplanned		80%	100%
Shared	Shared Router-related		16.5%
	Shared Optical-related		11.4%
	Unspecified		2.9%
Individual	High Failure Links		38.5%
	Low Failure Links		30.7%

Table 2.1 Failures partitioning into classes [4]

Thus failures represent an important proportion of the network problems. Failures may happen at different protocol layers for different reasons. At the physical level, a failure may be a fiber cut or optical equipment failure. Hardware failure, in essence linecard failures, router processor overloads, software errors, protocol implementation and misconfigurations errors may also cause loss of connectivity.

This study offers an extensive analysis of the IS-IS failure data in the Sprint network backbone. It is the first study of its kind; the main reason is that such information is not commonly offered by ISPs; this poses a limitation for the research community.

2.2.2 Understanding the Causes and Impact of Network Failures

The authors of [5] conducted a study on the CENIC network with the same goal of identifying the causes of network failures. Contrary to the previous presented study, they are focusing on obtaining data from sources such as router configurations, system logs and email logs, information which is available to network operators without any extra dedicated equipment. Even if the size of the CENIC network (network of approximately 200 routers serving public education and research institutions) is much smaller than Sprint, they are analyzing five years of data, which makes it the largest study of its kind. CENIC network is administratively divided into three components: Digital California (DC) network, High-performance Research (HPR) network and customer-premises equipment (CPE).

Failure events over this period are extracted from syslogs and router configuration data. The cases of the failures are inferred from the administrator email logs and the consistency of the data is verified using independent sources of failure data: active probes of the network

from CAIDA Skitter/Ark effort, BGP logs from Route View Project and administrative announcements from CENIC operators. The failure event reconstruction is done according to the flow in figure 2.9. It is worth to notice the data sources for both the extraction of link failure events and for the validation of the events.

After isolating the failures from the data, the authors proceed with an analysis of the failures duration, cause and impact; similar to what was conducted in the case of the Sprint backbone. The main difference is in the data sources. For the Sprint backbone specialized listeners were employed, whereas in this architecture readily available data is used.

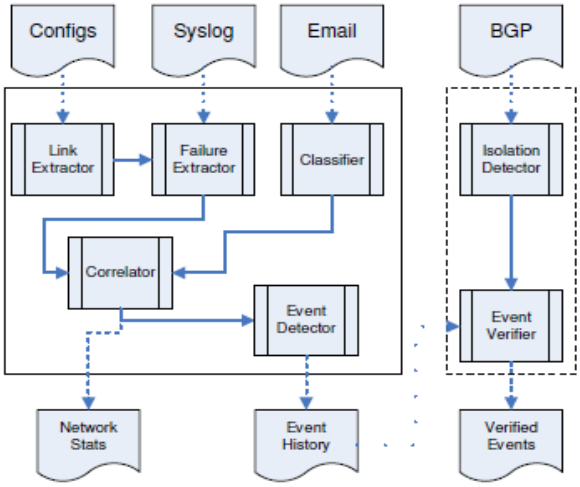


Fig. 2.9 Failure event reconstruction flow [5]

The analysis is aimed to show how often the failures occur, how long they last; to give an understanding of the causes of the failures and to show the impact of the failures, namely how much of the service in the network is disrupted due to the failure.

The results for the three network components regarding the link downtime and the time between failures, for link operating for more than 30 days can be seen in figure 2.10. There can be seen a clear difference depending on the component of the network. Naturally, for the HPR network, which is experimentally focused, the lowest downtime is registered.

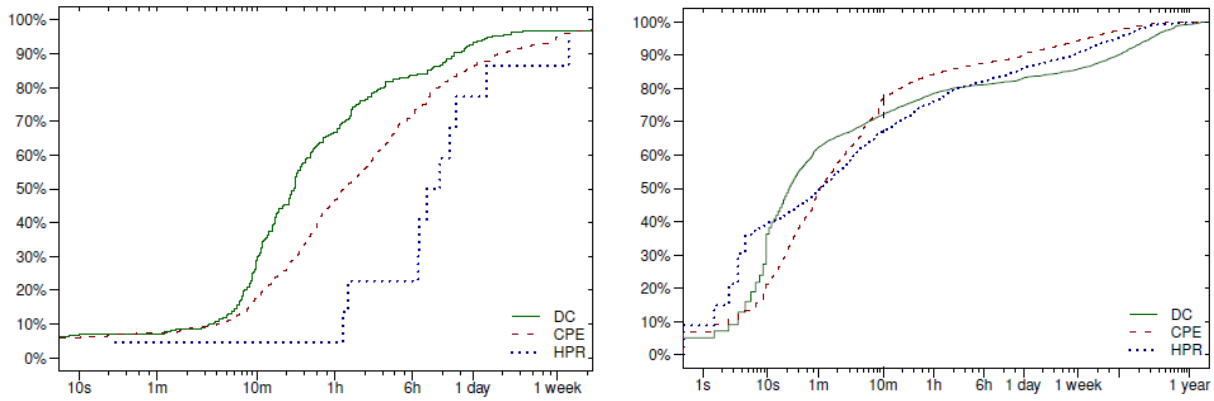


Fig. 2.10 Annualized link downtime (left) ; time between failures (right) [5]

The distribution of the time between failures resembles the distribution of the time between failures of the high failure links from the Sprint network. Additionally to the previous work, in this paper the authors give also an estimate of what fraction of the events were supposed to have an impact on the network, however small.

Using operators' readily available data sources for the analysis makes the approach general and applicable to a variety of IP networks, however, the data sources are private to the network operators and consequently not suitable for the analysis of failures over the Internet.

2.2.3 Diagnosing Network Disruptions with Network-Wide Analysis

Huang et al [6] introduces the first troubleshooting technique that uses network-wide routing information for detecting and identifying network disruptions. The study is conducted over six months, by analyzing BGP routing data from all routers in the Abilene backbone. In order to detect network disruptions a multivariate analysis technique on dynamic routing data (update traffic from all Abilene routers) is used. In this way network-wide dependencies are being exploited which makes it possible to detect even small disruptions and which together with the static routing configuration can point to the disruption's location, namely the involved node or link.

The main findings of the paper are:

- the method to connect documented network disruptions, nodes and link failures, to the BGP routing data. BGP routing data is preferred to Interior Gateway Protocols (IGP) since BGP messages contain information on both internal and external network changes. However it is more noisy (many messages show changes in network conditions, not on network events) and carries little information about the source of the problem.
- exploration of how network-wide analysis exposes classes of network failures. Once the knowledge on how disruptions present themselves in the routing information is acquired, multivariate analysis can be applied to analyze this information. Multivariate analysis allows for multiple statistical variables to be analyzed in parallel; Principal Component Analysis is used.
- localization of network disruptions by combining the analysis of routing dynamics with static configuration analysis. This is called hybrid analysis. Figure 2.11 gives an overview of the framework's operation.

Three types of disruptions are considered, which are all visible in the BGP routing data:

- link disruption. A link disruption will cause routing updates depending on the session that is disrupted and on the number of routes employing the link. A link disruption will definitely cause correlated events.
- periphery ("peer") disruption. A link failure at the edge of a network can affect the way routers inside the network route the traffic to outer network destinations. As in the case of link disruption a periphery one will also cause correlated routing events.
- node disruption. Node failures are generally uncommon.

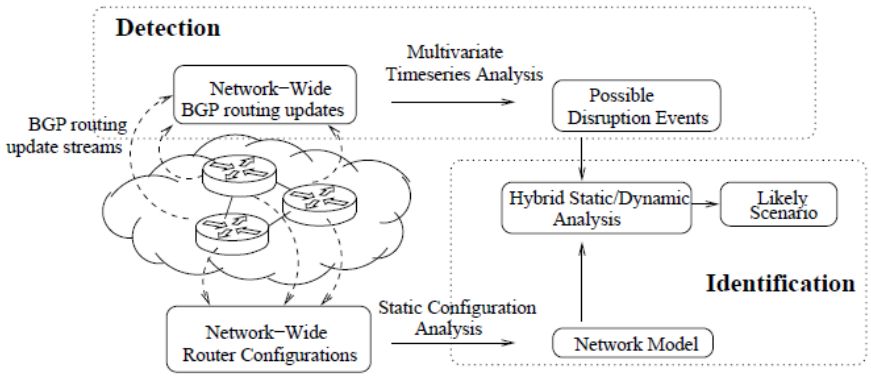


Fig. 2.11 Approach on detection and identification of network disruptions [6]

The results according to the study are shown in table 2.2. The subspace method detects all node and link disruptions, but only 60% of the peer disruptions. This proves that the disruptions can be detected using PCA even if they do not generate a large number of BGP routing updates.

	Visible in BGP	Detected by PCA	Rate
Node	2	2	100%
Link	19	19	100%
Peer	89	54	60.67%

Table 2.2 Number and fraction of network disruptions detected by the subspace method [6]

The study comes up with a new method of network disruptions analysis which gives excellent results and could be a first step in automatic disruptions detection possibly implemented in real environment. The method has two important limitations: its applicability over the Internet is reduced since routing information would have to be acquired from several ASs and secondly they are considering only disruptions caused by complete entity failure, whereas disruptions can be more diverse.

2.2.4 Network Troubleshooting Conclusion

The network troubleshooting techniques presented above target network disruptions, basically link failures, by analyzing complex sets of data: IS-IS routing information, syslog messages, administrator e-mails, network wide router configurations, network wide BGP routing updates etc. This results in strengths, but also limitations of the techniques, such as:

- high accuracy in detecting network states and ease of analyzing noisy network data at the price of having a reduced diversity of types of disruptions possible to detect;
- extensive in-depth analysis methodology of the IS-IS failure data; however such information is not commonly offered by ISPs
- the use of operators' readily available data sources, therefore applicability on a variety of IP networks; the downside is that these sources are private to the network operators and consequently not suitable for the analysis of failures over the Internet.

2.3 Conclusion

In this chapter two different approaches on improving network performance have been presented: delay analysis and network troubleshooting.

They are essentially different given the input information and the analysis stages, but they both have as goal a better network performance either by understanding delay behavior and optimizing the network based on it, or by tracking down the causes of network instabilities. Thus they use different information on the network for different level purposes.

Dragon-Lab framework represents a hybrid of these two approaches in the sense that the framework is based on network end-to-end delay measurements, as delay analysis methods, and low level routing information, in essence traceroute data for the measured Internet paths, to be able to infer the causes of abnormal delay variations, namely network instabilities, such as the network troubleshooting methods.

There are several significant differences between Dragon-Lab and the presented techniques:

- Dragon-Lab uses delay measurements to infer network states as opposed to delay analysis which increases network performance by further delay analysis;
- Dragon-Lab uses low quality publicly available routing information combined with delay measurements to infer network instabilities; network troubleshooting techniques regularly use high quality private network information.

The methodology of Dragon-Lab framework will be presented in the following chapter; its capabilities will be studied throughout the rest of the report.

Chapter 3

The Dragon-Lab Framework

The content of this chapter introduces the technical foundation and methodology of a new network troubleshooting technique: Dragon-Lab framework [9]. The framework is based on a new approach on network delay analysis, namely the one introduced in the paper: A Structural Analysis of Network Delay [10].

Dragon-lab is a network troubleshooting framework able to diagnose the Internet backbone states using two end-to-end metrics: packet delay and packet loss and routing information from traceroute data to spatially localize instabilities: link failures, congestions and queues building up. The analysis consists of detecting and identifying significant unusual changes in the mentioned metrics. In this way, there is no need for information regarding the structure, operations or events in the backbone from the ISPs side.

Based on the analysis of these types of instabilities statistical models of how instabilities occur on the Internet can be built. This may be particularly important for network design and traffic engineering or for being able to guarantee certain end-to-end QoS parameters for real-time applications or for VPNs. This capability of Dragon-Lab will be examined thoroughly in Chapter 4.

The existing network analysis approaches focus on the analysis of link failures harvesting a wide range of network information such as BGP routing tables, IS-IS routing messages, router logs, router configurations, SNMP MIBs, administrator files, syslog messages. They have been presented in Chapter 2. These approaches are effective and relatively accurate; however they are mostly suitable for small or medium scale networks, since, on the Internet, they pose scalability issues. Moreover, Internet instability problems are not only caused by link failures. Congestions and queues have a significant effect on the Internet applications' proper operation. Dragon-Lab data provides the support to analyze also these two types of problems, as well as confirming the results of previous studies regarding link failures behavior.

Temporal localization of Dragon-Lab is accompanied by a geo-localization of the instabilities. This feature of the framework focuses on using Dragon-Lab available data in order to locate the source of instabilities on the Internet paths, in essence to find the problem hop on the path by IP. This is achieved by combining algorithms from the Compressive Sensing field with Dragon-Lab knowledge on instabilities as it will be seen in Chapter 5.

3.1 Overview

Dragon-Lab is an automated framework for network diagnosis. The goal of the framework is to infer Internet backbone network states. These are the following: stable state, congestion, queuing building up, link failure. The diagnosis consists of detecting and identifying any significant unusual change in the measured network features: delay and loss. The study data is based on active and passive measurements taken in the network shown in figure 3.1. It basically consists of three end systems situated in Norway (NTE), China (CERNET) and New Zealand (AUCKLA). Probe packets were sent every 10ms in both directions making this a rare, high "granularity" measurement. In order to guarantee analysis tractability, aggregation was applied to the raw measurement data.

The framework is divided into three main components: aggregation into time bins/averaging of delay over long term measurement, analysis of obtained data using sparse and low rank matrix decomposition and projection of the refined anomalies on the feature space. A clear image of the Dragon-Lab logical flow can be seen in figure 3.2. After PCA based detection the anomalies are refined, basically filtered, based on the knowledge about the network. Afterwards, using decision metrics denoted as "backbone features" they are classified into the network states, which were already defined. Identified anomalies are then confirmed by taking snapshots of the time bins where the alarms were triggered.

3.2 Methodology

It will be proven that the analysis of network delay and of packet loss can result in detection and identification of the network states. The intuition behind this is that irregularities in a large scale network have direct impact on packet delay and on packet loss. For example, if there is an increase in the delay and an increase in loss then the network may be congested. On the other hand, if there is only an increase in delay there might be that there is queuing building up in the networks router buffers involved. A link failure is represented by a high decrease in delay, meaning that the packet does not reach the destination, and high increase in loss.

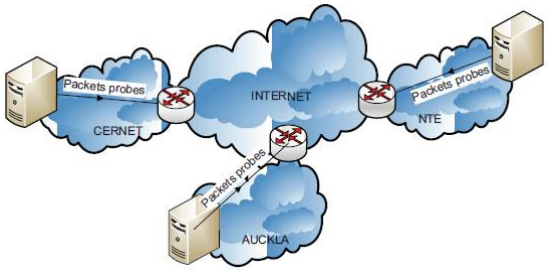


Fig. 3.1 Topology overview [9]

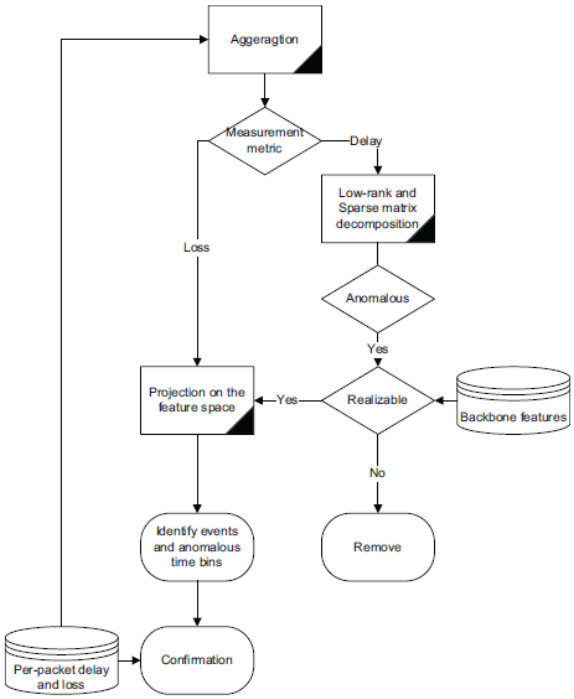


Fig. 3.2 Dragon-Lab overview [9]

3.2.1 Aggregation Strategy

Raw data is not tractable for our analysis, therefore aggregation will be used. Thus the classical approach of per-packet delay analysis is not employed in this framework, but aggregation during a certain time interval and analysis over longer periods of time, namely per day, are being used. Aggregation is practically the summation of per-packet delays over the particular time interval, time bin. Averaging, derivation of the temporal mean of the aggregated measurement is also being used. The effect of aggregation is smoothening of the fast fluctuations of per-packet delay measurements. It can be observed in the figure below:

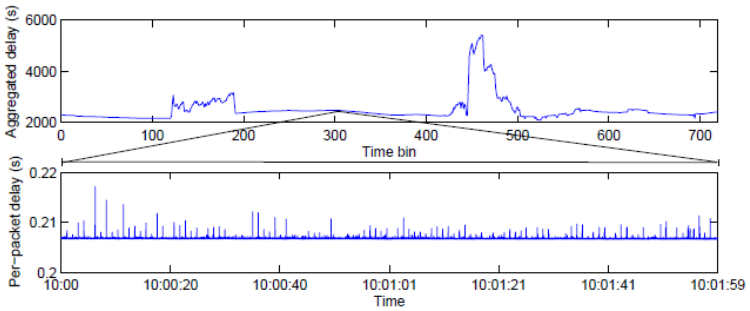


Fig. 3.3 Aggregation effect [9]

In order to find the best fit time bin, a study of the duration of each of the previously mentioned network states has been done.

In order to determine the duration of a link failure, events called downtimes are studied. They are actually outage time periods that can last from milliseconds to seconds when all transmitted packets are lost. Short downtimes are due to congestions while longer ones are due to link failures. Figure 3.4 presents the cumulative distribution function of the duration of downtimes longer than 1s. It is the result of a study conducted over on the same four Internet links as the ones analyzed in this paper, for three months of measurements. It is shown that more than 90% of the downtimes have duration of 120s, and the maximum duration of a downtime is 360s.

Queuing building up duration is directly dependent on the length of the Internet routers buffers and link loads. There is no standard buffer length, but it is known that the aim is to be short, not exceeding several hundred packets. A longer buffer would determine frequent queues building up, while a shorter buffer would cause transition to congestions. Therefore, the duration of a queue may vary from several seconds to minutes.

Congestion duration varies from several seconds to several minutes. It depends on the link capacity and on the traffic load of the link. Congestions depend on the internet paths physical behavior.

As a consequence of the above facts the duration of aggregation time intervals has been empirically chosen 2 minutes, as a trade-off between accuracy and scalability; longer time bin would smoothen too much aggregation delay leasing to non-detection, shorter time bin would increase the dimensions of the data. Thus for each day there will be 720 time bins. The size of the new data, the aggregate delay, is reduced with a ratio of over 15000 to one. As shown in figure 3.3 the effect of aggregation is also the smoothening of the per-packet variation and the fact that the similarity to the per-packet delay behavior seems to be lost. Moreover it is observed that on a three weeks measurement period the aggregated delay plot shows periodicity (figure 3.5). This fact could have not been observed without aggregation.

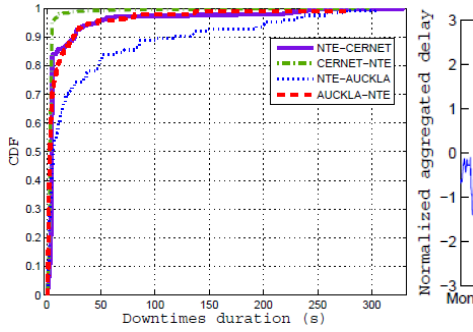


Fig. 3.4 CDF of downtimes exceeding 1s [9]

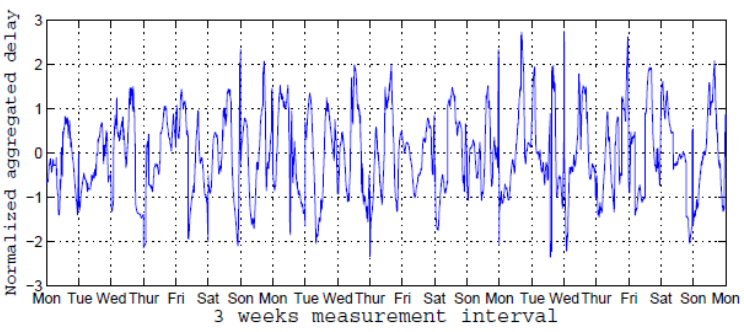


Fig. 3.5 Periodic behavior of aggregated delay [9]

3.2.2 Network State Transitions

The choice of the aggregation strategy, summation or average, has great importance on network state investigation. Consequently the following features are proposed for inferring network states:

- Aggregate packet loss
- Aggregate delay
- Average delay

In order to give physical sense of the chosen features consider the examples: a link failure can be characterized be high decrease in aggregated delay, high increase in loss and no noticeable change in average delay; the beginning of a queue building up might be characterized by increase in aggregated delay and average delay but no loss.

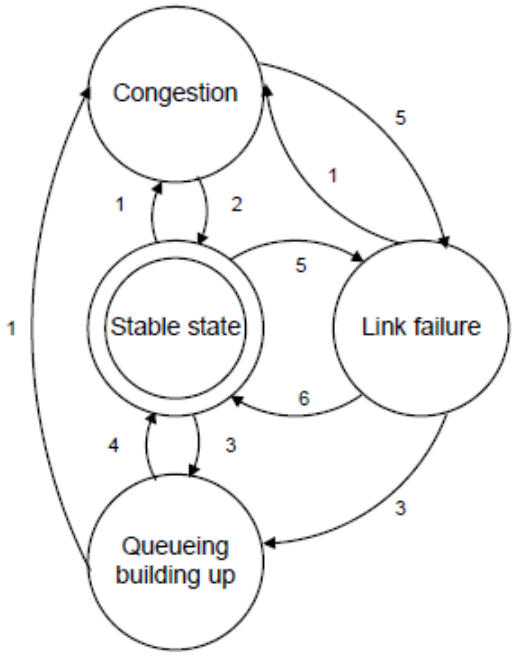


Fig. 3.6 Transition diagram [9]

The possible transitions between network states considered in Dragon-Lab, are the ones illustrated in figure 3.6. The changes in network features involved in the state transitions are summarized in table 3.1. The number on the arrows in figure 3.6 represents the transition ID and it is given in the table. As observed from the figure most of the transitions from one state to the other are possible, with the exception of queuing building up to link failure and from congestion to queuing building up.

Link failures for example start and end with high losses and a decrease in aggregate delay and most of them are of short duration, within one time bins. A link failure may end up with either a stable state or with congestion, if the link becomes overloaded after recovery or just with a queue build up.

A queuing building up detection is triggered by an increase in aggregated delay and in average delay, but no loss. However, a queue might transform into congestion if the traffic continues to increase, router buffering fails and losses occur. Similar accumulation may lead from congestion to link failure.

(Transition, ID)	Loss	Aggregated delay	Average delay
(Start congestion, 1)	High	Increase	Increase
(End congestion, 2)	High	Decrease	Decrease
(Start queuing building up, 3)	0	Increase	Increase
(End queuing building up, 4)	0	Decrease	Decrease
(Start link failure, 5) & (End link failure, 6)	High	Decrease	Any

Table 3.1 Feature based backbone state transition

Measurement data was collected using the topology presented in the overview. Probes packets were sent every 10ms. Measurements lasted for three months. The end systems were NTP synchronized and located close to the Internet backbone, which results in negligible network access delay and loss. The NTP clock server for NTE and CERNET was located in a third network in Norway – UNINETT. Therefore NTP synchronization's influence will have to be considered. For obtaining delay and loss data the RUDE/CRUDE tool was used. The access to packet headers, which hold information on the number of hops within a route, was possible using the Tcpdump tool.

3.2.4 PCP Analysis and Delay Decomposition

Aggregation makes it possible to emphasize to components of the delay, which are:

- An abrupt impulsive short component due to the abrupt variations in aggregated/average delay, called "abnormal" delay behavior.
- A smooth component which shows periodicity (figure 3.5), is a combination of the real variation in delay and the clock adjustments, and is called "normal" delay behavior.

A good representation of the decomposition of the two components is given in figure 3.7.
In order to extract the two types of behavior aggregated delay time series will be analyzed within a structure like the one in figure 3.8. Each line consists of 720 positions containing the value of the aggregated delay of the corresponding time bin. Based on the principal component analysis (PCA) applied on the delay space the normal component, which is a low rank matrix, will be extracted. The residual of PCA is the noisy abnormal sparse component. A sparse vector is a vector with few non-zero components. Even if PCA performs well on decomposing the aggregated delay space, it breaks down when there is high percentage of high variations of the delay space. This is explained in the following way: PCA searched the best rank-k estimate E of a data matrix X by solving:

$$\min \|X - E\|_2 \text{, subject to rank (E)} \leq k, \tag{3.1}$$

where $\| \quad \|_2$ is the l_2-norm.

The rank has to be known a priori. This is rarely occurs in real situations. Moreover the problem is a least square problem. It gives good results when Gaussian noise is involved, in essence low and medium abnormal delay. However, l_2 fitting is sensitive to for high abnormal delay leading to a low rank subspace perturbation phenomenon. Consequently there will be inaccurate detection and high false positive rates. [11][12]

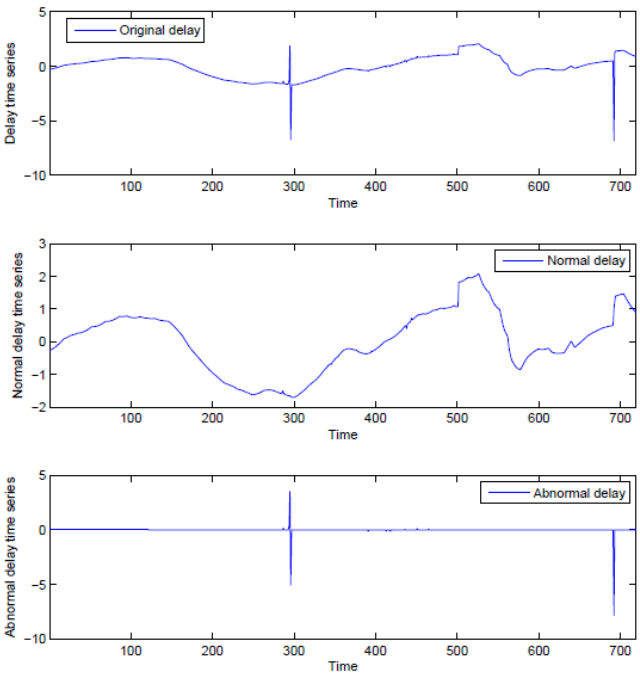


Fig. 3.7 Delay decomposition [9]

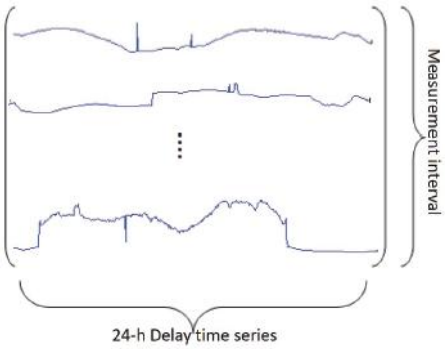


Fig. 3.8 Delay space [9]

In order to be able to detect the anomalous states the low rank normal component of the delay has to be extracted and the abnormal one has to the found. Transitions from the stable state to any other are given by this component. Intuitively this component is sparse, since the number of anomalous states is small compared to the number of stable states. The structure of the abnormal component vector is dependent on the anomaly involved. Practically if there is a queuing building up or a link failure, which are short duration instabilities, the abnormal component will have a bursty behavior, while for congestion will produce blocks of non-zero abnormal vector components. The sparsity is guaranteed, since the number of instabilities compared to the number of stable states is within the limits: [0.01, 0.06].

To overcome PCA's limitations robust delay decomposition is used. This takes advantage of the fact that additional to the low rank of the normal component, the abnormal component is sparse. Up until this point the following problem has to be solved:

$$min_{N,A} \|A\|_0 \text{ , subject to } X = N + A, \text{ rank } (N) \leq k, \tag{3.2}$$

where $\| \quad \|_0$ is the l_0-norm: the cardinality of non-zero components.

This optimization problem is NP-hard. However, studies [12] proved that the low rank component can be recovered using the nuclear norm (sum of singular values) and the sparse component can be recovered by using the l_1 norm (sum of absolute non-zero element values), thus the previous problem can be solved using the Principal Component Pursuit [12] defined as:

$$min_{N,A} \|N\|_* + \lambda\|A\|_1 \text{ , subject to D = N + A,} \quad (3.3)$$

where D of n_1 X n_2 is the measured network space, $\| \ \|_*$ is the nuclear of the normal delay matrix N , $\| \ \|_1$ is the l_1 norm of the anomalous events matrix A, and $\lambda > 0$ is a weighting parameter.

In Dragon-Lab framework for this problem a solver employing the inexact version of the Augmented Lagrange Multiplier (IALM) [13] is used. The PCP problem is reformulated as:

$$F(N, A, Y, \mu) = \|N\|_* + \lambda\|A\|_1 + < Y, D - N - A > + \mu/2 \|D - A - N\|_F^2, \quad (3.4)$$

where Y and μ are Lagrange multipliers, <,> is the inner product and $\| \ \|_F$ is the Frobenius norm.

For a given set of Lagrange multipliers (Y, μ), IALM, using the Singular Value Decomposition (SVD) technique, iteratively minimizes F with respect to A, keeping N fix, then it minimizes F with respect to N, keeping A fix. The Lagrange multipliers are also iteratively updated in order to reach convergence. A soft thresholding technique is being used, namely:

$$ST_e(x) = \begin{cases} x - e \text{ if } x > e, \\ x + e \text{ if } x < -e, \\ 0 \text{ otherwise.} \end{cases} \quad (3.5)$$

The IALM for low rank and sparse matrix recovery algorithm is the following:

Input: Delay measurement D, the regularization parameter λ

Output: Normal N, Abnormal A delay behavior

1: $A_0 = 0$, $Y_0 = D / (\max(\|D\|_2, \|D\|_\infty))$, $\mu_0 = 0$, $\rho > 1$, $k = 0$

2: while **(**not converged) do

3: $(U, S, V) \leftarrow \text{svd}(D - A_k + \mu_k^{-1} Y_k)$

4: $N_{k+1} \leftarrow U*ST_{\mu_k^{-1}}(S)*V^T$

5: $A_{k+1} \leftarrow ST_{\mu_k^{-1}}(D - N_k + \mu_k^{-1} Y_k)$

6: $Y_{k+1} \Leftarrow Y_k + \mu_k(D - A_{k+1} - N_{k+1})$

7: $\mu_{k+1} \Leftarrow \rho\mu_k$

8: $k \Leftarrow k + 1$

9: end while

By making use of IALM in PCP and of PCP in the Dragon-Lab framework, its functioning, presented in figure 3.2, will work according to the following algorithm:

Input: Per-packet delay D, loss events L

Output: Anomaly A ϵ {Congestion, Queuing, Link Failure}

1: while (not converged) do
2: $D_{aggregated}$, $D_{average}$ ← aggregate (D)
3: $L_{aggregated}$ ← aggregate (L)
4: $D_{anomalous}$ ← PCP ($D_{aggregated}$, λ)
5: Check Backbone Features
6: if $D_{anomalous}$ ϵ realizable, then
7: A ← Project on feature space ($D_{ano.}$, $L_{ano.}$)
8: end if
9: end while

Based on PCP results of the delay analysis and on the backbone features the identification of the different network states is done as in the following section.

3.2.5 Network States Identification

Internet backbone states can be identified by the following features' behavior: packets losses, packets aggregated delay, packets average delay. Therefore a PCP detected anomaly will lie within the space Γ of basis: $\gamma = \{ \overrightarrow{\text{Losses}}, \overrightarrow{\text{Agg.delay}}, \overrightarrow{\text{Avg.delay}} \}$, thus agg.delay and avg.delay are resulted after PCP analysis of measured aggregated and average delay respectively.

This space can be further decomposed using the definitions in table 3.1 into three subspaces correspondent to each type of anomaly. The classification methodology is as follows:

- Congestions will be localized in the subspace:
  $\Sigma_1 = \{(x, y, z); x > 0, y > 0, z > 0\}$. Congestions are defined regardless of their duration.
- Queuing building up will be localized in the subspace:
  $\Sigma_2 = \{(x, y, z); x = 0, y > 0, z > 0\}$. Queuing building up is characterized by high aggregated and average delay, but no losses.
- Link failures will be localized in the subspace:
  $\Sigma_3 = \{(x, y, z); x > 0, y < 0\}$. Link failures are characterized by decrease in aggregated delay and high losses.

Based on the backbone states definitions described in table 3.1, every detected anomaly in Γ should be included in the space of realizable network states: $\Delta = \{ \cup \Sigma_i, i = 1..3 \}$. Detected instabilities that do not fit this space are filtered from the detected problems on the reason that

either it is physically impossible for them to exist, or they fit the conditions but are actually false detections.

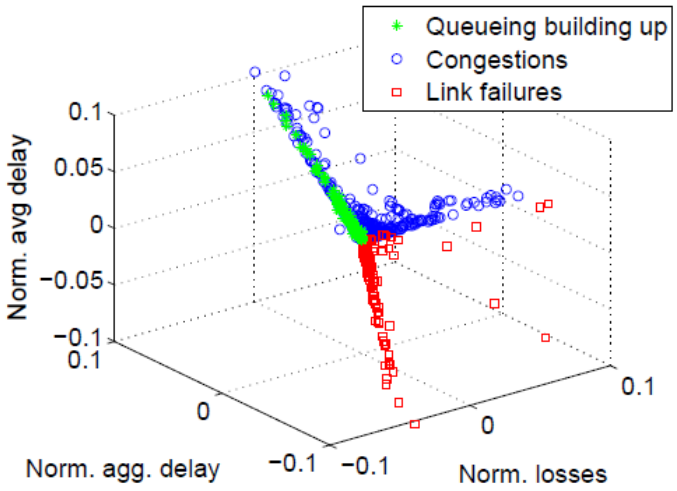


Fig. 3.9 Γ space projection of the NTE-CERNET states

The instabilities detected on the NTE – CERNET path during the 82 days of measurement projected on the Γ space can be seen in figure 3.9. Each of the point represents a detected anomaly. It can be observed that similar types of instabilities cluster. Each of the dimensions has different scales since the values have been normalized to show the clustering effect. It is shown that different instabilities are clearly separated in the Γ space. Most of the detected instabilities are close to the center of the three axes, thus close to zero in value, which suggests that the instabilities have low intensity, making Dragon-Lab framework even more valuable since it detects low intensity instabilities.

3.3 Conclusion

The basic features on Dragon-Lab, namely detection and identification of instabilities, introduce several improvements compared to previous techniques:

- it solves the problem of NTP synchronization present in delay time series;
- it is using publicly available data such as packet delay and packet loss for network analysis;
- it is capable of detecting various network states, not only link failures, based on the analysis of the previously mentioned data.

In the next chapters, 4, 5 and 6, the performance of detection and classification of network states of Dragon-Lab will be investigated , and furthermore these basic features will be extended into a complete Internet paths behavior analysis, and moreover into a spacial localization capability.

Chapter 4

Dragon-Lab Performance Evaluation

The value of any tecqnique lies in its performance. Therefore, in this chapter the performance of Dragon-Lab will be assessed. The data used in performance evaluation was collected using the scenario and methodology presented in the previous chapter.

To quickly summarize, the measured and afterwards processed data to analyze consists of three backbone features, per time bin:

- Aggregated delay: summed per-packet delays over every 2 minutes (one time bin).
- Average delay: summed per-packet delays divided by the number of packets over every 2 minutes.
- Losses: number of lost packets summed over every 2 min.

The measurement is done on the following paths over the Internet:

- Norway – China; paths: NTE – Cernet and Cernet NTE;
- Norway – New Zealand; paths: NTE – Auckland and Auckland – NTE.

For each of the mentioned links the measurement extends over for 82 days; each day consists of 720 time bins. The data will be organized into matrices of size 82 X 720 for each of the three backbone features.

The performance analysis is aimed to answer the following questions:

- How well can Dragon-Lab detect actual network states observed in real data?
- How well can Dragon-Lab classify actual network states observed in real data?

In order to answer these questions the following steps will be taken: firstly, the analysis of the measured data and the PCP output data and the isolation of the real detected instabilities, practically by removing the falsely detected ones. In this way, general detection rates and false positive rates will be obtained. Secondly, through the splitting of the detected anomalies into each of the three types of anomalous network states, the same procedure will be applied to the detected false anomalies and various rates per type of instability will be computed.

4.1 Detection Performance

Using PCP analysis the delay space is decomposed into the normal component and abnormal component; the abnormal component triggers the alarms for unstable states detection. However, this first detection may not be completely true. That means that among the detected anomalous states one may find alarms that should not be there – false alarms. In order to separate false alarms from the real ones a visual inspection of the data has been carried out. Practically, aggregate delay time series, PCP processed aggregate delay, average delay and losses data have been simultaneously visually analyzed to distinct between real anomalies and false ones, for all the four measured Internet paths. The visual inspection had also had to detect possible non detected PCP abnormalities. By fulfilling the visual inspection real instabilities were isolated. Detection rates have been calculated as the percentage of the ratio between number of detected anomalies and the number of detected anomalies together with the number of non detected ones. False positive rates have been computed as the percentage of the ratio between the number of false alarms over the number of detected unstable states.

4.1.1 PCP Parameter Tuning

As shown in the presentation of the PCP algorithm in the previous section, it has one tuning parameter λ, which may favor either the extraction of the normal component or of the abnormal one from the measured data. Thus PCP can be more or less sensitive to low variations in the delay data. If for example λ increases, PCP is less sensitive to low variations. Consequently, the false positive rate decreases, but the number of non detected instabilities increases, determining the detection rate to decrease. Therefore, λ has to be chosen rationally so as not to bias the decomposition. The authors of [13] proposed a scaling factor in the order of $O\,(\,1\,/\,\sqrt{max\,(\,n_1, n_2)}\,)$, where n_1, n_2 are the dimensions of the delay space.

All these considered, the variation of the detection rate and of the false positive rate with the tuning parameter λ on the four links involved in the measurements in the Dragon-Lab case, can be seen in figure 4.1.

It can be observed that performance differs from one path to another. Even between the same end systems, due to internet path asymmetry, detection rates differ. For example in the case of Norway – China paths, for λ = 3 * Scaling_factor: on the Cernet-NTE direction detection rate is 88%, while for NTE-Cernet direction it is only 64%. λ can be chosen according to the application for best results. From figure 4.1 it can be seen that for λ = 2.5 * Scaling_factor detection rates for NTE – Cernet and Cernet – NTE paths are 80% and 94% and the false positive rates are relatively low: 6% and 14%. Ratios are similar on the other two paths.

Therefore, it has been decided that for this particular scenario and the links involved the best trade-off high detection rate – low false positive rate is obtained using λ = 2.5 * Scaling_factor.

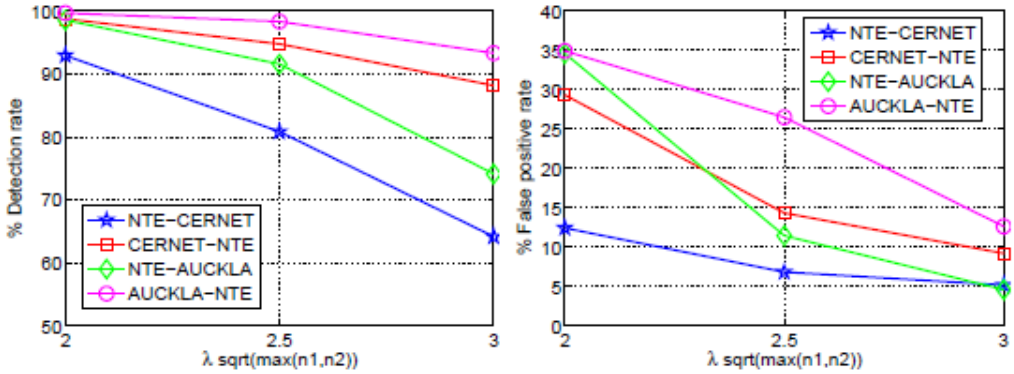


Fig. 4.1 General Dragon-Lab performance

The dependence between the detection rate and the false positive rate results in a curve called Receiver Operator Characteristic (ROC). The ROC curves for the four paths are shown in figure 4.2. Clearly, Dragon-Lab performs similar on all of them. Additionally the curves show once more the importance of adjusting λ in order to achieve a desired detection rate versus false positive rate. Moreover, the curves show that a detection rate of around 90% induces about 10% false positives for the four paths. All this is achievable on the cost of tuning λ.

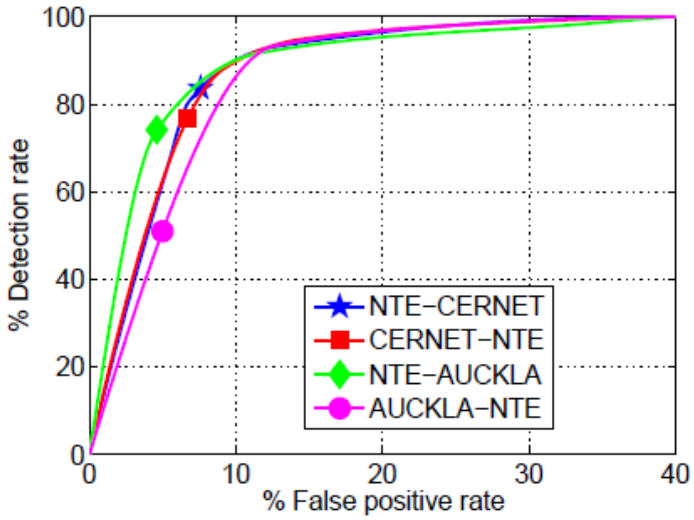


Fig. 4.2 Dragon-Lab ROC curves

4.1.2 PCP Effectiveness

In order to show PCP effectiveness a few snapshots that include common situations encountered along the analysis will be given, for the value λ = 2.5 * Scaling_factor. Figures

4.3, 4.4 and 4.5 include situations from paths NTE – AUCKLA, NTE – Cernet and Cernet NTE, for measured aggregated delay (left side) and their PCP analysis result (right side), for one day of measurements. The moments when instabilities are known to occur are marked with circles. Figure 4.3 shows one day of aggregated delay measurement with the corresponding PCP result. Clearly there is high oscillation of packet delay, combined with NTP synchronization delay. The overall variation of the aggregated delay, the magnitude, is not high, about 70s, and is dominated by the clock effect. Considering the high level of perturbation, PCP functions relatively good, eliminating the clock effect and detecting the medium and high variations. Figure 4.4 presents a fortunate case: perfect detection and no false positives. In this particular case the biggest variation of the aggregate delay time series is almost 1000s. However PCP is able to separate even the smallest abrupt variations. The trend of the aggregated delay curve is typical for the NTE – Cernet path. Figure 4.5 shows one day of measurements on Cernet – NTE path: just one abrupt variation in aggregated delay, detected by PCP. However the other smooth variations, due to NTP synchronization are also considered as anomalies. This situation is however solved within Dragon-Lab by applying efficient filtering outside the possible anomalies space (presented in the previous chapter).

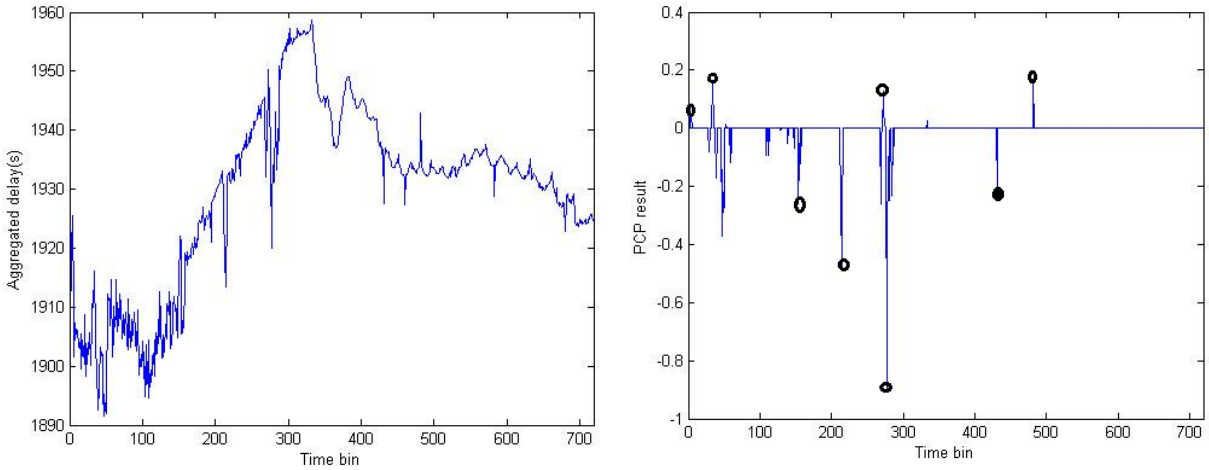


Fig. 4.3 PCP effectiveness: NTE – AUCKLA

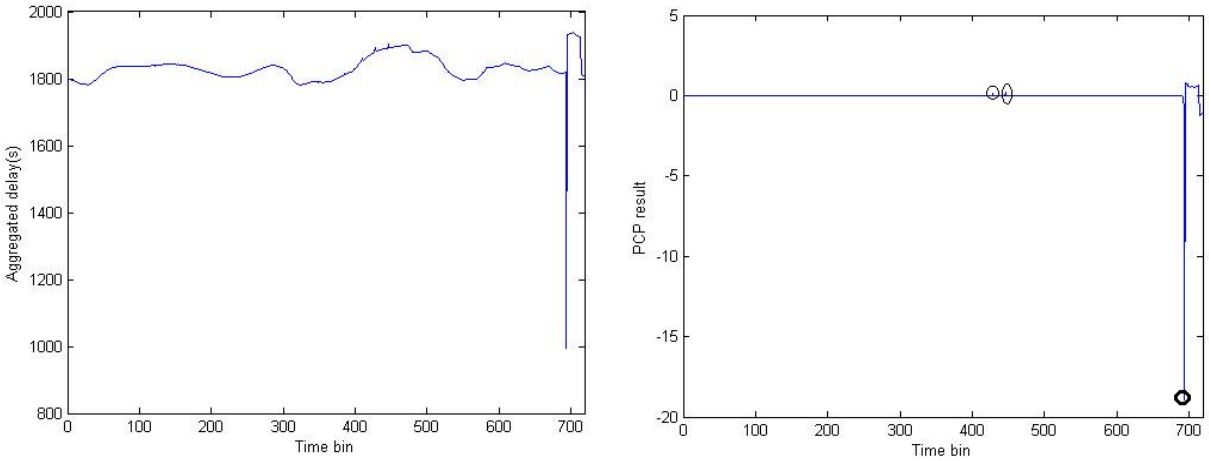


Fig. 4.4 PCP effectiveness: NTE – Cernet

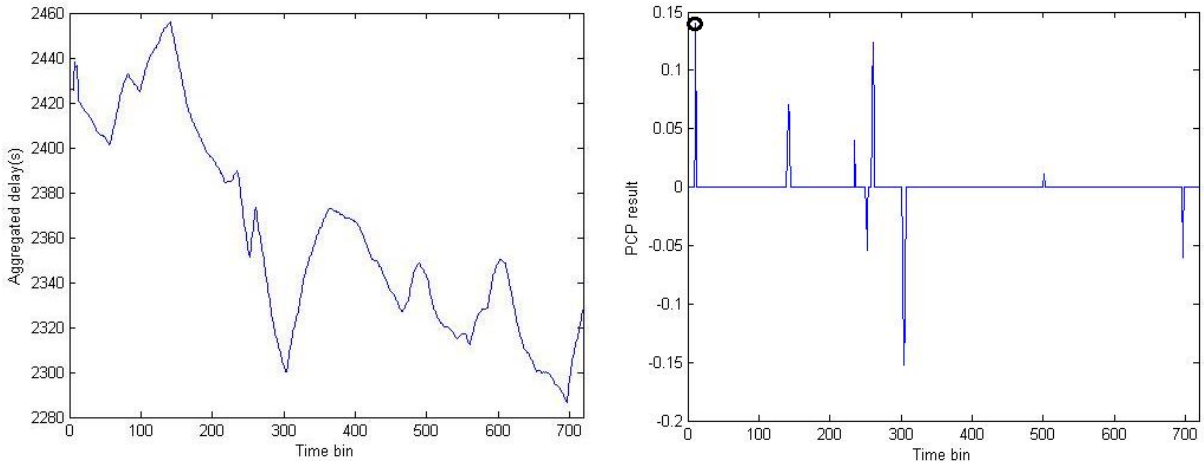


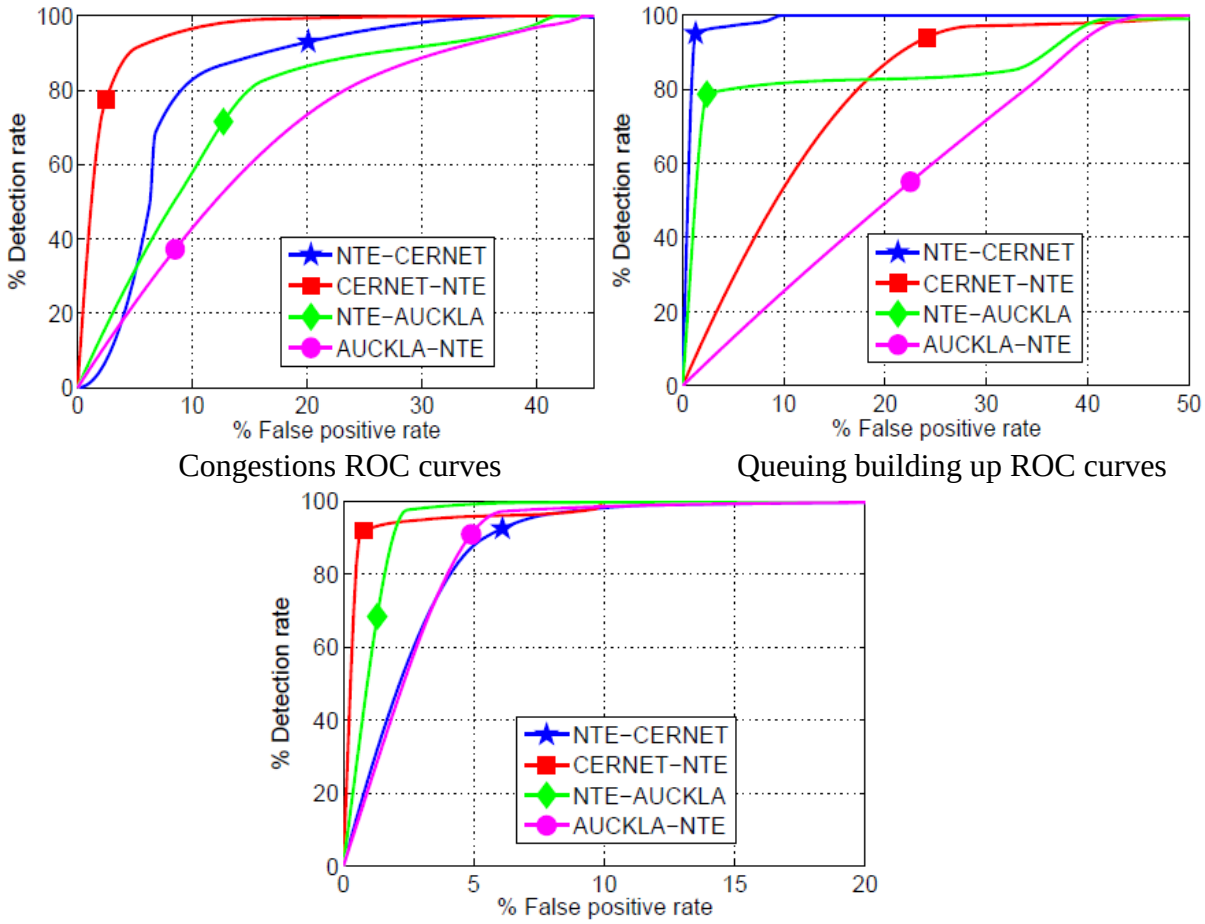
Fig. 4.5 PCP effectiveness: Cernet – NTE

4.2 Classification Performance

Based on the set of "true" unstable states isolated from the total number of PCP detected instabilities, obtained by visual inspection of both aggregate and average delay time series, detected and false positive rates of Dragon-Lab identified network states are studied. The ROC curves per type of anomaly are obtained and a summary of the classification results over the four measured paths is given.

4.2.1 ROC Curves

The ROC curves per type of instability for the four paths are given in figure 4.6. Notice that the x scale differs from one case to another. It is easily noticeable that Dragon-Lab identifies best link failures. They have the lowest false positive rate with respect to detection rate of all links. This is natural because link failures cause high perturbation in the aggregated delay, which will easily be detected by PCP. Regarding congestions, they are slightly worse identified by Dragon-Lab than link failures. Congestion classification is based on all the three defined backbone features: aggregated delay, average delay and packet loss. However, congestions do not necessarily create outliers. Therefore, NTP synchronization or simply fluctuations in the aggregated delay may induce false congestion alarms. Dragon-Lab has the worst performance in identification of anomalies in the case of queuing building up; they present the highest rates of false positives. This is explained by the fact that the identification of this type of anomaly is based only on PCP performance for abrupt variations detection in delay time series.



Congestions ROC curves

Queuing building up ROC curves

Link failures ROC curves

Fig. 4.6 ROC curves per type of network state

4.3 Conclusion

Dragon-Lab is able to detect network unstable states with a high detection rate, over 90%, at a cost of around 10% false positives rate. The identification performance depends in a high degree on the specific type of network state.

A downside is that Dragon-Lab's performance depends much of the PCP tuning parameter which has to be carefully chosen in order to achieve the best results. Numerical values of all the rates used in this chapter can be found in the Appendix.

Chapter 5

Dragon-Lab Internet Paths States Analysis

5.1 Introduction

The novelty of Dragon-lab framework is that it is able to temporarily identify three Internet backbone unstable states in the by end-to-end measurements of delay and packet loss. These states, as defined in Chapter 3, are: link failures, congestions and queues building up. The measurement methodology has been described in the same chapter.

Based on the analysis of these types of instabilities statistical models of how instabilities occur on the Internet can be built. This may be particularly important for network design and traffic engineering or for being able to guarantee certain end-to-end QoS parameters for real-time applications or for VPNs.

Most of the existing approaches for network analysis focus on the analysis of link failures harvesting a wide arrange of network information such as BGP routing tables, IS-IS routing messages, router logs, router configurations, SNMP MIBs, administrator files, syslog messages. They have been presented in Chapter 2. These approaches are effective and relatively accurate; however they are mostly suitable for small or medium scale networks, since, on the Internet, they pose scalability issues. Moreover, Internet instability problems are not only caused by link failures. Congestions and queues have a significant effect on the Internet applications' good operation. Dragon-Lab data provides the support to analyze also these two types of problems, as well as confirming the results of previous studies regarding link failures behavior.

Having the necessary information of the path operating state has crucial importance, especially for real-time end-to-end applications. In this way applications can be guaranteed a minimum quality of network requirement for a proper functioning. Congestions and queues are not as harmful as link failures, but, as it will be seen, they represent a significant percentage of all disruptions and consequently have to be taken into account.

This chapter is a proof of the Dragon-Lab temporal instabilities identification function; it provides data for an analysis of four Internet paths. Together with the following chapter, which handles the spacial localization of the instabilities, they increase the added value of the

Dragon-Lab framework. The chapter comes first with a methodology of the analysis, presents the results of analysis depending on the type of unstable state and then offers a general analysis of instabilities regardless of type.

5.2 Methodology

In this chapter the results of Dragon-Lab temporal detection and identification are used. The chapter tackles the following issues:

- how long do the instabilities last;
- how often do instabilities occur;
- how similar are the paths with respect to instabilities behavior;
- how do the instabilities impact the general functioning of the network.

The previous questions are answered for all the four measured Internet paths between Norway, China and New Zealand; the analysis is divided into two parts:

1. An analysis of unstable Internet backbone states behavior. This part gives an image of how each of the predefined unstable states, link failures, congestions and queues building-up, separately, respond to the problems stated above. The practical solution to this is, for each type of unstable state and for each of the fours paths to:
   - plot CDFs of the duration of instabilities;
   - plot CDFs of the time between instabilities;
   - determine the percentage of instabilities distributed on each path;
   - determine particularities of the instabilities, such as dispersed/frequent instabilities, or of long/short duration.

When possible, CDFs have been approximated with well-know distributions for later modeling use. The distributions have been approximated using the Maximum Likelihood Estimation method.

2. An analysis of instabilities whatever the type, in order to give a general overview of the degree the paths are affected by instabilities. Additionally, three metrics are analyzed, to characterize the state of the network: availability, fatigue and stability. They will provide a quantitative measure of the impact of instabilities on the network proper functioning.

5.3 Unstable States Statistical Analysis

Within this section each of the different types of unstable Internet backbone states will be analyzed separately.

First a statistical analysis of the durations of instabilities is provided, secondly an analysis of the frequency distribution instabilities, thirdly a distribution of the number of failures per path is presented and lastly instabilities are classified based on their dispersion level and duration.

5.3.1 Link Failures Analysis

Link failures are an intensively studied topic. Several studies have been conducted on link failures analysis; some of the most representative are presented in Chapter 2. However this analysis comes with a different perspective on link failures analysis, for the simple reason that this analysis is based on end-to-end Internet measurements, for paths that include several autonomous systems (ASs). All the other studies are centered on small or medium scale networks and make use of different, higher level, input information for their studies. Moreover, this analysis combines the magnitude of the studied network, the Internet, with the long duration of measurements, three months. These two facts give the analysis offers a trusted characterization of link failures behavior based on Internet end-to-end paths measurements.

Throughout this sub section, in order to generate the plots, the same sets of Dragon-Lab signaled link failures have been used; clearly they are unique for each of the four analyzed Internet paths, since link failures occur at different time instances and vary different durations.

To start with, the empirical CDF of link failures duration has been plotted in figure 5.1. It can be seen that this type of instabilities have duration of maximum 6 minutes, so they have relatively short duration. These types of link failures are possibly caused by software problem, transient equipment problems, routers CPU overload, routers mistakenly considering an adjacency to be down when it is actually not etc. Most of the failures last less the one minute – a common bulk of about 70% for the four links. The paths with the highest percentage of low duration failures are the paths between Norway and New Zealand. The other two paths, between Norway and China, experience a higher percentage of longer link failures – around 20% for duration between 4 and 6 minutes and 5% for duration between 4 and 6 minutes.

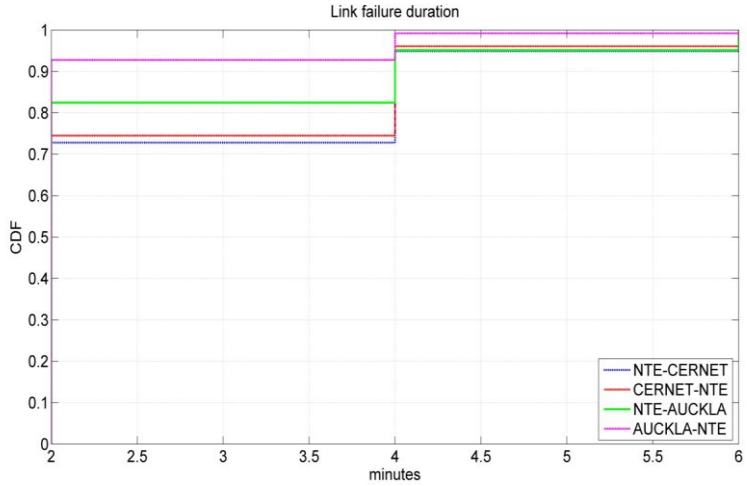


Fig. 5.1 CDF of link failure duration

Based on this result it can be stated that, on Internet paths, link failures are of relatively short duration; this might be due to better maintenance and performance of backbone equipment. Compared to access networks, the Internet backbone is much more reliable. Previous studies, [7], [4], [5], have shown that the Sprint network backbone has 60% of the failures up to 6 minutes and the CENIC network about 70% of the failures up to 6 minutes.

A possible explanation for the long lasting link failures, the ones between 2 and 6 minutes, can be the longer convergence time of inter-domain routing protocols [16]. This has been proven by conducting a visual inspection of the traceroutes data, by checking the IPs with anomalous delay behavior, at the moments of time where Dragon-Lab detected medium (2 to 4 minutes) and long (4 to 6 minutes) duration link failures. The results show that most of the long duration failures are due to inter-domain routing convergence, while medium duration failures are due to intra-domain routing.

For the purpose of future application, the CDFs have been fitted with well known distributions. Weibull distribution is the one that fits best the link failures durations distributions. The distributions parameters vary depending on the link; they are shown in the table:

Path	Distribution	α	β
NTE-CERNET	Weibull	2.993	2.445
CERNET-NTE	Weibull	2.923	2.511
NTE-AUCKLA	Weibull	2.764	2.402
AUCKLA-NTE	Weibull	2.382	3.098

Table 5.1 Link failures duration CDF - distribution parameters

The next step is the analysis of the time between link failures. For this purpose the CDFs of the time between link failures have been plotted in figure 5.2. It can be noticed that up until an inter-link failures interval of 30 minutes the paths NTE - CERNET and NTE – AUCKLA and

the ones between CERNET – NTE and AUCKLA – NTE, have similar cumulative distribution functions. It is also observed that link failures on the paths CERNET – NTE and AUCKLA – NTE are more dispersed than the other two, since about 20% of the failures are less than 30 minutes apart, where for the other two, 40% are 30 minutes apart.

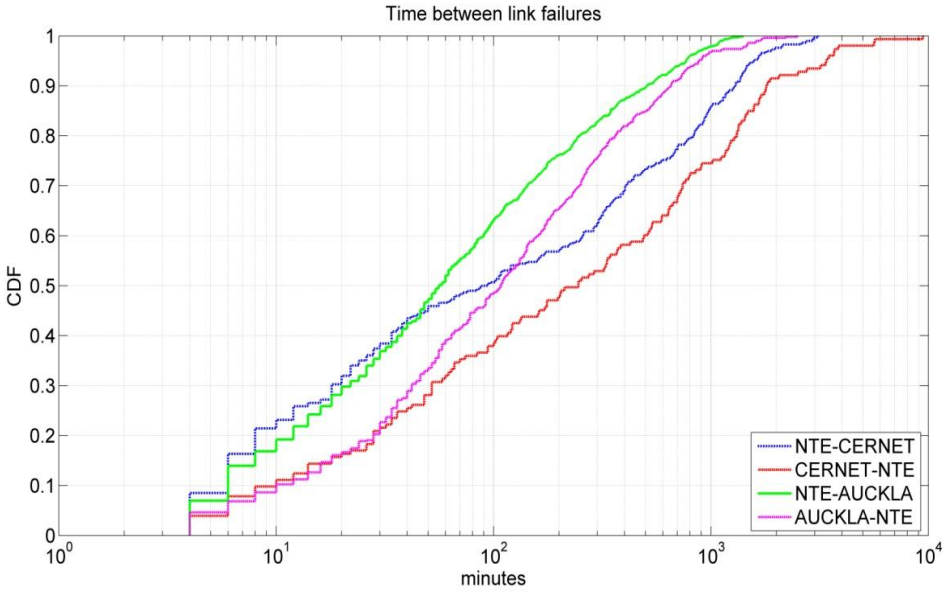


Fig. 5.2 CDF of time between link failures

Data fitting for these four cumulative distribution functions yield the results in table 5.2. The Weibull distribution fits well with the link failure CDF, since this distribution has been found applicable in reliability engineering to describe lifetime of components; additionally, it is derived as an extreme value distribution: for a large number of identical and independent components, the time to the first failure follows a Weibull [4]. The time between failures can be interpreted as the time to the first failure, assuming a renewal process. The cumulative distribution function of the Weibull distribution is given by:

$$F(x;\alpha,\beta) = 1 - e^{-\left(\frac{x}{\beta}\right)^{\alpha}}, \text{ for } x \geq 0 \quad \text{and} \quad F(x;\alpha,\beta) = 0, \text{ for } x < 0 \tag{5.1}$$

Path	Distribution	α	β
NTE-CERNET	Gamma	0.425	1884.08
CERNET-NTE	Weibull	972.1	0.585
NTE-AUCKLA	Weibull	245.6	0.680
AUCKLA-NTE	Weibull	391.6	0.749

Table 5.2 Time between link failures CDF - distribution parameters

Another possibility of analyzing link failures and their distribution is by dividing them into several classes based on:

- inter-link failure time;
- link failure duration.

The parameters for the two criteria and the names attributed to each category have been chosen as follows:

- 30 minutes as limit between what will be called frequent and dispersed link failures;
- link failures of maximum duration 2 minutes are named short link failures, for duration between 2 and 4 minutes are named medium link failures, and from 4 to 6 minutes are called long link failures.

Figure 5.3 shows the results of combination of the categories above for each of the four paths. On all paths the majority of link failures, minimum 70%, are of short duration. Among these minimum 45% are dispersed. However, a significant percentage, between 20% and 30%, is hold by the short frequent link failures, namely flapping failures. As proven by Markopoulou et al [4], link flapping is a predominant cause of instabilities; therefore, protocols and routing algorithms should be able to handle flapping implicitly. The lowest percentage for the analyzed types of link failures is for the long duration ones. Most of them, up to 5%, as included in the dispersed group; on the CERNET-NTE and AUCKLA-NTE paths, the frequent long link failures are not even present, which may infer the good quality of these two paths and generally of the Internet paths. This type of graph is in fact a combination of the link failures duration CDF, inter-failure time CDF and the threshold of 30 minutes between frequent and dispersed link failures.

In order to have a high level comparison of the paths' reliability characteristics, a distribution of the number of link failures per path, regardless of their duration or frequency, has been constructed. Figure 5.4 shows what percentage of the total number of detected link failures is hold by each path. Clearly the paths between Norway and New Zealand are less stable.

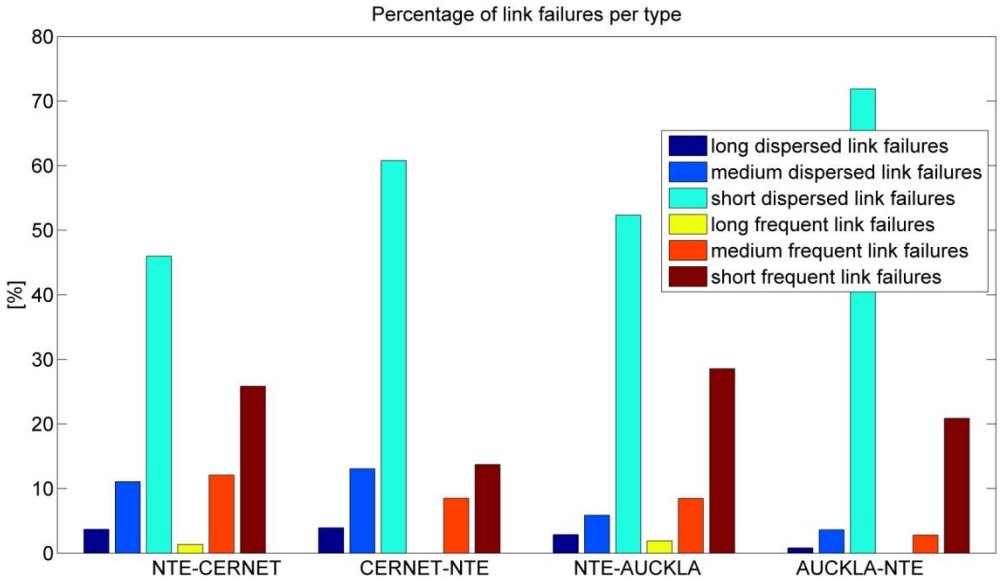


Fig. 5.3 Types of link failures – per path distribution

This result is difficult to explain because the systems are very complex, however what is certain is that these two paths traverse more autonomous systems than the paths between Norway and China. Still, up to a certain point the paths are common for all paths and this may account for the common characteristics. The NTE-AUCKLA path is the worst counting the percentage of link failures; moreover, this path contains the highest percentage of short frequent link failures, approximately 28%, which are considered the most disruptive type of link failures.

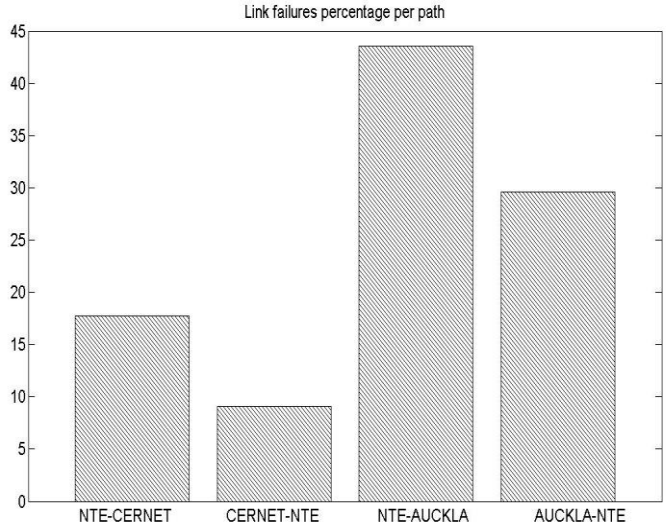


Fig. 5.4 Link failures percentage per path

5.3.2 Congestions Analysis

The analysis of congestions for the four measured Internet paths follows the same main lines as the analysis of link failures and as described in the methodology. It is important to understand the behavior of congestions since persistent congestions can degrade any type of application, especially the ones that require guaranteed throughput. [16]

Throughout this sub section, in order to generate the plots, the same sets of Dragon-Lab signaled congestion have been used; these four sets of detected congestion states are unique and fixed for each path.

To start with, the empirical CDF of congestions duration has been plotted in figure 5.5. Compared to link failures congestions' durations have a much more varied range of durations. They extend from 2 minutes up to 90 minutes. The path between Norway and New Zealand show 80% of the congestion length up to 2 minutes, while the paths between Norway and China show longer duration of congestions and only around 60% of all congestions have short

duration. The 2 minutes long congestions represent a significant amount on all paths, therefore the congestions will be divided depending on their duration into:

- transient congestions; for durations up to 2 minutes;
- persistent congestions; for longer durations.

The two types of congestions have different causes; transient congestions may be created due to the burstiness of IP traffic with short term fluctuations [16]; persistent congestions are mainly the cause of traffic overflow in highly utilized network [15]. The maximum durations of persistent congestions are 90 minutes on the AUCKLA – NTE path and 60 minutes on the CERNET – NTE path.

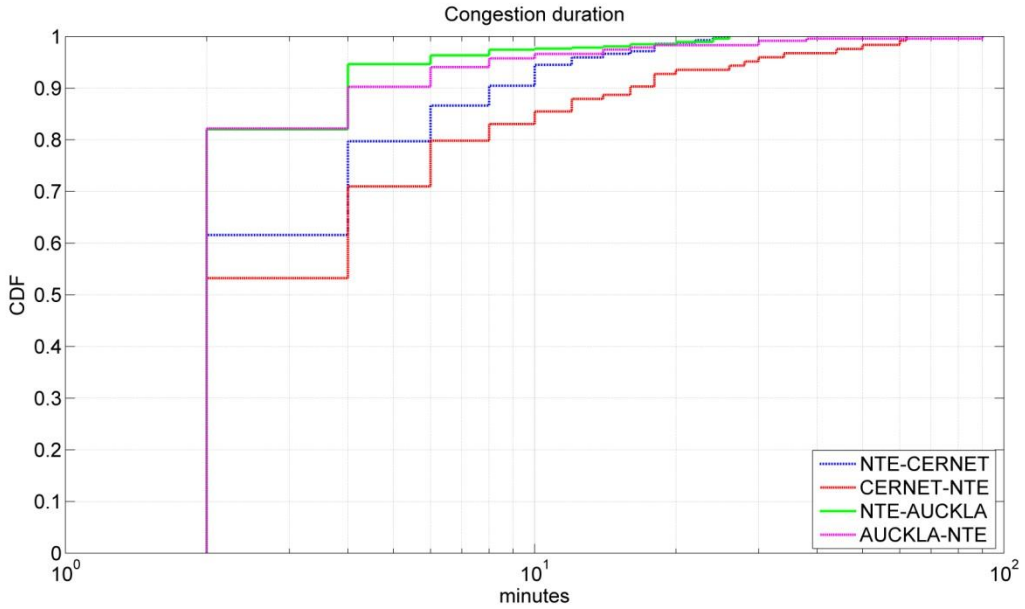


Fig. 5.5 CDF of congestion duration

Path	Distribution	μ
NTE-CERNET	Exponential	0.154
CERNET-NTE	Exponential	13.51
NTE-AUCKLA	Exponential	5.661
AUCKLA-NTE	Exponential	6.949

Table 5.3 Congestion duration CDF - distribution parameters

For the purpose of future simulations, fitting of the CDFs of congestion durations was performed. The results are shown in table 5.3. For all paths, the CDFs present exponential distributions which the parameters found in the table. The exponential distribution has the following cumulative distribution function:

$$F(x; \mu) = 1 - e^{-\mu x}, \text{ for } x \geq 0 \quad (5.2)$$

The exponential distribution fits well the CDF of congestion durations since this type of distributions are used to describe processes in which small occurrences are extremely

common, while large instances are rare [4]. The congestion durations fit this scenario, because most of the durations were within 2 minutes length, as shown above.

In order to better describe the behavior of congestions CDFs of the time between congestions was constructed; it can be seen in figure 5.6. The figure shows a generally similar inter-arrival time for congestions on the four paths. However, for more in depth analysis, like in the case of link failures, congestions are divided depending on the frequency of appearance into: frequent and dispersed.

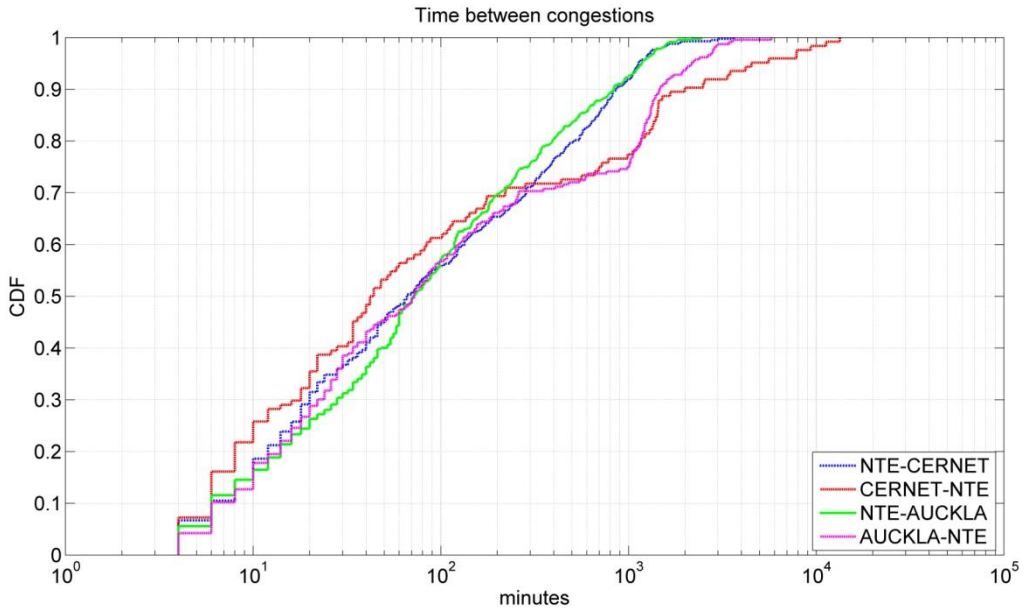


Fig. 5.6 CDF of time between congestions

Based on this thresholding, it is noticed that, for all paths, approximately 30% to 40% of the congestions are frequent. Consequently, the majority of congestions will be dispersed. This finding comes to support the theory that Internet backbone does not contribute much to congestions since it is better maintained than access networks. Paths AUCKLA – NTE and CERNET – NTE hold the largest inter-arrival intervals, with 9 days and 5 days respectively.

CDFs fitting for this case yielded the results in table 5.4:

Path	Distribution	β	γ
NTE-CERNET	Birnbaum-Saunders	155.3	2.304
CERNET-NTE	Birnbaum-Saunders	296.5	3.514
NTE-AUCKLA	Birnbaum-Saunders	150.2	2.158
AUCKLA-NTE	Birnbaum-Saunders	227.2	2.708

Table 5.4 Time between congestions CDF - distribution parameters

All paths CDFs are best approximated by Birnbaum-Saunders distributions, also known as fatigue life distributions. The cumulative distribution function for this distribution is:

$$F(x;\beta,\gamma) = \Phi\left(\frac{\sqrt{\frac{x}{\beta}} - \sqrt{\frac{\beta}{x}}}{\gamma}\right) \quad , \text{for } x > 0 \tag{5.3}$$

where Φ is the cumulative distribution function of the standard normal distribution. The fatigue life distribution is applicable in reliability engineering to describe the lifetime of components. The time to component failure is equivalent to the time to congestions.

A possibility to present the connection between congestion duration and the time between congestions is shown in figure 5.7. Congestions have been divided into frequent or dispersed, based on whether they occur at a smaller or larger time interval than 30 minutes. For each of the groups, further classification was done, into short/ transient congestions (for duration smaller or equal to 2 minutes) and long/ persistent congestions (for duration larger than 2 minutes). The plot shows that for the path between Norway and China, congestion types are split relatively equally between the four types of congestions. For NTE- AUCKLA path most of the congestions, 55%, are short and dispersed, which would be the best combination relative to the impact of congestions on applications' proper functioning. CERNET – NTE, on the other hand, has the largest number of long frequent congestions, more than 30% of all congestions occurring on the path. This behavior may create problems on the applications' side; therefore it has to be accounted for from the design phase.

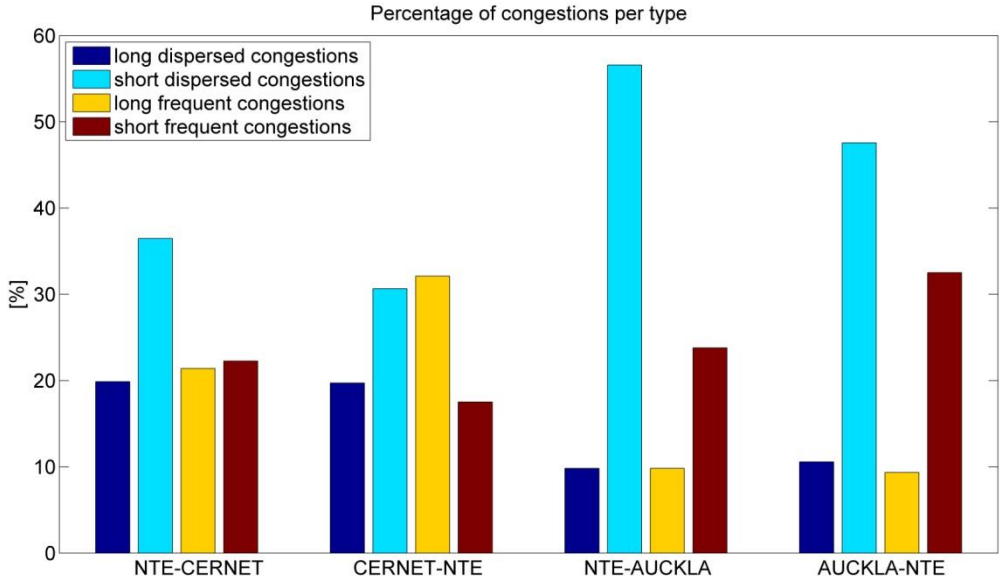


Fig. 5.7 Types of congestions – per path distribution

A better perspective to as how the congestions are distributed and what their possible impact would be on the applications using the four analyzed Internet paths, is achieved by combining the observations from figure 5.7 with figure 5.8. This plot contains the percentages of the total number of congestions, distributed on each path. Both NTE – CERNET and NTE – AUCKLA have a high percentage of congestions. However, the first path has a higher

percentage of long frequent congestions, which makes it of lower quality. CERNET – NTE, which had the highest percentage of long frequent congestions, has, however, the lowest total number of congestions.

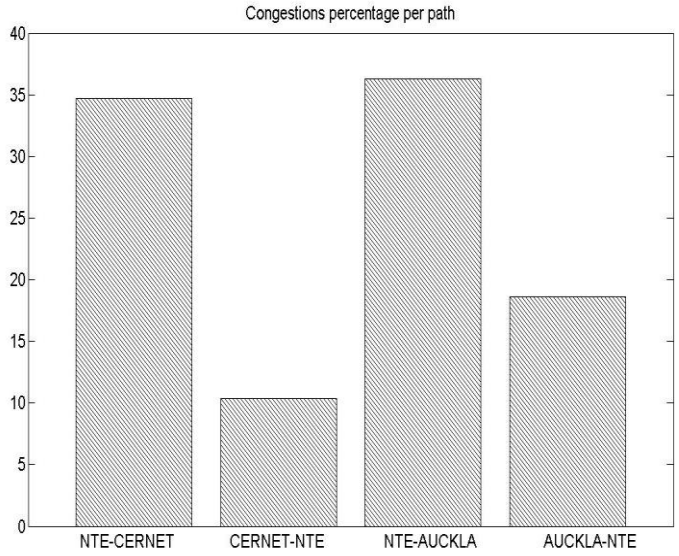


Fig. 5.8 Congestions percentage per path

5.3.3 Queues Build up Analysis

The analysis of queues build up for the four measured Internet paths follows the same lines as the analysis of link failures and the one of congestions. Queues build up causes are varied; in the Internet they happen mainly due to the queuing policy in the routers on the paths, in addition to the traffic load [14]. Therefore the size and type of data packets are important. However, the causes of queues are not the aim of this analysis; the goal is to find the behavior of the queuing building up process so as to be able to better operate networks. Queues build up states are generally of short duration (several seconds), due to the short queues length in today's routers. Queues induce spike in per-packet delay, which may be harmful for applications.

Throughout this sub section, in order to generate the plots, the same sets of Dragon-Lab signaled queues building up have been used; these four sets of detected queuing states are unique and fixed for each path.

To begin with, as in the cases of link failures and congestions, the empirical CDF of queues duration has been plotted in figure 5.9. It confirms that the majority of queuing states are of short duration, 100% for the NTE- AUCKLA path, and minimum 63% for the rest of

the paths. However, on the CERNET – NTE path there is 15% queues with duration up to 6 minutes.

The distributions that fit best the CDFs of queuing duration are, similar to the congestions case, exponential distributions with the parameters from table 5.5.

Path	Distribution	μ
NTE-CERNET	Exponential	2.49
CERNET-NTE	Exponential	3.01
AUCKLA-NTE	Exponential	2.52

Table 5.5 Queue duration CDF - distribution parameters

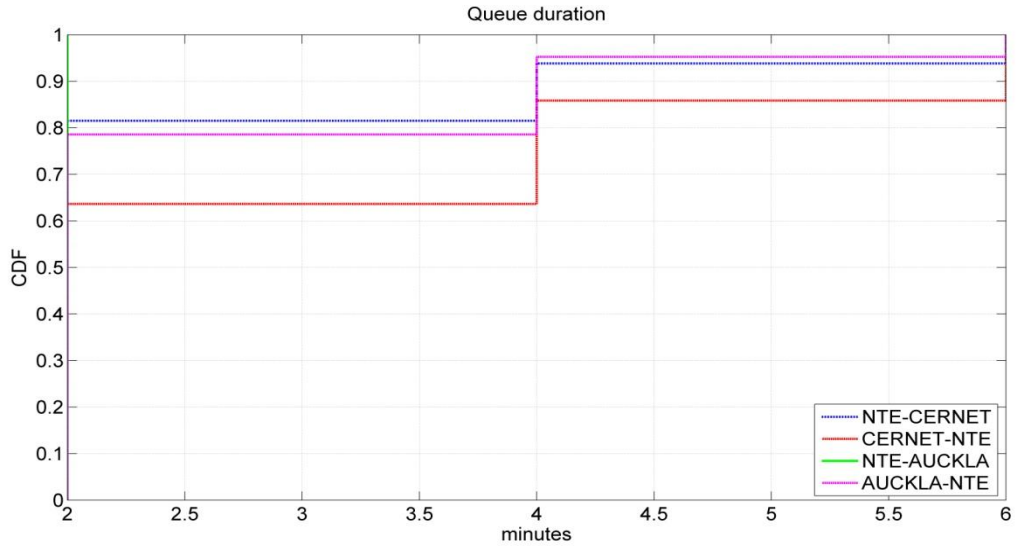


Fig. 5.9 CDF of queue duration

Even if the duration of queues might be relatively high, the CDFs of the time between queues, presented in figure 5.10, show that their frequency is not so high. NTE – AUCKLA, which had all queues of length maximum 2 minutes, has the most dispersed queues, with just 10% happening at less than 30 minutes apart. If the same threshold of 30 minutes is considered, the other three paths have around 20% to 30% of all queues detections within this limit. For the Norway – China paths the maximum time between queues is approximately 7 days. The results of these CDFs, combined with the threshold of 30 minutes between frequent and dispersed queues and the different queue durations, can be seen in figure 5.11.

The best approximations for the time between queues building up is given, as for the case of congestions, are Birnbaum-Saunders distribution with the parameters as in table 5.6.

Furthermore, the duration of the queues build up has been divided into short duration, for instabilities duration of less than 2 minutes, medium length queues, for durations between 2 and 4 minutes, and long duration queues, for durations between 4 and 6 minutes.

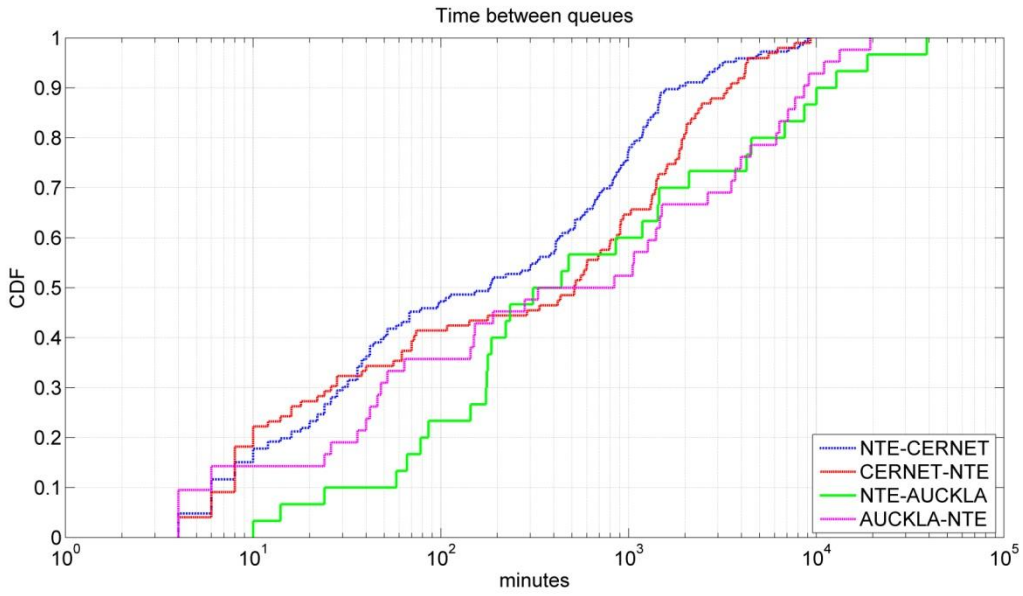


Fig. 5.10 CDF of time between queues

Path	Distribution	β	γ
NTE-CERNET	Birnbaum-Saunders	273.9	3.075
CERNET-NTE	Birnbaum-Saunders	305.1	3.459
NTE-AUCKLA	Birnbaum-Saunders	1258	3.285
AUCKLA-NTE	Birnbaum-Saunders	488.3	4.335

Table 5.6 Time between queues CDF - distribution parameters

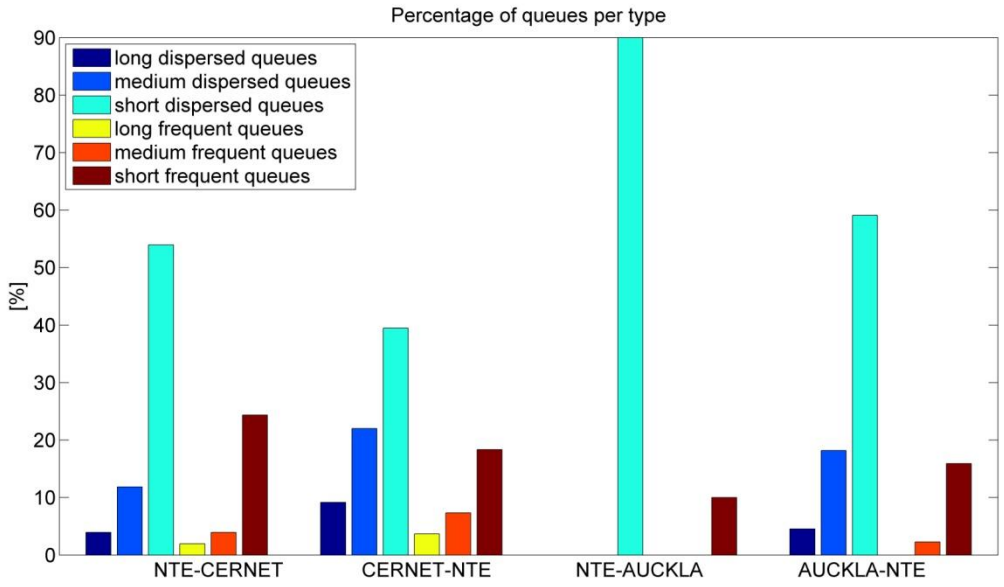


Fig. 5.11 Types of queues – per path distribution

As observed in figure 5.11, all paths have a majority of short dispersed queues, which is encouraging with respect to the effect of queues build up on network performance. NTE – AUCKLA path has an impressive 90% of short dispersed queues; the rest are frequent but still short. This makes sense if the result is compared to the CDF of queue duration for this path, where it was shown that all queues building up on this path have short duration, namely maximum 2 minutes. The rest of the paths stand within 40% to 60% of short dispersed queues, which infers good network performance with respect to queues. Medium length queues are particularly present on the CERNET – NTE path, where they account for 20% of all queues. This path also holds the most long frequent queues, which are considered most disruptive for network traffic; moreover queues are distributed rather much over all six types of queues defined, inferring higher instability for this path.

Important within the analysis of queues build up is the percentage of queues distributed on each of the four analyzed Internet paths. The distribution is shown in figure 5.12. This plot emphasizes that path NTE – AUCKLA offers the best performance. In addition to the high percentage of short dispersed queues, if also holds the least number of queues build up out of all paths. Path CERNET – NTE, on the other hand, has a high number of instabilities and, as shown in the previous figure, a significant part is long or medium and frequent; this add up to the poor performance of the path, from the queue build up point of view.

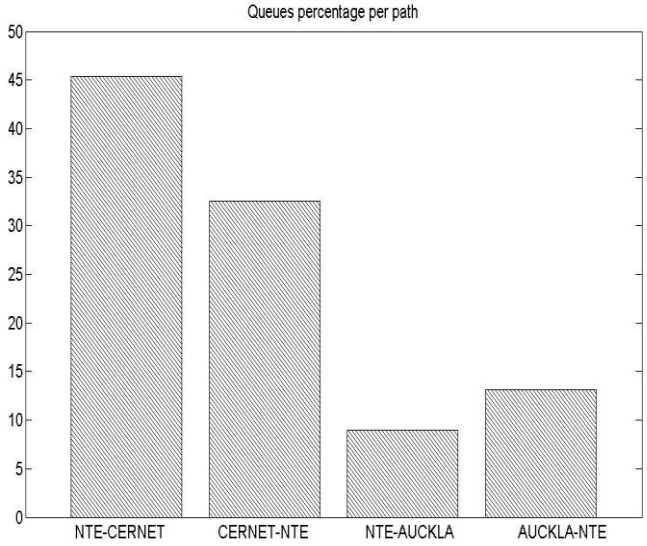


Fig. 5.12 Queues percentage per path

Generally, paths between Norway and China have more frequent longer duration queues build up in comparison to the paths between Norway and New Zealand, which are characterized by more dispersed queues with shorter duration.

5.4 General Instabilities Analysis

In the previous section the behavior of each type of Dragon-Lab detected instability has been presented. However, it is also important to know what the minimum achievable performance of the network is, regardless of the type of instability.

Therefore in this section the previously analyzed instabilities are combined in order to give a model of the impact of instabilities, irrespective of type, on the Internet paths performance. A first step in this analysis is the CDFs of instabilities for every path; this is shown in figure 5.13. The distribution is the result of the reunion of link failures, congestions and queues build up durations. Link failures and queues have lengths up to 6 minutes; therefore it is natural that the CDF in figure 5.13 resembles much the CDF of congestions duration; however the influence on the short duration instabilities, namely up until 6 minutes, affects the entire distribution. The paths NTE – CERNET and NTE – AUCKLA have instabilities of maximum 28 minutes durations, whereas for the other two paths the duration is maximum 1 hour. However, 60% of all instabilities duration are below 2 minutes, therefore of short duration.

The best fit for the CDFs of instabilities durations is the exponential distribution. This proves that congestions and queues, which have the same type of distributions, dominate disruptions durations. The parameters for the exponential distributions are given in table 5.7.

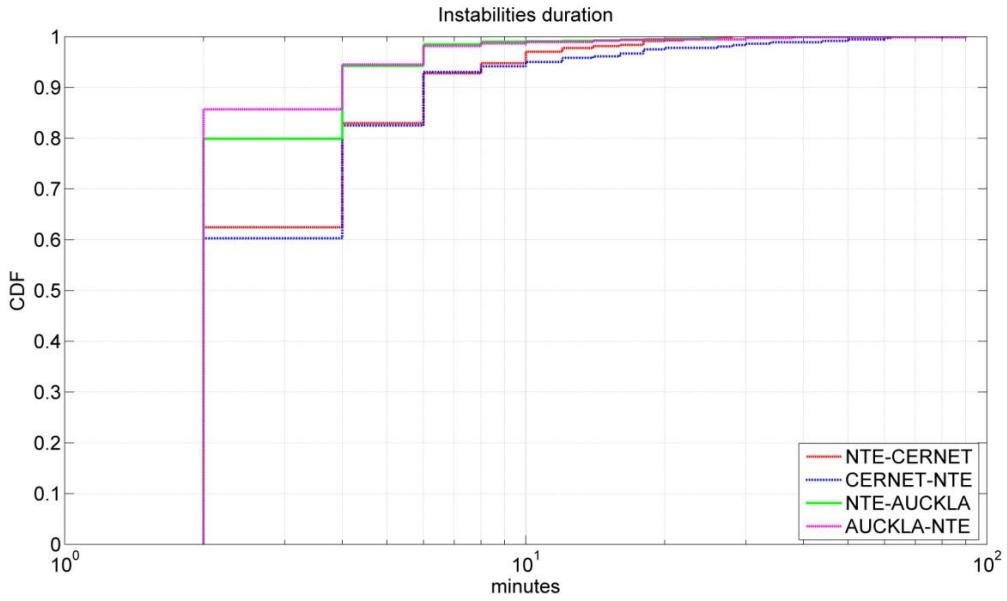


Fig. 5.13 CDF of instabilities duration

Path	Distribution	μ
NTE-CERNET	Exponential	3.564
CERNET-NTE	Exponential	4.261
NTE-AUCKLA	Exponential	2.662
AUCKLA-NTE	Exponential	2.688

Table 5.7 Instabilities duration CDF - distribution parameters

The following step in the analysis process is the visualization of the time between instabilities. The CDF for it is shown in figure 5.14. It can be seen that the maximum inter-instabilities time differs from one path to the other: for the Norway – New Zealand paths it is approximately 30 hours, whereas for the path between Norway and China it is higher, around 70 hours.

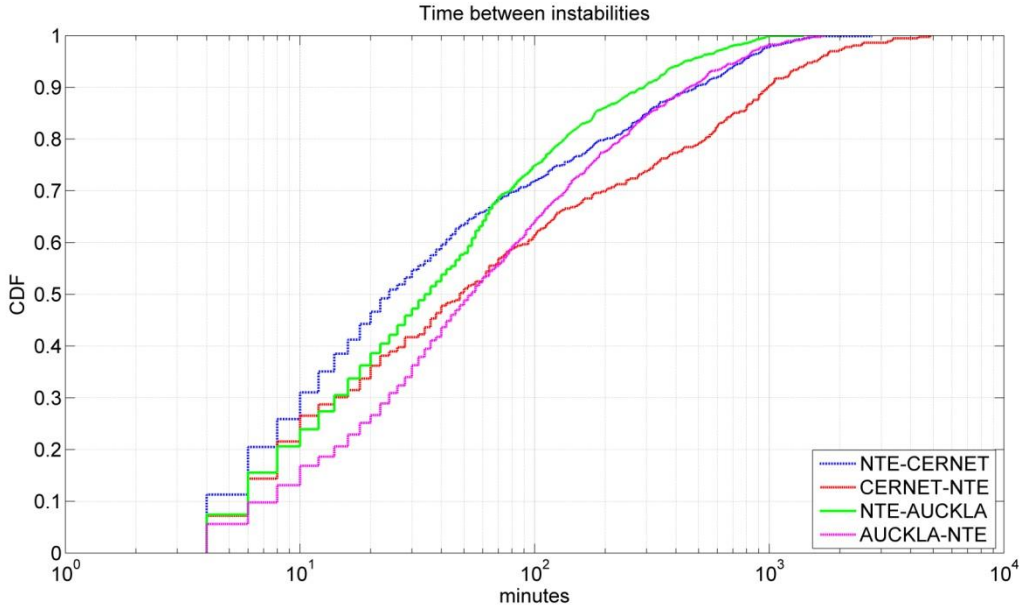


Fig. 5.14 CDF of time between instabilities

A significant percentage of the instabilities, from 60% to 90% depending on the path, happen within a time interval of 3 hours. This may infer correlation between instabilities; however this will be studied in future work.

The CDFs of time between instabilities are well approximated by the Birnbaum-Saunders distribution, also known as fatigue life distribution. This distribution models the life cycles of an entity during the growth of stress, which resembles the impact of disruptions on Internet performance, which is considered cumulative. Birnbaum-Saunders distributions have also approximated best time between congestions; consequently this may reinforce the observation that congestions are influencing instabilities on the Internet backbone more than the other types of disruptions. The parameters of the fatigue life distributions, in the cases of the four analyzed Internet paths, are given in table 5.8.

Path	Distribution	β	γ
NTE-CERNET	Birnbaum-Saunders	48.96	2.134
CERNET-NTE	Birnbaum-Saunders	81.06	2.566
NTE-AUCKLA	Birnbaum-Saunders	41.66	1.716
AUCKLA-NTE	Birnbaum-Saunders	59.32	1.838

Table 5.8 Time between instabilities CDF - distribution parameters

5.4.1 Metrics Analysis

Previous analysis focused on the distribution of particular types of instabilities and instabilities in general. In the following the focus is laid on the impact of the combined different instabilities on the Internet paths.

Different instabilities have different effects on the paths:

- link failures affect the time the network is available; in essence one is able to communicate between two end-points, irrespective of the quality of the communication;
- congestions and queues building up affect the quality of an available network.

In order to express the possible combination of these effects on the network, the following metrics have been defined:

1. Availability: expresses the amount of time the network is available.
2. Fatigue: expresses the effect of congestions and queues on an available network; a network with affected performance due to these two instabilities is a network under fatigue.
3. Stability: a network not under fatigue is a stable network. On this type of network a minimum quality of service is guaranteed.

Thus the metrics have the targets presented in the following table:

Metric	Target
Availability	Impact of link failures
Fatigue	Impact of congestions and queues
Stability	Impact of instabilities

Table 5.9 General metrics target

Practically, the three metrics are defined over a particular amount of time, in the case of this analysis it will be per hour. Thus the total measurements time will be divided in hours, in order to be able to calculate the metrics. Dragon-Lab data is used to compute the metrics; the time unit in this type of data is called time bin and is lasts for 2 minutes. This time unit will be used for calculation of the metrics.

Mathematically, the three metrics are defined as follows:

$$Availability = \frac{Total\ \#TB - \#TB\ LF}{Total\ \#TB} \quad (5.4)$$

$$Fatigue = \frac{\#TB\ (CONG + QUEUE)}{Total\ \#TB - \#TB\ LF} \quad (5.5)$$

$$Stability = \frac{Total\ \#TB - \#TB\ (LF + CONG + QUEUE)}{Total\ \#TB} \quad (5.6)$$

,where TB stands for time-bins, LF for link failures, CONG for congestions, QUEUE for queues.

Each of the defined metrics will be analyzed based on its cumulative distribution function, for each of the four paths between Norway, China and New Zealand.

Availability

The availability CDF is given in figure 5.15. Several facts can be clearly observed: path CERNET – NTE has the highest availability, 77%, with 94% of the time units available 100% of the time, while path NTE – AUCKLA has the lowest availability, 70%, with 75 of the time unit available 100% of the time. Availability is calculated in time units/bins available divided by the total number of time bins existent during one hour time interval. Since availability expresses the impact of link failures, there is a connection to the types and distribution of link failures per path, presented in figure 5.3 and to the percentage of link failures per path, from figure 5.4.

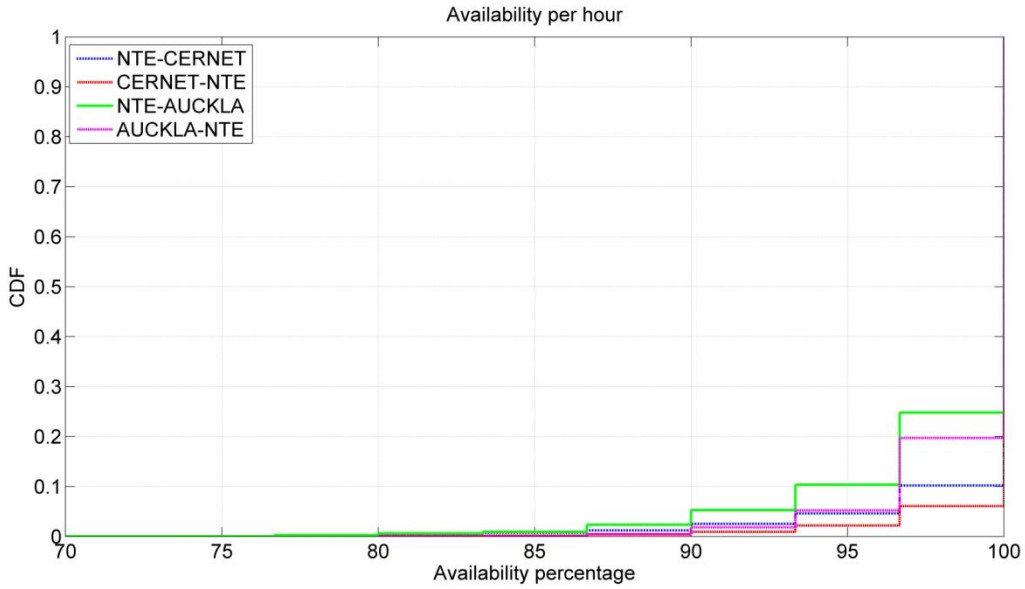


Fig. 5.15 Availability CDF

The path with the lowest availability, NTE – AUCKLA, is also the path with the highest percentage of failures per path, and it is the path that holds the highest percentage of frequent link failures; thus all possible conditions to achieve a poor availability. The situation

for path CERNET – NTE is the opposite: it has the lowest percentage of total link failures and there is a very small percentage of frequent failures. The other two paths maintain the trend given by the total percentage of failures per path, namely NTE – CERNET has the second highest availability and the second lowest number of total link failures, and AUCKLA – NTE the third number of total link failures and the third highest availability.

The duration and frequency of link failures has thus little influence on the general ranking of availability when the total percentage of link failures differs significantly. This might also be due to the fact that most link failures in this analysis were of short duration and dispersed.

Fatigue

Fatigue expresses the impact of congestion and queues on the performance of an available network. For analyzing this metric, similarly to availability, the CDFs of fatigue on all paths have been constructed; they are shown in figure 5.16. In the case of fatigue, as for availability, the path that performs worst is NTE – AUCKLA, which 0% fatigue for only 82%

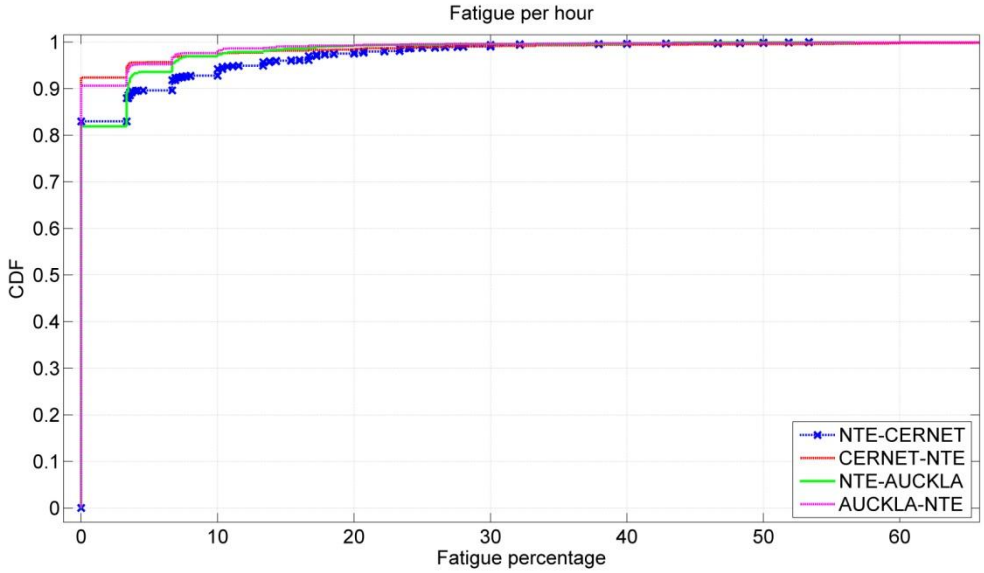


Fig. 5.16 Fatigue CDF

of the time units. CERNET – NTE has 92% of the available time units with 0% fatigue. The maximum fatigue rate varies from 53% for NTE – CERNET to 100% (figure 5.16 has been cut for better visualization) for AUCKLA – NTE. Fatigue may be influenced by congestions and queues building up distributions of the four paths, especially by their distribution over the measurement period. Thus it is worth to verify the behavior of congestions and queues once more. NTE – AUCKLA path holds the majority of congestions; however they are mostly of short duration and dispersed. CERNET – NTE has the least number of congestions, but most of them are frequent and of long duration. Queuing has an opposite behavior for the two paths. In this case, CERNET – NTE has higher percentage of queues and more long frequent queues than NTE – AUCKLA. However, the fatigue distribution seems to be influenced by

congestions rather than queues, since the distribution presents a long tail induced by the long duration congestions.

Stability

This metric expresses the impact of instabilities, irrespective of type, on the performance of each path. It gives a measure of the percentage of time units unaffected by instabilities, in essence stable time units. The distribution of stability over the four paths is presented in figure 5.17.

Based on the analysis so far, there are two paths, NTE – AUCKLA and CERNET – NTE, with extreme behavior of instabilities. Particularly, NTE – AUCKLA has the most significant percentages of link failures and congestions, but one of the lowest of queues, whereas CERNET – NTE has the lowest percentages of link failures and congestions and highest of queues, out of all paths. This situation should be expressed by the stability metric. An observation of the distribution of stability shows that NTE – AUCKLA has the worst stability: 64% of time units are 100% stable; and CERNET – NTE has the highest stability, with 88% of time units being stable. This situation proves once more the great influence of congestions and link failures on the Internet paths performances, and that traffic is not so much affected by queues building up.

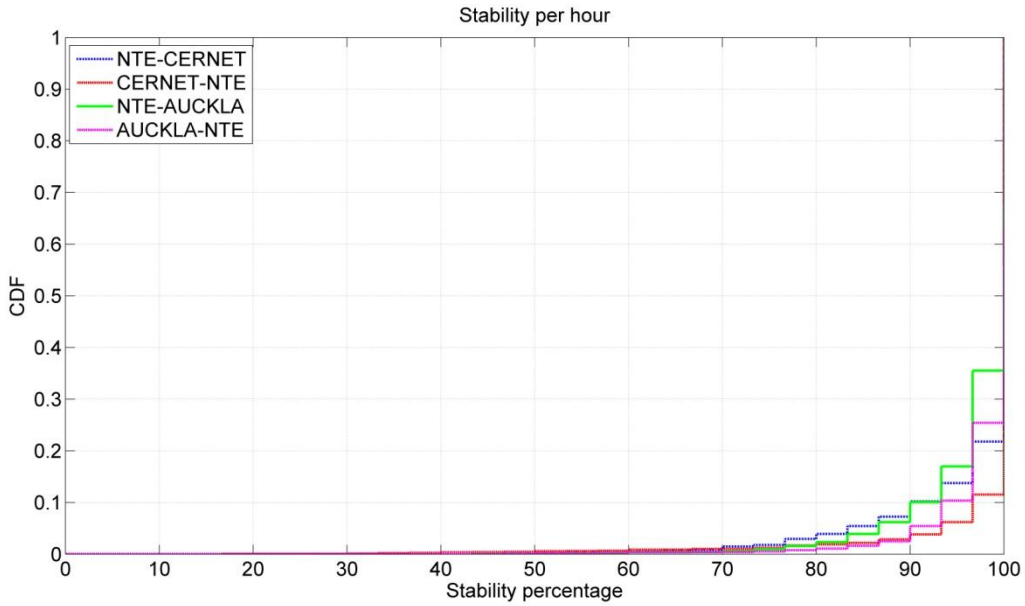


Fig. 5.17 Stability CDF

5.5 Conclusion

This chapter provided an analysis, based on the Dragon-Lab processed information, on the impact of instabilities on the paths' performance and behavior.

It is structured in two parts:

- an analysis of the three types of instabilities, link failures, congestions and queues building up, in order to provide statistical characteristics of their behaviors, for the purpose of generating realistic behavior models for use as input to network design and traffic engineering;
- an analysis of the combination of the different instabilities in order to give a model of the general behavior of instabilities on the four Internet paths.

The chapter shows that a complete analysis of the four Internet paths, with respect to of the unstable network states, is possible based on Dragon-Lab data. This is achieved using two end-to-end metrics: packet delay and packet loss. Compared to the techniques presented in Chapter 2, Dragon-lab accomplishes network troubleshooting targets, thus more complex tasks, using just delay measurements.

In the next chapter the current features of Dragon-Lab will be extended for attaining instabilities localization on Internet paths; the piece of information to the current data is traceroute data.

Chapter 6

Dragon-Lab Instabilities Localization

6.1 Introduction

Dragon-lab is a network troubleshooting framework able to diagnose the Internet backbone states using two end-to-end metrics: packet delay and packet loss. The diagnose consists of detecting and identifying network instabilities. In this way, there is no need for information regarding the structure, operations or events in the backbone from the ISPs' side.

The four possible network states that can be detected by Dragon-Lab are: stable states, congestions, queues and link failures. The framework provides an output vector consisting of time-bins of non-zero values for the unstable network states, namely congestions, queues and link failures, and zeros time-bins for the stable states. Because the instabilities are positioned at certain moments in this vector, this type of detection is called time localization detection.

The time localization of instabilities is very important as it gives an image of the behavior of the network, allowing the performance evaluation and the possibility of modeling the network. Consequently, applications will be able to tune their parameters based on previous knowledge of the network. Time localization provides the necessary framework for the analysis conducted in Chapter 4.

However, an attractive perspective would be not only to be able to localize the instabilities temporarily, but also to be able to pinpoint them spatially – to be able to indicate the node in the network that is causing the problems. So far two metrics have been used: delay and loss. By adding one additional piece of information to the equation, traceroutes, Dragon-Lab will be able not only to time localize the instabilities, but also to spatially localize them. It will be able to identify the node, in essence IP address, which is the cause of the excessive delay.

Additionally to providing the theoretical assumptions, capabilities and accuracy of the method, since Dragon-lab framework is aimed to be implemented as a network analysis software product, a localization algorithm will be presented based on the observations made across this chapter.

6.2 Available Data

Dragon-Lab is an automated framework for network diagnosis. The diagnosis consists of detecting and identifying any significant unusual change in the measured network features: delay and loss. By identifying Internet backbone network states, Dragon-Lab is actually performing a time localization of the various instabilities. The framework's performance in achieving this has been studied in previous work and the benefits of the results have been studied in Chapter 4.

However in this chapter is it aimed to localize the instabilities spatially, namely identify the IP or IPs which are causing the instabilities. The input information for this challenge is:

1. Dragon-Lab generated output vectors;
2. Aggregate delay data;
3. Traceroute data.

1. Dragon-Lab analyzes three metrics at the same time: aggregate delay, average delay and packet loss. It is using Robust PCP to detect the abrupt variations in aggregate and average delay and it combines these results with the correspondent packet loss in order to identify the network state. In the case of this chapter the instabilities that present interest are congestions and queues, since link failures will not provide valid traceroutes. Therefore, out of the instabilities vector generated by Dragon-Lab, only the time bins representing congestions and queues are kept. An example of the filtered Dragon-Lab output vector for one day measurements is plotted in the following figure:

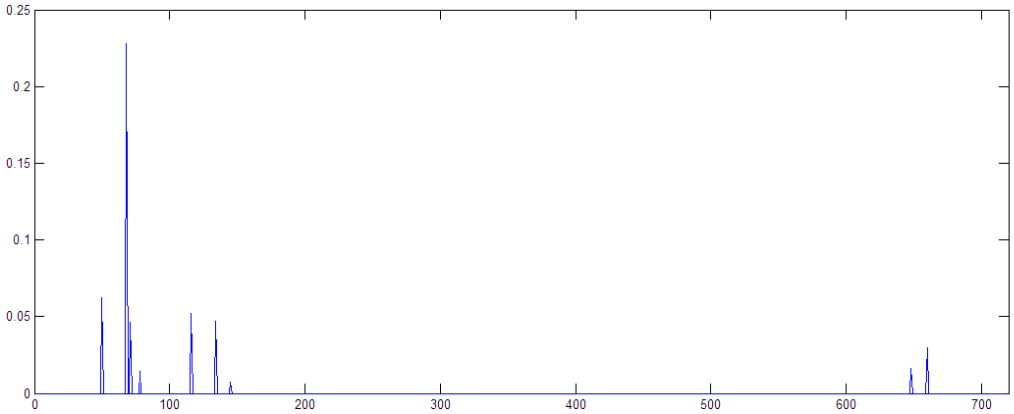


Fig. 6.1 Dragon-Lab output

2. Correspondent to the detected congestions and queues time-bins from the output in figure 6.1, the aggregate delay can be selected from the measurements performed on the path in question. The aggregate delay data is organized as 82 rows representing the days, per 719 columns, representing the 2 minutes time-bins, for each day. From this data the important time-bins are the ones correspondent to the detected instabilities. This is illustrated in figure 6.2 with the plot of the aggregate delay data for the same day of measurements as in figure 6.1. As it can be observed from the figure, the aggregate delay data has relatively similar magnitude values.

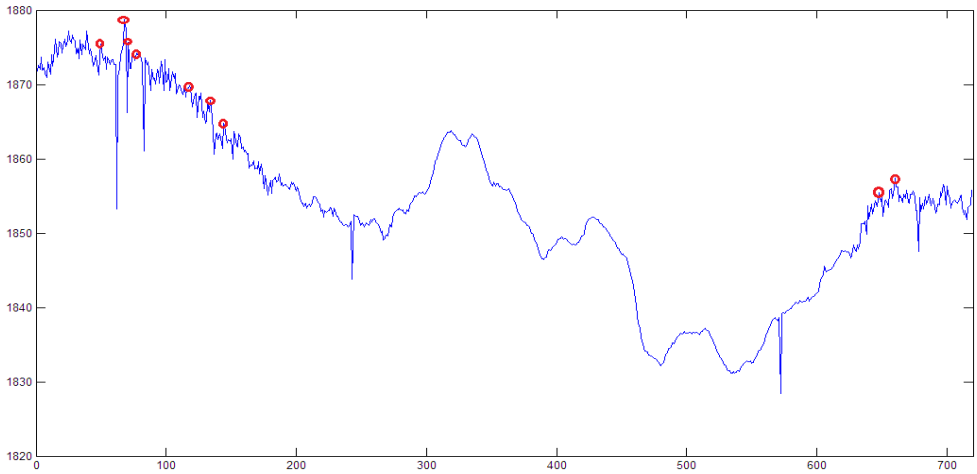


Fig. 6.2 Aggregate delay data

3. The traceroute data is available every two minutes. After basic text processing the data will have the format as follows:

06:06:28		08:37:57	
1	IP_1	1	IP_1
2	IP_2	2	IP_2
3	IP_3	*	
4	IP_4	4	IP_4
5	IP_5	5	IP_6
6	IP_7	6	IP_8
7	IP_9	7	IP_10
8	IP_11	8	IP_12
9	IP_13	9	IP_14
10	IP_15	10	IP_16
11	IP_17	11	IP_17
…		…	
19	IP_25	19	IP_25

Every traceroute starts with the time instance when it was taken; afterward, the IPs that compose the route are listed. In the example above, every real different IP has received an IP number. It can be noticed that routes have 19 nodes for this end-to-end path. Moreover the

first 4 nodes and the last 8 nodes are the same for all traceroutes. Another observation is the star present on the second traceroute. The IP of node 3 of the path is missing. This can be due to a link failure or to the fact that firewalls ban ICMP packets; traceroutes containing these types of elements will be discarded.

Based on the three types of input information, Dragon-Lab should be able to localize the source of the instabilities. Practically it is attempted to identify the node or nodes causing the detected instabilities. The only source to obtain this is the traceroute information, namely the IPs of the nodes. The fitting of the data for being able to process it in order to achieve the goal is presented in the next subsection.

6.3 Dragon-Lab Problem Statement

Dragon-Lab is desired to be an automatic instabilities detection-localization framework. To this end the detection and analysis of instabilities follow an easy automated line. However, localization of the instabilities also needs to be performed. At this point the available information is the one in the previous subsection, 6.2. It will have to be arranged in a feasible way for processing.

The components of the equation will be: a list of end-to-end aggregate delay, a number of traceroutes and the list of nodes involved in the traceroutes.

As presented in subsection 6.2 the only part of the delay that presents interest is the aggregate delay that corresponds to the Dragon-Lab detected instabilities. Once more, the only types of instabilities considered for this analysis are congestions and queues. The reason for this is that link failures determine loss of traceroute probes that cause the appearance of stars in the traceroutes and consequently IPs cannot be recovered. As shown in figure 6.2, aggregate delay corresponding to a few time bins will be selected. The data is presented in the form of a vector, a measurement vector.

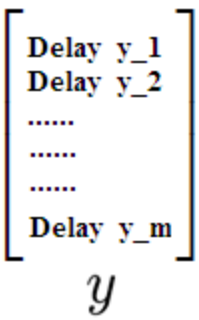


Fig. 6.3 Measurement vector

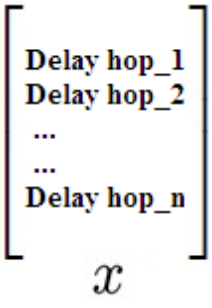


Fig. 6.4 Data vector

1930,86
1929,14
1930,13
1933,41
1933,36
1932,66
1934,81

Table 6.1 Measurement vector example

Practically the measurement vector has values in seconds in the order of the ones in table 6.1. This is the measurement vector for the NTE – Auckland path for measurement day 11. It can be observed that the values of the delay are relatively close to each other, so noisy, considering the magnitude of the values (in the order of 1900 seconds).

The unknown in the attempt to localize the Dragon-Lab detected instabilities is a vector of the type shown in figure 6.4. It will be called the data vector. It contains the complete list of delays on nodes/ hops involved in the traceroutes correspondent to the instabilities. Out of empirical reasons the extent of the time periods for which all instabilities within are analyzed, is one day. Therefore, the data vector will contain a list of delays. Each of them represents the delay for a particular node. As an example, for hop number two, the data vector will hold, in the position given by the numerical ID of the hop, the delay registered on it. In essence, the data vector will hold a list delay values with peak on the nodes that are the cause of instabilities. It is assumed that the bigger delay values will be responsible for the instabilities. Knowing their position within the data vector offers the IP associated with them and so the geographical position can be obtained.

In order to make the processing easier, each different IP was tagged with a unique ID (an integer number) in the order of IP occurrence during the measurements day. This is better explained with an example from the same path and the same day as for the measurement vector above, namely path NTE – Auckland for day 11. The real IPs are not given due to proprietary reasons, but different IPs are tagged in the order of appearance in these two traceroutes; the assigned IDs on the ID columns.

Traceroute IPs	ID	Traceroute IPs	ID
06:06:28		08:37:57	
1 IP_1	1	1 IP_1	1
2 IP_2	2	2 IP_2	2
3 IP_3	3	3 IP_3	3
4 IP_4	4	4 IP_4	4
5 IP_5	37	5 IP_6	5
6 IP_7	6	6 IP_8	21
7 IP_9	41	7 IP_10	43
8 IP_11	42	8 IP_12	42
9 IP_13	9	9 IP_14	49
10 IP_15	10	10 IP_16	36
11 IP_17	11	11 IP_17	11
…	…	…	…
19 IP_25	19	19 IP_25	19

Table 6.2 IP – ID example

This Internet path contains 19 hops; hops 1 to 3 and 11 to 19 are stable over the whole day. The remaining nodes shift between a range of IPs, due to either load balancing or possible route changes caused by instabilities. It is the last category that presents interest. As you may

have noticed following the IP with ID = 4, may come ID = 37. This happens because the tagging of the IPs has been done for the whole day, previous to the selecting of the traceroutes involved in the instabilities. Practically the path between NTE and Auckland, for the traceroutes considered as involved in instabilities, not only the two present in table 6.2, will look like in figure 6.5: 3 common IPs for all routes in the beginning, several possible routes in the middle of the path, due to load balancing over the Internet, and another 9 common nodes in the end. As noticed in the figure the path passes several ASs. The complete traceroutes scheme contains in fact 18 possible nodes for the 8th hop of the path, so it is much more varied. However this shows how varied the routes are between the end points. This fact helps since, as it will be seen later, the diversity of the routes results in a diversity of the matrix that will be constructed based on them.

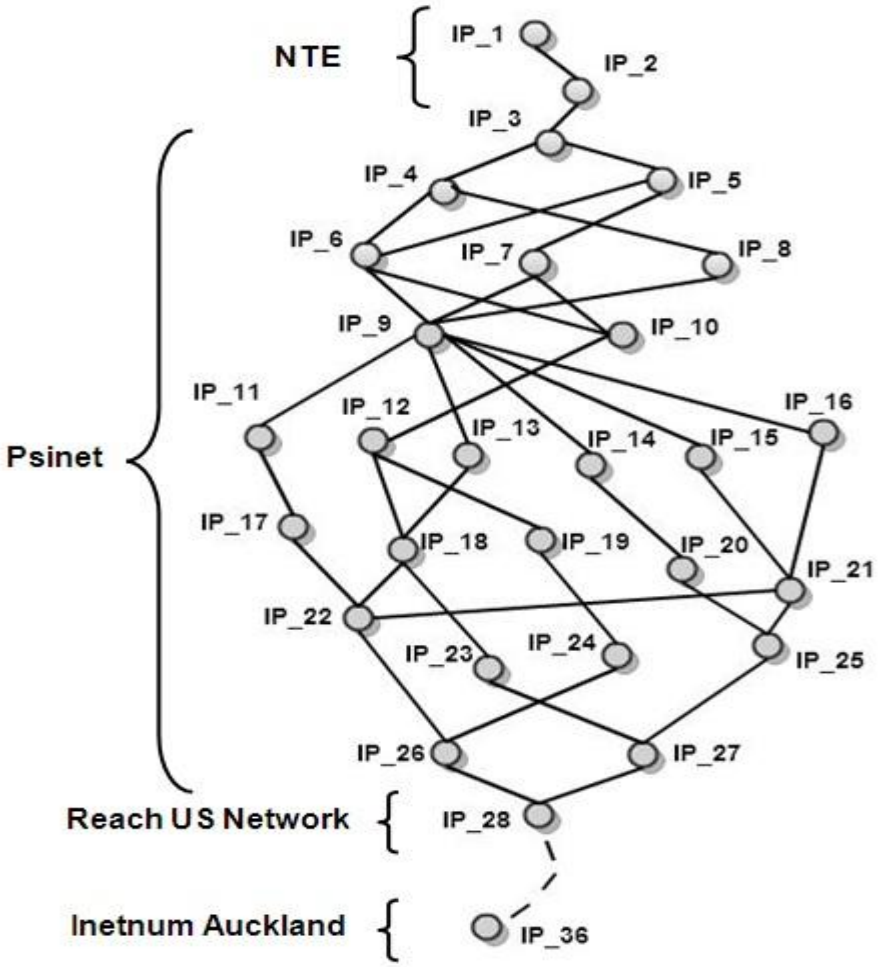


Fig. 6.5 NTE – Auckland routes involved in instabilities

In order to connect the two already available vectors, the vector of measurements and the data vector (vector of unknowns), a binding element is needed. It has to make the connection between the cumulated aggregated delay on the entire Internet path, in essence the measurements vector, and the delay represented by each of the nodes involved in the instabilities during one day.

This is achieved by constructing a routing matrix based on the traceroutes and the assignment of IDs to the nodes.

As explained earlier, IDs are attributed to each of the IPs present in the traceroutes during one day of measurements. Afterward, the traceroutes associated with the instabilities are selected. There is no perfect synchronization between the time bins corresponding to the instabilities and the time of the traceroute. Moreover the traceroute marks one moment in time, whereas the time-bins last over two minutes. The number of the time-bin gives the time interval during the day when the measurement was conducted. For example: time-bin number 112 corresponds to the real time 3:42:00 - 3:44:00. Consequently a convention is needed so as how to select the right traceroute. This is the following:

- choose the traceroute within the time interval determined by the time-bin number; even if there is a traceroute within this time interval, it is not 100% sure that the nodes that appear in that traceroute are the ones that cause the instability; this is due to the fact that the traceroute returns the IPs at one moment in time; routes however may change during the two minute interval of the time bin;
- if there is no traceroute in the first case, choose the first traceroute before the time interval determined by the time bin number; since the nodes that cause instabilities are tracked, the reason behind this choice is that: if a node causes the instability it is more probable to be in the traceroute before the actual anomaly that is detected.

Once rules for choosing the right traceroute exist, and the IPs within the traceroutes of one day measurements are assigned IDs, the construction of the routing matrix is possible. The sequence is the following:

- take first traceroute; place ones on the first row of a zeros matrix on the column with the index equal to the ID from the traceroute;
- repeat previous step for all traceroutes increasing the row index.

1	1	0	1	1	0	1	0	0	0	0	0
1	1	0	1	1	0	1	0	0	0	0	0
1	1	1	0	1	0	1	0	1	0	0	0
1	1	0	1	1	0	0	0	0	0	0	1
1	1	0	0	1	0	1	0	1	0	0	0
1	1	1	1	1	0	1	0	0	1	0	0
1	1	0	0	1	1	1	0	1	0	0	0

Table 6.3 Routing matrix example

The result of such a construction in the case of NTE – Auckland path will be a matrix like in table 6.3. The routing matrix is not complete. On the NTE – Auckland path for day 11, the ID numbers go until ID = 51, therefore the routing matrix will have 51 columns. However, it illustrates the possible scenarios regarding the content of the columns of a routing matrix.

The scenarios are:

1. random number of ones;
2. columns of ones – caused by the fact that some IPs are the same on all traceroutes;
3. columns of zeros – caused by the fact that some IPs miss from the traceroutes involved in the instabilities.

At this point all needed information is available for creating a system of equations as the one presented in figure 6.6. Based on the form of the system and on the desired output it is now possible to make some modifications on the routing matrix:

- eliminate columns of zeros; they will determine a zero position in the data vector which does not indicate an instability; the indices of the eliminated columns must be saved in order to make a mapping between the initial data vector and the one with fewer entries (since some columns, which are as many as the unknowns, were eliminated);
- eliminate the columns of ones; they will determine false results when solving the system; the ones columns are determined by the nodes at the beginning and at the end of the paths and belong to the ISPs from the end of the paths; it is assumed that these nodes are reliable since they do not apply load balancing and that the instabilities are caused by nodes within the path, nodes that employ load balancing.

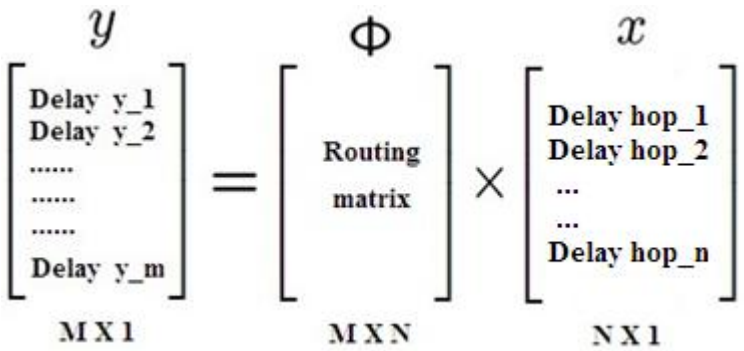


Fig. 6.6 Equation scheme

On the particular example of path NTE – Auckland, after the ones and zeros columns are eliminated, there are 24 columns left; this means 24 possible nodes where instabilities might appear. Even after removing approximately half of the nodes contained in the traceroutes, the system of equations has 7 equations and 24 unknowns, which makes it highly underdetermined. However, out of the total number of unknowns, is it assumed, again empirically, that the delay is mainly distributed on few of the nodes, since the nodes are core Internet backbone routers and consequently it is unlikely that more than 2 fail simultaneously on the same path. The rest of the nodes induce delay, but it is significantly lower than the delay on the anomaly-causing nodes. In other words, the data vector is a compressible signal.

Conclusion

The solution to solve this Dragon-Lab problem is the use of Compressive sensing reconstruction algorithms.

The motivation for using Compressive sensing is:

- the general form of the Dragon-Lab problem, in essence that it can be arranged as system of equations;
- the presence of an underdetermined linear system of equations;
- the compressibility of Dragon-Lab data vector.

Compressive sensing will be presented in the following section.

6.4 Compressive Sensing

Compressive sensing/ compressed sensing (CS) is a technique for finding sparse or compressible solutions to underdetermined linear systems. An underdetermined system of linear equations has more unknowns than equations and generally has an infinite number of solutions. However, if there is a unique sparse solution to the underdetermined system, then the Compressive sensing framework allows the recovery of that solution. Not all underdetermined systems of linear equations have a sparse solution. [17]

This technique has its origins in the signal processing field, particularly in signal acquisition. For acquiring a signal, it has to be sampled. In order to be able to reconstruct the signal a minimum number of samples is needed. Classically, the minimum sampling rate is given by the Nyquist rate, equal to twice the highest frequency of the original signal. However, the amount of data generated in nowadays sensing systems has become so large that even the Nyquist rate will generate a very large amount of samples. This may lead to very expensive or even impossible to build devices capable of reaching the required rate. Some of the areas which these high requirements are: imaging, video, medical imaging, remote surveillance, spectroscopy.

The classical solution for being able to process high-dimensional data is compression; it tries to find the most concise representation of a signal having a certain level of distortion. Transform coding is one of the most known techniques for signal compression. It is based on finding a basis that allows for sparse or compressible representation of signals in a class of interest. Sparse and compressible signals can be well represented by preserving the largest coefficients of the signals. This technique is called sparse approximation. For a signal of length N, the signal is sparse if it can be represented using $K << N$ coefficients, and the signal is compressible if it can be well-approximated by the K non-zero coefficients.

Based on the sparse approximation concept, Compressive sensing has emerged as a new framework for signal acquisition. For signals of a certain type, namely sparse or compressible in a known basis, CS is able to sample at a much lower rate than the Nyquist rate and still be able to recover the signals. The idea behind CS is: instead of sampling at a high rate and then compressing, directly sense the data in a compressed form – basically at a lower rate.

The founders of CS may be considered Emmanuel Candès, Justin Romberg, and Terence Tao and of David Donoho. They are the ones who proved that a finite-dimensional signal with a sparse or compressible representation can be recovered from a small set of linear, non-adaptive measurements. [18], [19], [20]

Compressive Sensing Problem Statement

The case of standard finite-dimensional compressive sensing model is presented. Given a measurement system and a signal $x \epsilon R^N$ the formulation of the CS problem is presented in figure 6.7. The process can be mathematically expressed as: $y = \Phi x$ (6.1)

The measurement system is considered to take M linear measurements of signal x, which is K sparse; therefore the measurements vector $y \epsilon R^M$. The number of measurements M is generally much smaller than the dimension of the signal; the CS matrix or measurements matrix Φ represents a dimensionality reduction, mapping R^M to R^N. If the system can be solved, it means that the signal x is well-posed and that there have been taken enough number of samples. However, it is possible that the system is under-determined even for the CS framework. This might happen if the sparsity of the signal is larger than the number of measurements. The measurement process in CS is considered to be non-adaptive; this means that the elements of Φ are fixed; they do not depend on previous measurements.

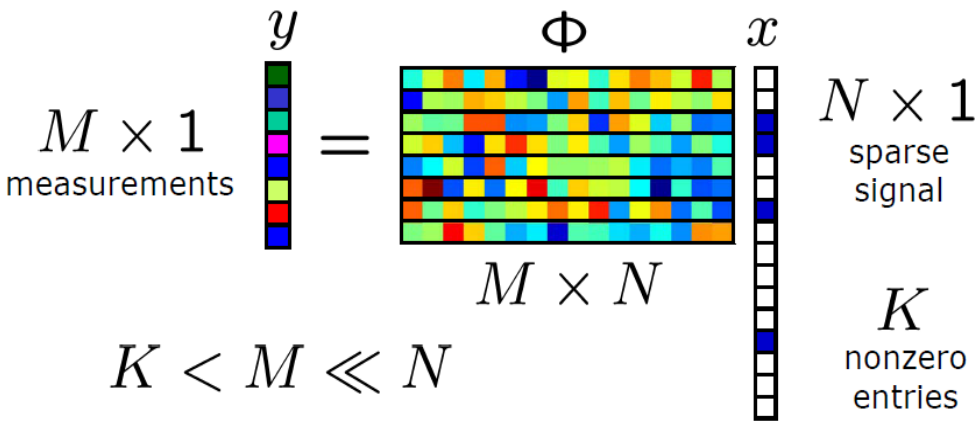


Fig. 6.7 Compressive sampling scheme

The two fundamental premises underlying CS are:

- sparsity: for being able to recover a signal from a small number of measurements, the signal has to have a sparse representation or to be compressible; the limit of the sparsity in relation to the number of measurements will be analyzed later on;

- incoherence: the coherence measures the correlation between the columns of the measurement matrix; it is desired that the coherence is small, thus high incoherence; the implications of coherence and the practical limits will be presented further.

The design of a functional CS method is based on two stages:

1. Constructing a stable measurement matrix which preserves the information while reducing dimension from N to M
2. Designing of a reconstruction algorithm which recovers *X* from *Y* by making use of only M measurements.

1. The measurement matrix is essential for the reconstructing algorithms to be able to recover the noisy signal from the reduced number of measurements. It is been proved by the authors of [21] that in order for a matrix to fulfill the above target, it is sufficient for the matrix to satisfy a condition called the Restricted Isometry Property (RIP). Mathematically the property can be stated as:

$(1 - \delta_k) \|x\|_{l_2}^2 \leq \|\Phi x\|_{l_2}^2 \leq (1 + \delta_k) \|x\|_{l_2}^2$, where $\delta_K \in (0,1)$ is called restricted isometry constant (6.2)

The significance is that Φ should be a transformation that preserves the distance for all K sparse vectors x. If the property holds, then all K subsets of the columns of Φ are nearly orthogonal. Evaluating the RIP property is a NP-hard problem so the property serves as a theoretical foundation for CS, but practically it does not say how to build the measurement matrix.

A property of Φ, that is related to the RIP, which is easily computable and provides recovery guarantees, is the mutual coherence, μ, of the matrix. Mutual coherence, denoted $\mu(\Phi)$, is the largest inner product between any two columns φ_i , φ_j of Φ, and it is mathematically defined as follows:

$$\mu(\Phi) = \max_{1 \leq i < j \leq N} \frac{|\langle \varphi_i, \varphi_j \rangle|}{\|\varphi_i\|_2 \|\varphi_j\|_2}. \quad (6.3)$$

It has been proven that the coherence has the following bounds: $\mu(\Phi) \in \left[\sqrt{\frac{N-M}{M(N-1)}}, 1\right]$. If N>>M, then the lower bound will approximately be $\mu(\Phi) \geq 1/\sqrt{M}$.

In [22] the authors provide the connection between the sparsity of the signal, K, and the coherence of the measurement matrix so that for each measurement vector $Y \in R^M$, it exists at most one signal *x* so that $y = \Phi x$. This connection is mathematically expressed as:

$$K < 1/2 \left(1 + \frac{1}{\mu(\Phi)}\right) \quad (6.4)$$

This formula will provide, in the context of the Dragon-Lab analysis, a limit for how many anomalous nodes will be possible to recover based on the incoherence of the measurement matrix for the chosen day.

Another important bound related to the RIP property is the necessary minimum number of measurements needed to satisfy the RIP property of order 2K with constant $\delta_K \in (0, 1/2]$.

$$M \geq CK \log\left(\frac{N}{K}\right), \qquad \text{where } C = 1/2 \log(\sqrt{24} + 1) \approx 0.28. \tag{6.5}$$

A simple way to construct a measurement matrix that satisfies the RIP and incoherence conditions is, for example, to choose its elements so that they are independent and identically distributed random variables from a Gaussian probability distribution with zero mean and 1/N variance. The measurements vector will then contain M randomly weighted linear combinations of the elements in *x*. Authors of various recovering algorithms may construct matrices based on other criteria, matrices that yield best recovery performance on their particular algorithms.

A particular type of matrices, namely binary sparse matrices, has been proven by the authors of [23] to work well with most of the algorithms using norm 1 signal reconstruction; the experiment results are presented in [24]. Their study is especially important because the routing matrices used in Dragon-Lab are binary sparse matrices.

2. The second stage in the design of a compressive sensing method is the design of a reconstruction algorithm. Considering noisy measurements: $y = \Phi x + e$, the main problem in CS is to recover the signal x from the set of measurements y. The design of recovery algorithms account for various criteria out of which the most important are:

- minimal number of measurements; the number of measurements should be as small as possible for stable recovery of the K sparse signals; a lower limit for M was given by (6.5);
- robustness to measurement noise; in practical measurements noisy is inevitable; therefore recovery algorithms should be stable to both signal noise and to noise added to the measurements;
- speed; the algorithms may deal with large amounts of data, thus they must use as little resources as possible.

Since M < N, the equations system in (6.1) is underdetermined, so there are infinitely many x' that satisfy $\Phi x' = y$. The reconstruction of *x* is attempted with the use of the concept of l_p norm. The l_p norm of a vector *x* is: $\left(\|x\|_p\right)^p = \sum_{i=1}^{N} |x_i|^p$ (6.6)

Three types of reconstruction based on the l_p norm have been attempted [26]:

- Minimum l_2 norm reconstruction

In order to find the solution x, the vector with the minimum l_2 norm (also called the energy norm) that solves equation (6.1) is searched. L_2 norm measures the energy of the signal. Mathematically, the system that has to be solved is given in (6.7). The minimization returns non-sparse vectors $\hat{x}$ with many non-zero elements, being almost never possible to reconstruct the original signal.

$$\hat{x} = argmin \|x'\|_2 \ \ such\ that \ \ \Phi x' = y \tag{6.7}$$

- Minimum l_0 norm reconstruction

L_0 norm counts the number of non-zero elements in x therefore it is more suitable for reconstruction of sparse vector; l_0 norm of a K sparse vector is K. The mathematical formulation in this case is (6.8). The approach looks attractive, but solving the equation is numerically unstable and requires an exhaustive search of all $\binom{N}{K}$ possible locations of the K non-zero entries.

$$\hat{x} = argmin \|x'\|_0 \ \ such\ that \ \ \Phi x' = y \tag{6.8}$$

- Minimum l_1 norm reconstruction

If the minimization is conducted using norm 1, which returns the sum of the non-zero elements, it has been proven that algorithms are able to recover K sparse signals and well approximate compressible signals, using M measurements. The mathematical formulation is:

$$\hat{x} = argmin \|x'\|_1 \ \ such\ that \ \ \Phi x' = y \tag{6.9}$$

In the presence of noise the above formula will have the following form:

$$\hat{x} = argmin \|x'\|_1 \ \ such\ that \ \ \|\Phi x' - y\| \leq \varepsilon \text{ ,where } \varepsilon \text{ is the error bound} \tag{6.10}$$

Most of the reconstruction algorithms are based on l_1 minimization.

They can mainly be split into three categories:

1. Convex optimization algorithms
2. Greedy algorithms
3. Combinatorial algorithms [27]

1. Convex optimization algorithms aim to optimize a convex function of the unknown signal *x, J(x),* over a convex subset of R^N. *J* is a sparsity-promoting cost function. When noise is present in the measurements another function, say *H*, is involved which penalizes the distance between the vectors *Φx* and *y*. The general formulation of the problem is:

$$min_x J(x) + \mu\, H\,(\Phi x,\, y) \tag{6.11}$$

μ is a penalty parameter which may be chosen by trial-and-error or by statistical techniques. The most common choices for *J* and *H* are:

$$J(x) = \|x\|_1 \text{ and } H(\Phi x, y) = \tfrac{1}{2} \|\Phi x - y\|_2^2. \quad (6.12)$$

Convex optimization algorithms guarantee convergence to the global minimum. One might want to use conventional optimization packages for the above formulation, however there are two challenges that have to be overcome:

- real-applications are large-scale; optimizations over millions of variables have to be performed, which is not possible for standard optimization software;
- the objective function is not smooth and standard smoothing techniques do not give good results; therefore conventional algorithms involving matrix factorizations are not applicable.

2. An alternative to the convex optimization algorithms are the greedy algorithms. The goal of the sparse approximation is to find the sparsest vector which solves the system of equations. The following non-convex problem has to be solved:

$min_S\{|S| : y = \sum_{i \in S} \varphi_i x_i\}$, where S is a subset of indices and φ_i is the i-th column of Φ (6.13)

Searching over the power set formed by the columns of Φ for the optimal subset S is a NP-hard problem. However, these algorithms greedily select columns and form successively better approximations for y. Greedy algorithms rely on iterative approximation of the signal and support. They either iteratively identify the signal support until a convergence criterion is met, or they obtain an improved estimate by accounting for the mismatched to the measurements vector. Some of the best known algorithms that fit this category are: Matching Pursuit (MP), Orthogonal Matching Pursuit (OMP) or Stagewise Orthogonal Matching Pursuit (StOMP).

3. A third approach to solve the CS problem is represented by the combinatorial algorithms. They have been developed in the computer science community, in the context of group testing. In these types of problems it is assumed that from a set of N items, K have faults; the K elements will be non-zero and the rest of the N values will be considered zero. The aim is to produce a smallest number of tests to be able to identify the position and possibly the value of these faulty items. In this context the tests are represented by a matrix which has 1 in the position where an item is tested; the tests are represented by the rows, the items are identified by columns. In this way the problem of recovering x becomes the same as the sparse recovery problem.

Some of the most known combinatorial algorithms are: Random Fourier Sampling, HHS Pursuit or Sparse Sequential Matching Pursuit.

There are various differences between combinatorial algorithms and the other two classes of algorithms. For example, in the combinatorial algorithms there is full control over Φ. It is

preferred that Φ has as few non-zero entries as possible, such that the amount of computation needed is reduced; this makes combinatorial algorithms faster than the other classes. In contrast, convex optimization algorithms and greedy ones work on any measurement matrix as long as it satisfies general condition such as RIP (6.2).

6.5 Compressive Sensing in the Context of Dragon-Lab

The previous section gives an understanding of the possibilities and general applicability conditions of compressive sensing. It also gives an insight of the various classes of reconstruction algorithms. One of the most important elements in the CS problem is the measurement matrix. Some algorithms have stringent constraints on it and therefore the range of their applicability decreases since matrices with particular structures are required; especially the algorithms in the combinatorial class. However, many of the algorithms work well on general measurement matrices. This fact encourages the use of CS algorithms in order to solve the Dragon-Lab problem, section 6.3.

Several CS algorithms have been tested on different routing matrices and different test vectors, in order to be able to make a comparison of the results and choose an algorithm that performs best.

The reconstruction performance of CS algorithms on Dragon-Lab data vectors will be tested, as it will be presented in the next section, on two routing matrices. These two routing matrices have been selected from an initial number of four. The four routing matrices have been constructed based on four scenarios; the scenarios depend on the position of the instabilities within the day and on the amplitude of the aggregate delay during the detected instabilities.

The scheme of the scenarios is:

- choose a day with spaced anomalous time-bins with varied aggregate delay amplitudes;
- choose a day with clustered anomalous time-bins with varied aggregate delay amplitudes;
- choose a day with distanced anomalous time-bins with close aggregate delay amplitudes;
- choose a day with clustered anomalous time-bins with close aggregate delay amplitudes.

The reasoning behind these choices is that: if instabilities are spaced over the day they might be caused by time - uncorrelated nodes, whereas clustered instabilities might be caused by the same node; varied aggregate delay amplitudes will generate a less noisy measurement vector whereas in the other case the recovery will be affected by high noise level. These four

scenarios, which practically represent four different days of measurement, on four different paths, have each their own routing matrix. In order to make a first selection of various available reconstruction algorithms, the worst case scenario matrix characteristics have been chosen.

***Methodology of CS applied on Dragon-Lab data**:*

1. Compute general performance characteristics of routing matrix:
- mutual coherence;
- maximum number of data vector peak components, i.e. sparsity K.

2. Build test CS matrix and data vector according to dimensions and characteristics of the worst case scenario routing matrix determined in 1 which will be used for the algorithms recovery performance testing section.

3. Select CS reconstruction algorithms that work with general sensing matrices and with compressible data vectors, and test them with data from point 2, in order for find which yields the smallest reconstruction error.

For each of the chosen four routing matrices the mutual coherence has been calculated according to (6.3): $\mu(\Phi) = \max_{1 \le i < j \le N} \frac{|\langle \varphi_i, \varphi_j \rangle|}{\|\varphi_i\|_2 \|\varphi_j\|_2}$. Two of the routing matrices have equal mutual coherence; however, it is desired to find the worst case scenario matrix, thus make a difference between matrices; therefore, another metric is calculated: average columns cross-correlation. Columns cross-correlation is defined as: $\frac{|\langle \varphi_i, \varphi_j \rangle|}{\|\varphi_i\|_2 \|\varphi_j\|_2}$, for i, j two different columns of Φ. This metric will express how much the columns differ from each other, which is important in compressive sensing. The third calculated metric, denoted "Max sparsity" in table 6.4, is the maximum recoverable data vector sparsity according to formula (6.4): $K < 1/2 \left(1 + \frac{1}{\mu(\Phi)} \right)$. The results can be seen in table 6.4. According to these calculations, the worst case matrix scenario is chosen as the one with the largest value for mutual coherence and the largest value for the average columns cross-correlation. Therefore, the dimensions of the routing matrix for the path between NTE and Auckland, and the corresponding form of its measurements vector, will be chosen in order to make a selection of the reconstruction techniques.

On the other hand, the best case scenario matrix is the routing matrix for NTE – Cernet, day 35, because it has the lowest mutual coherence. This matrix is not used for testing the algorithms, but it is itself tested in order to find out what the reconstruction performances are when using it.

	Mutual Coherence	Average columns cross-correlation	Max recovered sparsity	Matrix dimensions
Path NTE – Cernet, day 35	0.5774	0.3976	1	7 X 6
Path NTE – Auckland, day 11	1	0.2323	1	7 X 24
Path NTE – Cernet, day 11	0.8333	0.3184	1	8 X 4
Path Auckland – NTE, day 6	1	0.2105	1	10 X 36

Table 6.4 Routing matrices characteristics

The algorithms for the testing section, section 6.6, were selected based on:

- capability of solving problems similar to $y = \Phi x$ as defined in section 6.3, Dragon-Lab problem statement;
- generality of the sensing matrix they use;
- recovering of compressible data vectors. [28],[29]

The names, short description and source of the algorithms which were selected based on the criteria above are presented in the table 6.5.

All the algorithms have been tested on general random matrices, since reconstruction algorithms are designed for random matrices, and afterwards on the particular routing matrices constructed from Dragon-Lab traceroutes. Reconstruction algorithms are proven to work well on binary matrices also [23]. Some of them will be eliminated and only a few will be used for the recovering of the data vector based on the measurement vectors. This can be seen in the following section which is dedicated to testing.

Nr.	Name	Short Description	Source
1	Linprog	Part of the optimization toolbox of Matlab; solves linear programming problems; used for large scale optimization in the testing section.	[30]
2	L1EQ	Part of the l1Magic package; solves problem (6.9) by reducing it to a linear problem and using a primal-dual interior method.	[31]
3	GPSR	Gradient Projection for Sparse Reconstruction; it is a gradient projection algorithm where problem (6.12) is transformed into a bound-constrained quadratic programming (BCQP) problem.	[32]
4	BP	Part of the Sparse Lab package; solves problem (6.9) and makes use of a primal-dual interior method in order for solving the equations system.	[33]
5	LASSO	Implementation of the Least Angle Regression (LAR) algorithm for solving the least absolute shrinkage and selection operator (Lasso) problem (6.11) as in (6.12); this solver is part of the Sparse Lab package.	[33],[34]
6	StOMP	Stagewise Orthogonal Matching Pursuit; included in Sparse Lab; different from OMP as the algorithm operates in a fixed number of stages. Solves systems of the form in (6.9)	[33],[35]
7	YALL	L1 algorithms package developed at Rice University; contains several solvers; the one used solves a L1L2 problem with non-negativity constrain on x. Solves equation (6.12) with $\mu = 1/(2\rho)$. P is a sensitivity threshold parameter.	[36]
8	L1LS	Solves problem (6.12), namely the L1-regularized least squares problem which is further reformulated as a convex quadratic program and then solved using an interior-point method.	[37]
9	FIST	Fast Iterative Shrinkage-Thresholding Algorithm; gradient algorithm. Solves problem $\|\Phi x - y\|_2 + \varphi(x)$, where φ is a convex nonsmooth regularizer.	[33],[38]
10	BPDN	Basis Pursuit Denoising; it uses Homotopy as active-set method for handling the implicit bounds; solves the problem: $\min_{x, y-\Phi x}$ (tau $*$ L1(x) $+ 1/2 *$ L2(y $-$ Φx)^2)	[39],[40]
11	BCS	Bayesian Compressive Sensing; it is a Bayesian framework for solving compressive sensing problems by using the Relevance vector machine (RVM) that solves problem (6.10).	[41],[42]

Table 6.5 Algorithms description

6.6 Testing and Results

Target

There are two main targets for this section:

1. Perform reconstruction of the Dragon-Lab data vectors, basically localize unstable nodes, and validate the results.
2. Use the knowledge acquired from matrices testing in designing an algorithm for automatic localization for any type of Dragon-Lab input data.

In order to achieve the first target several tests have been conducted. The methodology that shortly follows was used. To achieve the second goal, the behavior of the algorithms on the routing matrices will be observed; afterwards, the best possible stages that an automatic algorithm should follow will be presented.

Testing Methodology

Testing is divided in 3 stages:

1. Testing of the general reconstruction algorithms presented in section 6.4, on general conditions sensing matrices and with data vectors that resemble the ones involved in Dragon-Lab data.
2. Testing of routing matrices performance using the algorithms selected at stage 1 and data vectors similar to the ones involved in Dragon-Lab data.
3. Dragon-Lab data vectors recovery.

A general conditions sensing matrix denotes a matrix satisfying the Restricted Isometry Property; practically it is a matrix with random entries. It is known that the sensing matrix has high impact on the reconstruction performance [18], [19]. Therefore, it is important to know how the algorithms perform on different types of matrices. Since no additional information concerning the data vectors is available in the Dragon-Lab case, it is very important to know how the algorithms perform on what is clearly known – the routing matrix.

Since it has been observed that the position in the data vector of the peak entry determines different recovery performance, three sets of test data vectors will be created for the tests.

In stage 1, the only known information is the dimensions of the sensing matrix; they have to match the ones of the worst case scenario routing matrix chosen in the previous section, 6.4. Since algorithms perform best on random matrices, the sensing matrix will be generated with elements in a random distribution. In order to make the transition to the routing matrix, another matrix of the same dimension will be tested; this second matrix contains random elements in $\{0, 1\}$. A decline or constancy in the reconstruction performance of the

algorithms will be observed. In order to be able to make this comparison the same data sets will be used on both matrix types.

Stage 1 will reduce the number of algorithms based on their reconstruction performance; worse performing algorithms are eliminated.

In stage 2, the worst case and best case scenario Dragon-Lab routing matrices are tested. The purpose is to determine the safety noise levels for which algorithms recover correctly the position of the peak entry from the data vector. Once more since the position of the peak entry is important, several scenarios will be tested. For one of the matrices its performance is compared also to the binary sense matrix from Stage 1, using the data sets defined for stage one.

Stage 3 uses the noise limits defined in stage 2 in order to guarantee recovery of the data vectors in the Dragon-Lab case based on real measurement vectors and the routing matrices.

6.6.1 General Reconstruction Algorithms Testing

In this first part the algorithms listed in table 6.5, which are designed to use general random sensing matrices, have been tested.

The goal of this section is to reduce the number of general reconstruction algorithms to the ones which perform best on the form of the CS problem given by the characteristics of particular routing matrices. CS algorithms are supposed to work well on large scale problems, which is not the case of Dragon-Lab for one day of measurements. The algorithms do not impose conditions on the size of the problems. However, the size of the problem, in essence the size of the sensing matrix, influences the properties of the matrix. Mutual coherence, even for a random matrix, is high for matrices with small number of rows.

Therefore, throughout this section, the form of the matrix with the worst characteristics has been chosen. This matrix was designated in the previous section as the one for path NTE - Auckland, measurements day 11. It has 7 rows and 24 columns, which makes it difficult even for a pseudo-random standard uniform distribution to achieve a low mutual coherence.

***The methodology of this section is**:*

6.6.1.1 Test algorithms reconstruction performance using a randomized sensing matrix for 3 different types of test data vectors depending on the position of the peak component/s;
6.6.1.2 Test algorithms reconstruction performance using a randomized binary sensing matrix using the same test vectors in order to compare results.
6.6.1.3 Make a decision on which algorithms performs best, based on the results.

This part of the testing process has the following similarities with the Dragon-Lab data:

- Dragon-Lab-like test data vectors;
- dimension of the sensing matrix for 6.6.1.1;
- dimension of binary matrix for 6.6.1.2.

For performing the tests, measurement vectors are obtained by multiplying the sensing matrix with the test data vectors. They are introduced as input variables for the algorithms. For both test cases, 6.6.1.1 and 6.6.1.2, graphs have been constructed representing a relative reconstruction error defined as:

$$E = \frac{\| \, |recovered\_data\_vector| - |data\_vector| \, \|_2}{\| \, |data\_vector| \, \|_2} \tag{6.14}$$

The tests are performed for data vector sparsity varying from 1 to 4. However the decision making errors are taken into about for sparsity 3 maximum. Sparsity 4 is kept for better aspect graphs. Each of the test data vectors have background noise generated as standard uniform distribution in the interval (0, 5) and 100 (unit-less) added for peak entries. Therefore the vectors can be considered compressible.

There are 3 sets of test data vectors depending on the number, namely sparsity K, and position of the high data vector entries; they are divided as follows:

- set 1: for K = 1, high entry located at the beginning of the data vector;
  for K = 2, high entries located away from each other;
  for K = 3, high entries located away from each other;
- set 2: for K = 1, high entry located in the middle of the data vector;
  for K = 2, high entries are consecutive;
  for K = 3, high entries are consecutive;
- set 3: for K = 1, high entry located at the end of the data vector;
  for K = 2, high entries located close to each other;
  for K = 3, high entries located close to each other.

These test vectors should represent most of the scenarios which may appear in a Dragon-Lab data vector. The sets are used both in 6.6.1.1 and 6.6.1.2, so that a comparison can be drawn. The sets of data vectors used can be seen in table A.1.

6.6.1.1 Randomized Sensing Matrix Testing

This part of testing has been conducted using the defined sets of test vectors and using a random, positive defined sensing matrix. The matrix has been generated using the elements is a standard uniform distribution on the interval (0,1). The mutual coherence of the matrix is 0.985; even for a pseudo-random matrix the mutual coherence is high. The matrix can be seen in table A.2.

For each data set and for each algorithm, the reconstruction error defined in (6.14) is calculated and afterward plotted for an easier observation of the results. The figures are presented alternatively, for the same vector data set, for this case and for the randomized binary matrix test, since it is important to see how much the difference in sensing matrix affects the reconstruction.

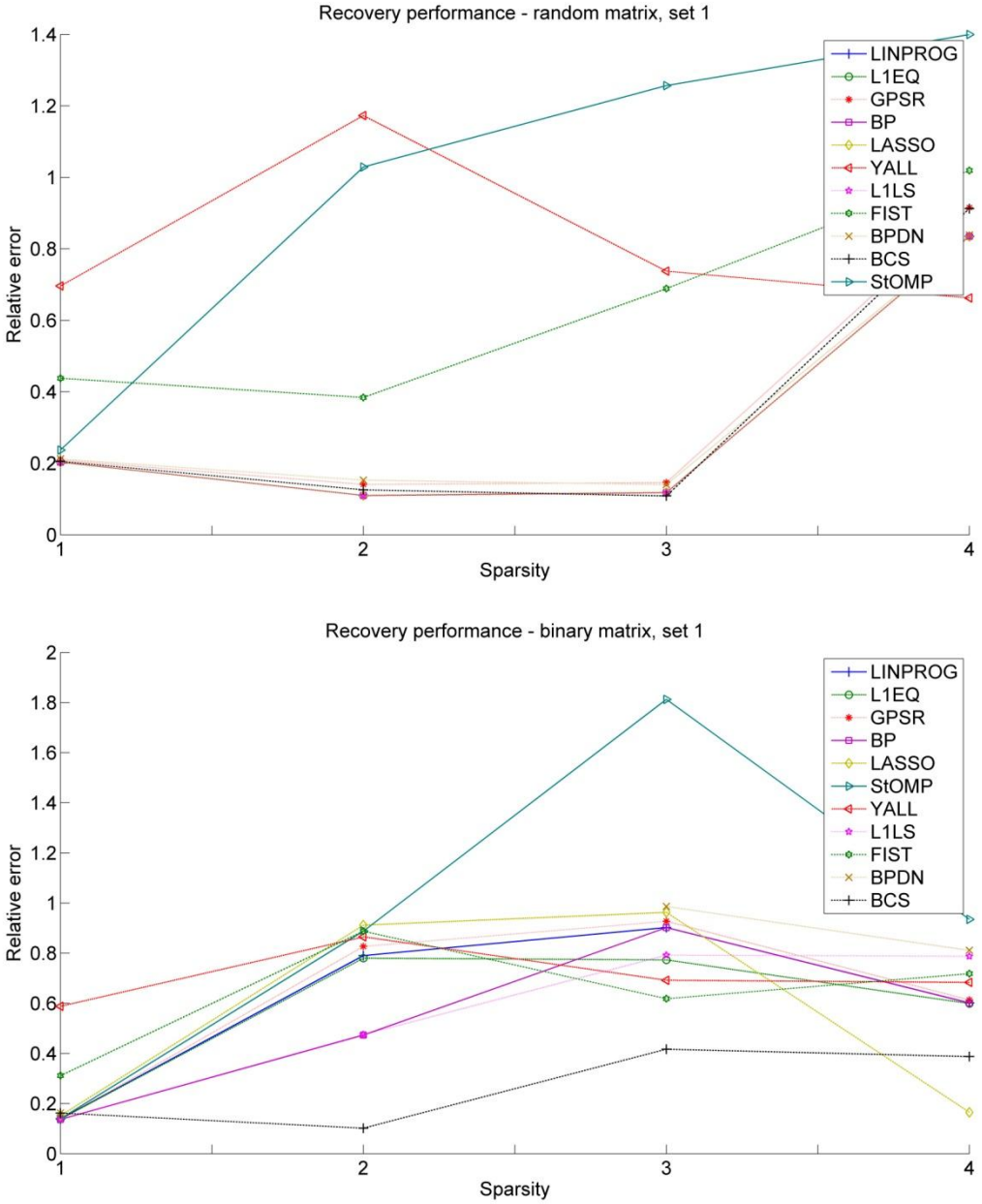


Fig. 6.8 Recovery performance – data set 1

6.6.1.2 Randomized binary matrix testing

As in the previous case, testing has been conducted using the defined sets of test vectors and using a binary matrix; this case has been considered since routing matrices are binary

matrices. The matrix has been generated to contain randomly positioned ones in a zeros matrix. The mutual coherence of the matrix is 1, just like two of the selected Dragon-Lab routing matrices. In this case the matrix has been randomly generated, but the mutual coherence is still not smaller than the maximum possible, 1. The matrix used in this case can be seen in Appendix, table A.3. As in the case of random sensing matrices, for each data set, for each algorithm, the reconstruction error as defined in (6.14) is calculated and afterward plotted. The results for this case and for the random matrices case are presented in figures 6.8, 6.9 and 6.10.

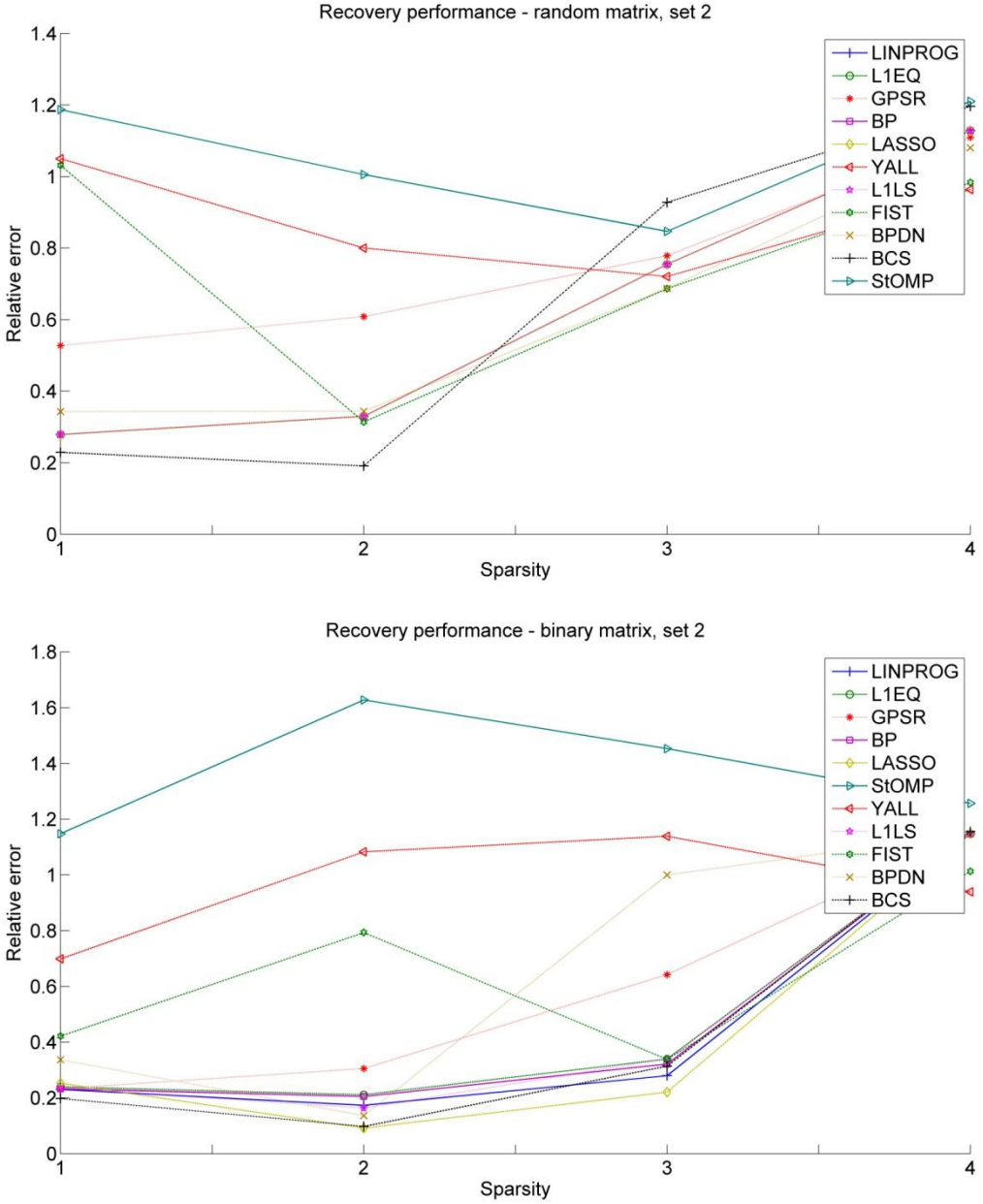


Fig. 6.9 Recovery performance – data set 2

6.6.1.3 Analysis Results

The comparison of figures 6.8, 6.9 and 6.10 is split qualitatively into two directions:

a) based on the used type of sensing matrix
b) based on the used data set

In the analysis of all the figures only sparsities 1, 2 and 3 are considered; the expression "all sparsities" will refer to these three.

a) The type of sensing matrix has a major influence on the reconstruction process. Thus figures are grouped together in order to facilitate the observation of the sensing matrix influence. Each of the figures is based on the same set of test data vectors.

In figure 6.8 the transition from a random matrix to a binary matrix results in an increase of the highest error for most of the algorithms and for the average error level; exceptions are YALL, which performs better and BCS, which is constant. It is noticed that for random matrix most of the algorithms have small reconstruction error and most of the curves, i.e. error values, are clustered. The introduction of the binary matrix significantly spreads the error values for sparsities 2 and 3. It is important to notice that sensing matrix does not have much influence in the error for sparsity 1.

Figure 6.9 shows the same trend as the previous figure: an increase in the general error level when changing from random matrix to binary matrix. Also the algorithms that perform better are mainly the same. Surprisingly, for the algorithms that have the lowest error for the random matrix, the error decreases even more at the introduction of the binary random matrix, especially for sparsities 2 and 3.

In figure 6.10 is it observed that the introduction of the binary matrix determines an increase in the error for K=1 for all algorithms, the errors for K=2 remain almost the same and errors for K=3 spread, but remain within the same values, relatively low.

b) The data set, namely positions where the largest entries in the data vector, has also its influence on the reconstruction results. To observe this, the figures, which represent results per data set, have to be observed separately, looking either to the upper (random matrix) or the lower (binary matrix) part of the figures. Because the binary matrix presents more interest, the lower part of each figure will be analyzed. The routing matrix is the same for all data sets, so the influence on the errors comes from the position of the peaks within the data vectors. The positions might also be regarded as the influence of the sensing matrix itself, by the structure of the columns to which the peaks correspond.

Sparsity K=1: since the theoretical reconstruction limit states that the maximum recoverable sparsity is one, the reconstruction error in this case is desired to be low regardless of the position of the peak in the data vector. This happens for most algorithms for data sets 1 and 2 (fig. 6.8, 6.9). In case of data set 3 the error increases, but is however less than 1.

Sparsity K=2: for consecutive peaks, data set 2, algorithms perform best, second best situation is peaks far apart, with error around 0.8 and the worst situation is when peaks are closely spaced.

Sparsity K=3: in this case the lowest error is obtained for closely spaced peaks; closely spaced error values are achieved also for consecutive peaks. The highest errors results from the spaced peaks.

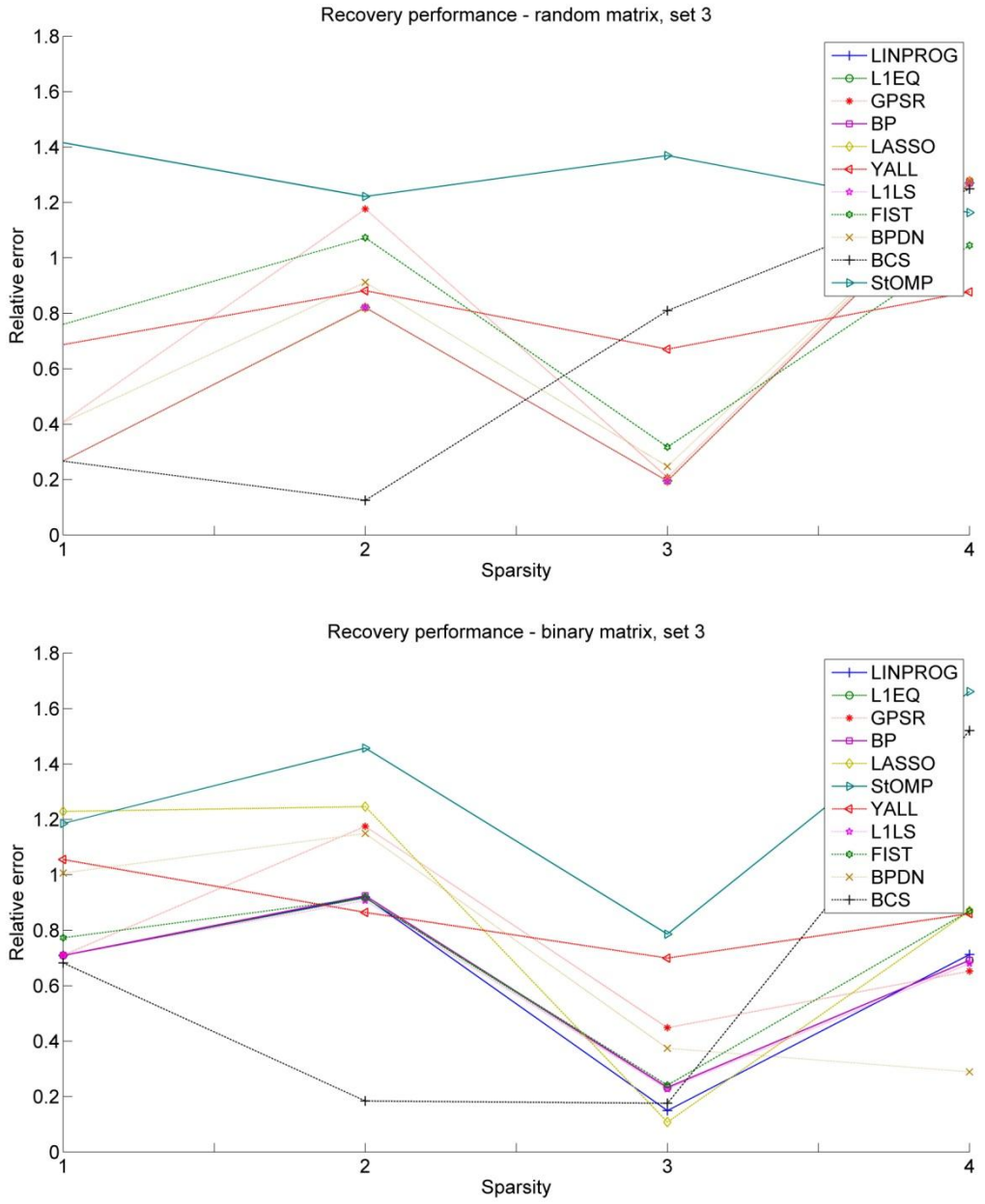


Fig. 6.10 Recovery performance – data set 3

Sparsities 2 and 3 are less important when it comes to selecting the best algorithms for the Dragon-Lab data, since the real sensing matrices, the routing matrices, do not have the

capacity to recover 2 or 3 peaks. Consequently the selection is based mainly on the performance for K=1 and on the value of the error.

In the figures it is noticed that errors go up to 1.6 or more. This means that the norm of the difference in absolute values between the recovered data vector and the original one can be 1.6 times bigger than the norm of the original data vector. Normally, if the data vector is reconstructed correctly this error should be 0. The situation is explained by figure 6.11.

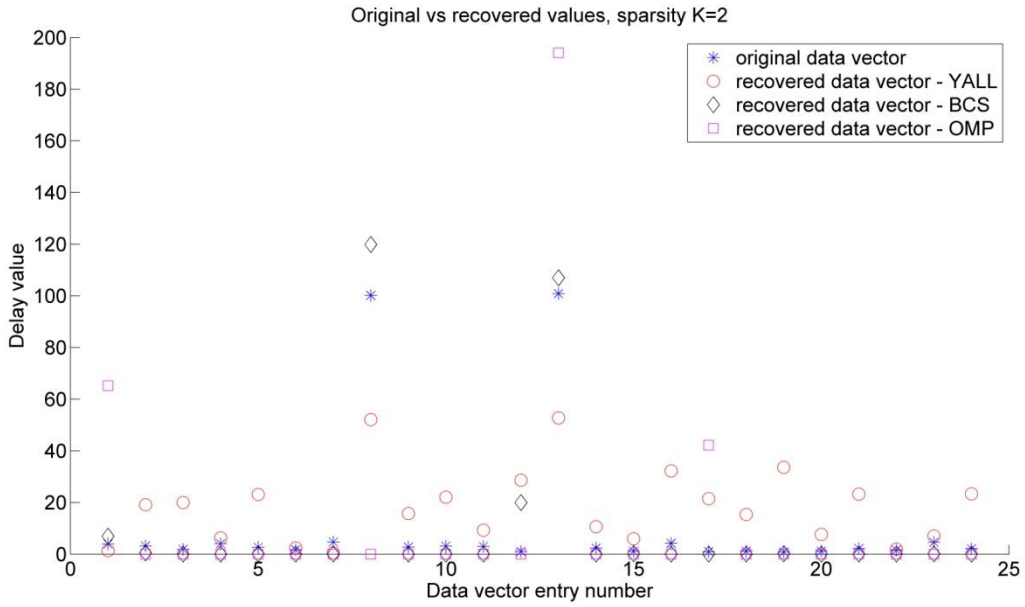


Fig. 6.11 Original vs recovered values example

It is the example of the reconstruction for three of the algorithms in the case K=2. The errors given for each of them are: YALL – 0.48 BCS – 0.17 OMP – 1.05. It is easy to notice based on figure 6.11 that in the case of BCS the recovery is close to the original data and therefore it gives the smallest error. In the case of OMP only one peak correct position is found but the amplitude is twice as big. The second peak is considered to be on entry number 1. When the absolute values of the vectors will subtract, high negative values appear. Euclidean norm is immune to negative values and therefore the error is big. So the error gives an insight of whether the peaks have been located or not, making it important in the decision making for the most suitable algorithms choice.

Conclusion

Even if reconstruction algorithms should use random sensing matrices for a good recovery, the comparison between recovery errors using random sensing matrices and recovery errors using random binary matrices shows that reconstruction algorithms work properly also using binary sensing matrices.

Selecting the best performing algorithms for testing them on the routing matrices (next section) is based on the recovery error for K=1 in the case of using binary sensing matrices. The error threshold was chosen 1 and the algorithms that recover with a smaller error are: Linprog, L1EQ, GPSR, BP, LASSO, L1LS and BCS.

6.6.2 Routing Matrices Testing

This part of the testing chapter is aimed to discover how the selected algorithms perform on the routing matrices, check the limits and conditions within which the recovery is achieved, and possibly perform more stringent selection of the algorithms. Since the routing matrix is one of the a priori known information in the Dragon-Lab problem, it is important to know the exact behavior of the algorithms when using the matrix.

Theoretical motivation for routing matrices testing

For exact recovery, a sensing matrix should satisfy the RIP property, as presented in the Compressive sensing section. Thus the RIP property guarantees *universal recovery*, in essence of any sparse data vector of sparsity K. However, it is proven [48] that recovery can be performed for more poorly conditioned matrices for smaller sparsities. Moreover, the reconstruction is achieved for a certain number of measurements, which is related to the dimension of the data vector and the number of ones per sensing matrix column, namely the weight of the column [50].

Additionally the reconstruction algorithms are influenced by the weight of the columns: Firstly, algorithms based on interior point methods [51], such as GPSR, BP or L1LS, require the columns of the matrix to be linearly independent [47], [45]; this is not the case of the routing matrices since they even have identical columns. On the other hand, in greedy algorithms [21], such as OMP or StOMP, a key step is to compute correlations of the measurement vector y with the columns of the sensing matrix in order to solve the equations systems [46].

Consequently, since the weight of the routing matrix columns is known to have influence on the algorithms reconstruction performance, it has been decided that an exhaustive testing to find out the exact influence of the columns weight will be conducted.

Practical motivation for routing matrices testing

Dragon-lab framework is aimed to be embedded in a network analysis software product. In this direction, the *routing matrices testing* provides an overview over how an algorithm for instabilities localization can be implemented as software application. Another motivation for extensive routing matrices testing is having the necessary recovery noise limits for the last section of this chapter, namely the reconstruction of the Dragon-Lab data vectors.

As a general observation, the routing matrix represents, in terms of traceroutes and nodes, the following:

- each matrix row represents one single traceroute; for example, the second row represents traceroute 2;
- the column number is equal to the node position in the data vector; for instance node 2 is mapped to column 2; if node 2 was present in the third traceroute then column 2 will have a "1" on the third row; consequently the statement "node 2 is involved in one traceroute" means that the routing matrix has a "1" in one of the positions of the second column.

Mathematically the mapping between the nodes and the columns of the routing matrix based on the traceroute is:

$$N_j \in x\ ,\ \forall\, j = \overline{1,n} \quad then \quad N_j \xrightarrow{mapped\ to} \Phi_{traceroute\_nr,j}, \forall\, traceroute\_nr = \overline{1,m}$$

,where N_j is a node in the data vector *x*, *j* is the index for matrix columns.

The test vectors for this section are chosen based on the idea of spread, consecutive or closely spaced peak entries; however, since the matrices are fixed and prior known, specific data sets are used in this second stage of testing.

According to table 6.4, for the current routing matrices only data vectors of sparsity K=1 may be recovered using the general recovery algorithms conditions. This signifies that in the practical case of Dragon-Lab, for each of the detected instabilities measured delay, most of this delay will have to be located on one node, involved in one of the traceroutes, and the rest of the nodes will have considerably smaller amount of delay distributed on them. However, in this study data vectors with sparsity K=2 will also be analyzed, considering that the position where the most part of the delay is located, relative to the routing matrix, makes a difference in the recovery process and thus even sparsity 2 can be recovered.

Depending of the path, the amplitude of the aggregate delay over the entire measurement period varies, as seen in table 6.6. This has an influence on how much difference there will be between the elements of the measurement vectors and consequently an influence on the values found in the data vectors, since each entry of the measurement vector is the sum of a combination of elements from the data vector.

Path	NTE - Cernet	Cernet - NTE	NTE - Auckland	Auckland – NTE
Maximum aggregate delay (s)	5000	5400	2100	2020
Variation (max-min) of aggregate delay (s)	3000	1500	250	200

Table 6.6 General path delay characteristics

The table and the traceroute data prove that the paths between NTE – Cernet, Cernet – NTE and the paths between NTE – Auckland, Auckland – NTE are very similar. This is the reason

matrix routing analysis will be conducted on only two of the four routing matrix, namely routing matrix of NTE – Auckland, day 11, and routing matrix of NTE – Cernet, day 35. They were designated in section 6.4 as the worst, respectively best case scenario matrices in respect to their theoretical characteristics.

Choice of test data vector values

For the NTE – Auckland path the maximum aggregate delay is about 2000 sec, as seen in table 6.6. The number of nodes on the path is 19. Assuming a lightly loaded network the delay on the nodes should be equally distributed. Since the sum of a combination of 19 hops from the delays in the data vector should equal around 2000 seconds, and it is desired that only one node has most of the delay and the rest are low values, basically noise compared to the main delay component, then the following values are chosen for the entries of the data vector: an average delay of 50 sec for 18 hops plus the main delay component of 1000 seconds on one hop. The sum will be around 1900 seconds. This is a value similar to the values existing in the measurements vector for this path NTE – Auckland. For this reason the basic test data vector will be chosen as in table A.5 from the Appendix, case K=1. For the case of K=1, the main delay component will be shifted on different positions within the vector. For the sparsity K=2, the three cases presented in the previous testing section, 6.6.1, peaks distanced, peaks consecutive and peaks closely spaced, will be used. Based on the same logic as for sparsity K=1, the values chosen for the entries of this test data vector will be: each of the high delay positions (two of them for K=2) will have 550 sec and the average delay on the other nodes (17 nodes) will be 50 sec. Over these basic values noise in different levels will be added. The noise will be added by subtracting a certain percentage from the highest component and spreading it equally over the rest of the components.

6.6.2.1 NTE – Auckland Routing Matrix Testing: Worst Case Scenario Routing Matrix

Based on the theoretical and practical motivation presented at the beginning of section 6.6.2, where routing matrix NTE – Auckland was designated the worst case scenario matrix, testing of this routing matrix is performed; the test data vectors are given in table A.1 from the Appendix. The section is split in several subsections in order to cover all reconstruction situations arising from structure of the matrix. As previously stated column weight affects reconstruction performance of data vector, consequently data vectors with different noise levels can be recovered. Based on the number of traceroutes the node is involved in, in essence the weight of the column corresponding to the node, and the sparsity of the data vector, the following scenarios were chosen:

1. Case NTE – Auckland K=1

1.1 Main delay component placed in a node corresponding to column weight 1;
1.2 Main delay component placed in a node corresponding to column weight 2;
1.3 Main delay component placed in a node corresponding to column weight 3;
1.4 Main delay component placed in a node corresponding to column weight 5.

2. Case NTE – Auckland K=2

2.1 Main delay components placed in distanced nodes;
2.2 Main delay components placed in consecutive nodes;
2.3 Main delay components placed in closely spaced nodes.

For both cases different levels of noise are added to the measurements vector, so as to test how the algorithms respond to various data and which of the algorithms work on the worst conditions. To have an estimate of the noise level, signal to noise ratio (SNR) will be calculated, where signal is considered the peak components of the data vector and noise are the rest of the components. It is defined as:

$$SNR = 10 * log_{10} \frac{\sum_{i=1}^{N} signal_i^2}{\sum_{i=1}^{N} Noise_i^2} \tag{6.15}$$

Moreover, the most important is that the algorithms localize the largest data vector component. The error is not so important as long as the position is correct. Therefore, for all the analyzed cases, tables will be made, showing whether the algorithms recover the main component position or not.

1. NTE – Auckland Case K=1

1.1 *Main delay component placed in a node corresponding to column weight 1*

The form of the routing matrix is very important. It is the one which influences recovery algorithms the most. In the case that the main delay component is located on one node which appears in only one of the traceroutes, a special situation arises. If the node appears in only one traceroute it means that the corresponding column from the routing matrix will have only a "1". It happens that this form of a column is not unique in the matrix. It means that there exists another node which appears only once in all traceroutes and during the same measurement. Consequently the two columns will be identical. It is observed that this situation determines the recovery algorithms to split the anomalous delay equally between the two nodes/columns. The only exception to this rule is Lasso, which finds the anomalous node as the first node in the series of nodes with identical columns. If the real anomalous node happens to be the first one in the series, then it means that Lasso works well. Otherwise, if the real anomalous node is the second one in the series of nodes with identical routing matrix columns, Lasso will still point to the first one in the series as being the anomalous node. An example of this behavior can be seen in table 6.7. The data vector used is the basic test vector. The table presents the recovered data vector, for each algorithm.

Node	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
7	41,694	25,062	34,975	27,314	0,000	32,641	2,402
8	133,413	130,659	104,741	131,584	344,148	116,684	132,743
17	45,670	41,121	45,048	36,806	0,000	18,483	18,610
18	18,042	21,214	2,024	25,991	0,000	36,859	4,215
19	122,825	103,313	106,987	92,805	0,000	40,232	99,100
20	367,545	303,232	234,746	316,978	950,187	343,340	262,439
21	0,000	0,000	0,000	0,000	0,000	-0,001	0,000
23	367,545	303,232	234,746	316,978	0,000	343,340	262,439

Table 6.7 Algorithms performance case 1.1

Due to space reasons some less important lines have been cut; it can be noticed in the "Node" column. It is observed that for most of the algorithms the largest part of the delay is distributed between nodes 20 and 23. This is because the original main delay component is placed on node 20 and the corresponding column for node 20 is identical to the ones for node 23. Therefore, the algorithms split the delay equally over the similar nodes. The exception is Lasso, which assigns most of the delay to the first node satisfying the column form. In this case Lasso gives the best result. The full table is found in Appendix, table A.4. There are 5 pairs of columns with one "1" per column with identical form in the routing matrix.

1.2 *Main delay component placed in a node corresponding to column weight 2*

On contrast to the previous case, there is just one pair of two identical columns with 2 ones: columns 3 and 18. If the main delay component is placed in node 18, similarly to the previous case, the main delay component is divided equally between nodes 3 and 18. Thus the main component is not detected even without noise, since it is unsure where it actually is located.

However, there are also unique columns with 2 ones. A challenging case for the recovery is when they are placed near columns with 3 ones for example, and this is why one of these cases was chosen for this testing. Consequently, the main delay component in the data vector was placed on the 16th node. Table 6.8 shows which algorithms can identify the position of the node with the main delay component, 16 in this case, for the introduced levels of noise. SNR = 23.6dB is the signal to noise ratio for the test vector with one high component and all the others having low values. This value is the highest SNR value based on the assumption of the chosen test vector values. Recovered peak localized is marked with "D"; not localized with "X". The next SNR value in the table, 13.27dB, is the value that marks the failure of the next algorithm, GPSR in this case. In essence all algorithms work until the lowest SNR of 13.27dB. Thus this type of tables emphasize the breaking point of each algorithm, depending on the noise level introduced over the data vector. It is noticed GPSR, Lasso and BCS stop recovering the peak position at a noise level of 9.39dB, whereas 4 others work up until 1.49 dB noise.

	Algorithm name						
SNR (dB)	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
23.60	D	D	D	D	D	D	D
13.27	D	D	X	D	D	D	D
9.39	D	D	X	D	X	D	X
1.50	X	X	X	X	X	X	X

Table 6.8 Algorithms performance case 1.2

1.3 *Main delay component placed in a node corresponding to column weight 3*

There are 3 columns which contain 3 ones. Two of them will be presented so as to see how much influence the sensing matrix has and to show the worst case scenario. Two of the columns are placed near columns with 4 ones and the other one is placed near a column with 2 ones. Worst case scenarios are always chosen.

The first case is when delay is placed in node 5, meaning column number 5. This is also the worst of the two cases because column 5 is placed near a column which contains 4 ones, namely column 6. The results for data vector with delay placed on position 5 can be seen in table 6.9. In this case the algorithms which fail first are again GPSR, Lasso and BCS.

For the second situation column 15 is selected to hold 3 ones; is has however lighter (fewer number of ones) neighbor columns; the results are given in table 6.10.

	Algorithm name						
SNR (dB)	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
13.70	D	D	X	D	D	D	D
12.86	D	D	X	D	X	D	D
9.96	D	D	X	D	X	D	X
3.33	X	X	X	X	X	X	X

Table 6.9 Algorithms performance case 1.3.1

	Algorithm name						
SNR (dB)	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
5.61	D	D	D	D	D	D	X
3.33	X	D	D	D	D	X	X
-0.11	X	D	X	D	D	X	X
-11.55	X	X	X	X	X	X	X

Table 6.10 Algorithms performance case 1.3.2

Lasso, BCS and Linprog fail first. It is noticed that the algorithms identify the anomalous node in much noisier data vectors. Consequently, it may be stated that the number of traceroutes the anomalous node is involved in and the number of ones in the neighbor columns, have an impact on the recovery performance.

***1.4** Main delay component placed in a node corresponding to column weight 5*

In this case the delay was placed in a node which appears in 5 out of 7 possible traceroutes. This means that in the routing matrix a column has 5 ones, column 8. The choice wants to show how influential the structure of the routing matrix is on reconstruction. The results are:

	Algorithm name						
SNR (dB)	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
9.61	D	D	D	D	X	D	D
9.70	D	D	D	D	X	D	X
-11.55	X	X	X	X	X	X	X

Table 6.11 Algorithms performance case 1.4

Intuitively, if a node is involved in most of the traceroutes and it is known that most of the delay is placed in it, the recovering should be very robust to noise. However, as it can be seen from the table above, two of the algorithms failed to recover the main component for a relatively low noise level. From what it has been observed in this chapter it would not be a mistake to make an even tighter selection of the algorithms and to consider Lasso and BCS for elimination. It remains to be seen how they will perform in the cases of the recovery of two main components and then the decision will be taken.

This case was analyzed so as to observe which algorithm is most general and for certain level of noise fits all possible scenarios; scenarios of where the anomalous node is located.

2. NTE – Auckland Case K=2

This case contains three sub cases based on the position of the main delay components within the data vector. Like in the previous case, the most important is the recovery of the position of the largest data vector components, under different levels of noise. Case K=2 resembles the testing performed in the previous subsection, which are the forms of the data vectors with respect to the position of the big components. However, the values of the delay placed in the data vectors are different and noise will be added. A second difference from case K=1 is the values of the largest entries in the data vectors. They will be considered 550 sec, for previously explained reasons. This makes the range of the variation, between small data vectors entries and large ones, about half of what it was in the previous case. A worse performance of the algorithms is expected.

***2.1** Main delay components placed in distanced nodes*

At this point it is known how the algorithms perform on the cases where only one component of the data vector is significantly larger. Intuitively, this sub case is like any of the sub cases from case K=1, put together. This sub case will be itself split into two sub cases:

- Columns have small number of ones and are placed relatively far from columns with many ones. In this case columns 10 and 20 fit. The result is the following:

	Algorithm name						
SNR (dB)	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
22.50	X	X	X	X	D	X	X
21.14	X	X	X	X	X	X	X

Table 6.12 Algorithms performance case 2.1.1

Even for the lowest level of noise only one algorithm detects the position of the highest entries of the data vector. The case of columns with small number of ones and positioned close to columns with large number of ones is obvious.

- Columns have large number of ones; for this case, columns 8 and 15 have been selected; and results are in table 6.13; similar results are obtained in case of columns 8 and 12.

	Algorithm name						
SNR (dB)	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
19.36	D	D	D	D	D	D	D
16.54	D	D	D	D	X	D	X
-20.40	X	D	D	D	X	D	X
-21.37	X	X	D	X	X	X	X

Table 6.13 Algorithms performance case 2.1.2

It can be noticed that two of the algorithms, Lasso and BCS, start giving erroneous results even with a high SNR. This is considered the most favorable case for sparsity 2, spread anomalous nodes.

***2.2** Main delay components placed in consecutive nodes*

If the columns have small number of ones and are positioned far/close from/to the columns with many ones, none of the algorithms detected the biggest data vector entries. In the case when the consecutive columns are one holding many ones and another next to it with few ones, like in the case of columns 8, 9, the results are the following:

	Algorithm name						
SNR (dB)	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
24.13	D/2	D/2	D/2	D/2	X	D/2	X

Table 6.14 Algorithms performance case 2.2

Most of the algorithms detect the biggest entry placed on the column with many ones. However they do not detect the entry set on the columns with fewer ones, at least not completely. They detect two identical smaller components, corresponding to two identical columns: columns 9 and 22. It is a situation similar to case K=1, sub case 1.1.

***2.3** Main delay components placed in closely spaced nodes*

In this case columns 8 and 11 are selected. Column 8 contains 5 ones, whereas column 11 just two and is next to a column with 4 ones. The performance of the algorithms is:

	Algorithm name						
SNR (dB)	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
23.60	D	D	D	D	D	D	D
19.11	D	D	D	D	X	X	D
14.57	D	D	D	D	X	X	X
10.77	X	X	X	X	X	X	X

Table 6.15 Algorithms performance case 2.3

Lasso, BCS and L1LS fails first, at a fairly high SNR. They do not identify the position of node 11 as one of the peaks.

In the case of NTE – Auckland routing matrix another interesting comparison can be made; a comparison between how the 7 selected algorithms perform, in terms of error as defined in (6.14), on the randomized binary matrix and on the routing matrix, using the same data vectors for testing, namely the three sets defined in section 6.6.1. The results for the routing matrices are shown in figure 6.12.

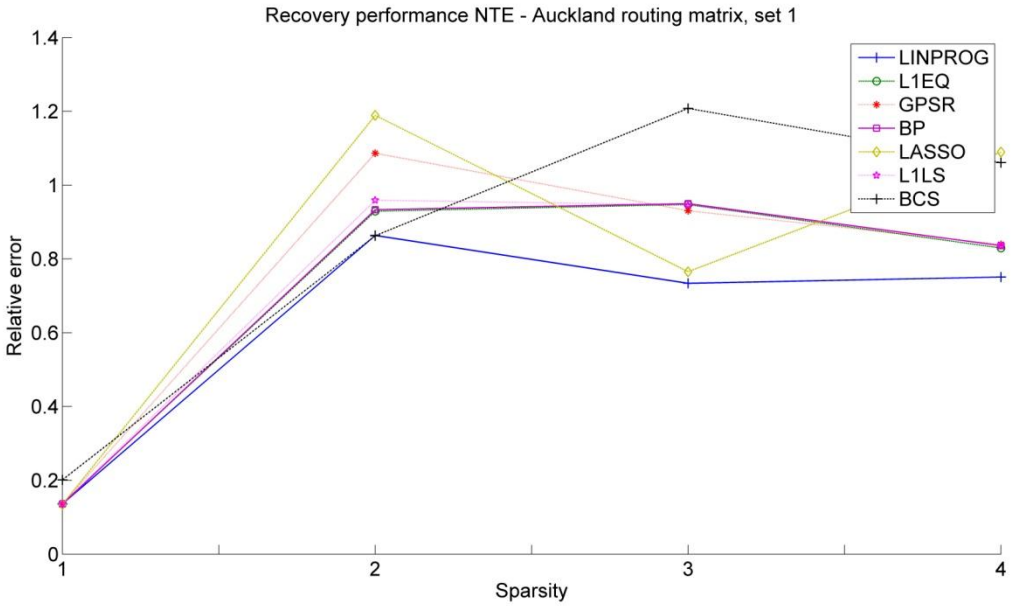


Fig. 6.12 a) Recovery performance NTE – Auckland routing matrix

The plots in figure 6.12 a) and b) have to be compared to the lower half of figures 6.8, 6.9 and 6.10. The algorithms reconstruct the same test vectors, but using two different binary matrices. The differences are:

Data set 1: for the case of the random binary matrix all algorithms produced errors below 1; for the routing matrix BCS and LASSO give error 1.2 for K=3, K=2 respectively. However for K=1 errors they are the same, low, and for K=2 errors remain in the same range as for the

random matrix. Thus the use of the routing matrix does not affect much the reconstruction for this data set.

Data set 2: for K=1 errors remain the same; for K=2 they increase from 0.2 to 0.5; for K=3, they go up from approximately 0.25 to 0.7, with the exception of Lasso. Generally, the increases in errors are significant, but the error remains below 0.8. Once more the algorithms perform well also on the routing matrix.

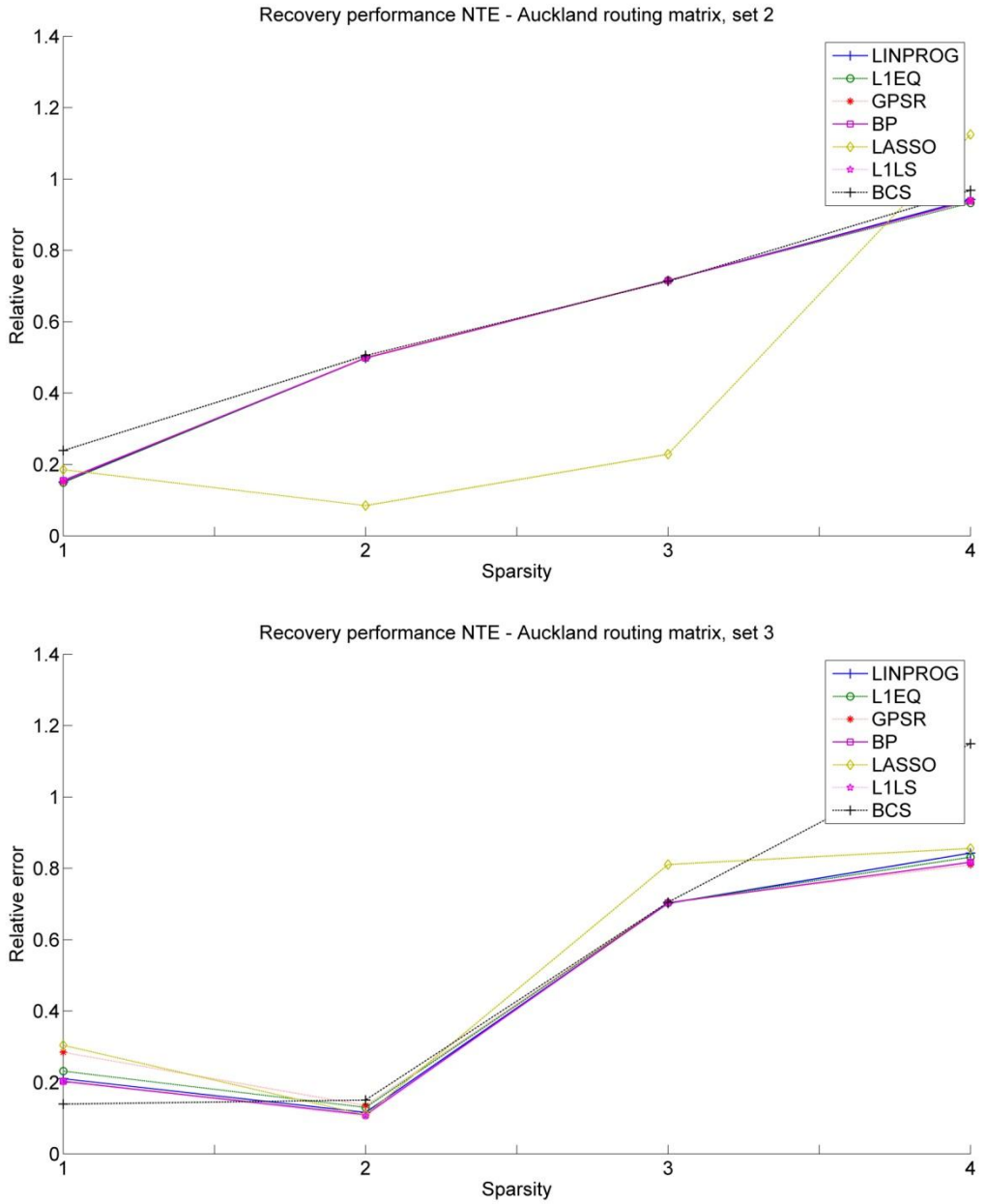


Fig. 6.12 b) Recovery performance NTE – Auckland routing matrix

Data set 3: for K=1 errors decrease significantly and all algorithms reconstruct around error=0.2; for K=2 errors decrease again compared to the use of the random binary matrix; for K=3 the errors slightly increase. However, in this case the routing matrix performs even better then the random binary matrix.

These observations show that the use of NTE – Auckland routing matrix does not deteriorate the reconstruction process, meaning that it may be used as sensing matrix compatible with the selected algorithms.

Conclusion NTE – Auckland Routing Matrix Testing: Worst Case Scenario Routing Matrix

Firstly: the algorithms perform differently based on the position of the instabilities in the data vectors, because the routing matrix is not completely random. The routing matrix for path NTE – Auckland has some pairs of identical columns, which makes recovery of one peak position impossible; however it is known that the peak is situated in one of the maximum recovered data vector values.

Secondly: the other parameter involved in the performance of the algorithms is the noise applied to the data. Noisy data vector yields a noisy measurements vector. The measurement vectors in Dragon-Lab are noisy; the ones for paths NTE – Auckland, Auckland – NTE have almost equal component values being the worst situation possible.

For sparsity K=1 the worst case scenario is for the lowest accepted SNR of 9.96 dB, and the algorithms which perform best are Linprog, L1eq, GPSR, BP and L1ls. They will be used in the last section of this chapter, namely the reconstruction of data vectors based on the measurement vectors.

6.6.2.2 NTE – Cernet Routing Matrix Testing: Best Case Scenario Routing Matrix

Path NTE – Cernet is considerably different from the previously analyzed path. First and foremost, this path contains only 15 nodes in comparison to the previous which had 19 nodes, and the nodes' IPs do not vary much. This creates a shorter routing matrix than the previous, which can be an advantage since the ratio unknowns/number of measurements is smaller. The routing matrix for NTE – Cernet path is the best with respect to the mutual coherence, thus the algorithms should perform better, without much influence from positioning of the ones on columns.

A problem encountered for this matrix is its rank. Linprog stops working because the large scale solver, LIPSOL, used by Linprog, requires that the matrix has full structural rank. The rank of this 7 x 6 matrix is 4, in comparison to the NTE – Auckland 7 x 24 matrix which was 7. A modification of the recovery algorithm used by Linprog is possible, but there would not be a comparison to the previous case and would be a loss of generality.

Another difference from the previous path is, according to table 6.6, the maximum aggregate delay. Consequently, there will be higher variations and the measurements vector is less noisy than the one for path NTE – Auckland. The test data vectors are chosen based on the same idea, except that the values should be increased such as to produce measurement vectors in the order of 3300 sec; there are 15 nodes on the whole path. The test data vectors for this routing matrix will be the ones in the table 6.16.

case 1.1	case 1.1 HN	case 2.1	case 2.2
125,69	207,04	128,96	126,27
133,77	250,52	700,00	133,09
132,63	212,80	139,88	133,97
140,31	209,47	135,00	132,32
1.400,00	280,00	134,60	700,00
128,74	206,80	700,00	700,00

Table 6.16 Test data vectors NTE - Cernet

Testing of the routing matrix of the path NTE – Cernet, for measurement day 35, is done, on the same procedure as in the previous subsection, using the test data vectors from table 6.16, in the following way:

1. NTE – Cernet Case K=1

1.1 Main delay component placed in a node corresponding to column weight 3;
1.2 Main delay component placed in a node corresponding to column weight 4;

2. NTE – Cernet Case K=2

2.1 Main delay components placed in distanced nodes;
2.2 Main delay components placed in consecutive nodes.

1. NTE – Cernet Case K=1

***1.1** Main delay component placed in a node corresponding to column weight 3*

Compared to the previous routing matrix, this one does not have identical columns and the minimum number of ones per columns is 3, maximum 4. This is shown also by the lower mutual coherence value. Consequently, it is expected that the algorithms recover better the data vector. If node 5 holds the anomalous delay, the results are:

	Algorithm name			
SNR (dB)	L1eq	GPSR	BP	L1ls
-14.40	D	D	D	D
-15.44	D	X	D	D
-15.23	X	X	X	X

Table 6.17 Algorithms performance case 1.1

Even for a very noisy data vector, like the one in table 6.16, case 1.1 HN, the main delay component position is recovered.

***1.2** Main delay component placed in a node corresponding to column weight 4*

The second node in the test vector is chosen to have the main delay component. As in the previous sub case and as expected high level of noise is allowed. The results are:

	Algorithm name			
SNR (dB)	L1eq	GPSR	BP	L1ls
-14.43	D	D	D	D
-14.77	X	X	X	X

Table 6.18 Algorithms performance case 1.2

The conclusion for this case is that any of the four algorithms can recover the data vector if it contains only one important aggregate delay component, independently of the position in the data vector and for high noise levels.

2. NTE – Cernet Case K=2

Since the routing matrix columns are different, two sub cases will be analyzed: the anomalous nodes are positioned away from each other and secondly, they are consecutive nodes. The test vectors for sub cases 2.1 and 2.2 can be seen in table 6.16. Since there are only 6 nodes involved in this routing matrix out of a total of 15 on the entire path, it is less probable that 2 of them will be anomalous. The case is tested anyway. All algorithms are strong to noise and recover the test data vectors' position of main delay component.

***2.1** Main delay components placed in distanced nodes*

	Algorithm name			
SNR (dB)	L1eq	GPSR	BP	L1ls
-5.44	D	D	D	D
-5.81	X	X	X	X

Table 6.19 Algorithms performance case 1.2

***2.2** Main delay components placed in consecutive nodes*

	Algorithm name			
SNR (dB)	L1eq	GPSR	BP	L1ls
-5.55	D	D	D	D
-5.82	X	X	X	X

Table 6.20 Algorithms performance case 2.2

Conclusion NTE – Cernet Routing Matrix Testing: Best Case Scenario Routing Matrix

Lasso, which gave poor results on the NTE - Auckland routing matrix does not work at all on this routing matrix; therefore, eliminating it is supported. The remaining algorithms perform well in recovering the positions of the largest delay component/s in any possible case and with

high noise level, that is low SNR; the routing matrix of NTE – Cernet path can be considered as having suitable properties for the selected algorithms.

Conclusion Routing Matrices Testing

The two routing matrices tested in this section are representative for all the four paths, NTE – Cernet, Cernet – NTE, NTE – Auckland, Auckland – NTE, measured within this research. This section gives and insight on how the recovery algorithms perform on these two matrices, setting limits such as which algorithms can be used for obtaining a certain recovery performance.

It was decided that the algorithms which perform best using any of the two routing matrices are: L1eq, GPSR, BP and L1ls. They will be used on the recovery of the real data vectors in the following section, 6.6.3.

6.6.3 Dragon-Lab Data Vectors Reconstruction

This stage of the testing chapter is dedicated to recover the position of the anomalous delay entries using the Dragon-Lab measurement vectors, and the two tested routing matrices. Thus the section will be divided into two subsections: NTE – Auckland, day 11 data and NTE – Cernet, day 35 data. The two cases are different because:

- NTE – Auckland routing matrix has different properties from the NTE – Cernet one;
- NTE – Auckland measurement vector is more noisy than the NTE – Cernet one.

The measurement vectors for the two paths are in the first two columns of table 6.21. The difference in values can be noticed. All values represent aggregate delay and are measured in seconds.

NTE – Auckland Day 11	NTE – Cernet Day 35	Reconstructed NTE – Auckland Day 11, L1eq	Reconstructed NTE – Auckland Day 11, GPSR	Reconstructed NTE – Auckland Day 11, BP	Reconstructed NTE – Auckland Day 11, L1ls
1930,86	3821,48	1930,86	1935,78	1930,86	1930,87
1929,15	3186,15	1929,15	1923,27	1929,15	1929,15
1930,13	2974,13	1930,13	1926,59	1930,13	1930,13
1933,42	3329,01	1933,42	1927,40	1933,42	1933,42
1933,37	3684,12	1933,37	1930,49	1933,37	1933,36
1932,66	4749,98	1932,66	1931,37	1932,66	1932,66
1934,81	2778,02	1934,81	1930,08	1934,81	1934,81

Table 6.21 Measurement vectors

Methodology

The each of the two paths, the scheme for testing is:

- recover data vectors; calculate SNR based on definition (6.15) in order to have an empirical guarantee that the algorithms recover the correct vector;
- calculate y_reconstructed as routing matrix * x_reconstructed; compute reconstruction errors for cases K=1 and K=2;
- analyze detected peak values and nodes in order to give a validation for the method.

Two errors have been defined in order to characterize the performance of the reconstruction based on the measurement vector. As mentioned above y_reconstructed is calculated. Since the only certain information available is the routing matrix and the real measurement vector y, two errors have been defined based on y. The first one is (6.16) and represents the relative difference between original y and reconstructed y.

$$E1 = \frac{\| routing\_matrix * x\_reconstructed - y \|_2}{\| y \|_2} \tag{6.16}$$

The second one estimates the difference between original y and reconstructed y after x_reconstructed has been sparsified; meaning only the peak value entry is kept, or two of them for K=2. The formula is:

$$E2 = \frac{\| routing\_matrix * x\_reconstructed\_sparse - y \|_2}{\| y \|_2} \tag{6.17}$$

6.6.3.1 NTE – Auckland Data Vector Reconstruction

For this case E1 and E2 are presented in table 6.22, for each algorithm. Both sparsities, K=1 and K=2, are presented together for a better comparison. The recovered data vector is included in table A.9, the K=1 sparse and K=2 sparse vectors in table A.10. Y reconstructed can be seen in table 6.21 for all algorithms. The error between original y and reconstructed y is very small. Thus algorithms recover a possible x that yields an almost original y.

Algorithm / Sparsity		L1eq	GPSR	BP	L1ls
E1 (%)		1.089e-10	0.0023	3.904e-12	1.608e-06
SNR (dB)	K=1	-8.90	-5.55	-9.33	-9.71
E2 (%)		0.7265	0.6961	0.7302	0.7335
SNR (dB)	K=2	-2.70	-0.15	-2.91	-3.12
E2 (%)		0.5895	0.5320	0.6493	0.5932

Table 6.22 NTE – Auckland measurement vector recovery results

The SNR of the recovered data vector is minimum -9.71 dB. In the previous section it was shown that the 4 best algorithms identify a data vector with sparsity K=1 for a minimum SNR of 9.96 dB in the situations where the peak value entry is not situated on a column with just

one “1”. However, in this case the reconstructed gave node 8 as solution for the highest peak. For this case the algorithms require a minimum of -11.54 dB to give a correct recovery. This means that the result for K=1 for algorithms L1eq, BP and L1ls is within functional noise limits

In the K=2 case, there are many more possibilities. The algorithms give, for the reconstruction of the measurement vector, different results. Two of them point to nodes 8 and 12 and the problem nodes, and one of them to nodes 8 and 15. The minimum SNR calculated for the recovered data vector is -3.12 dB, which is within the noise limits for these two cases. If these two nodes are indeed the ones causing excessive delay, then the four algorithms recover them correctly. In other cases higher SNR is required to guarantee a trusty reconstruction.

NTE – Auckland Reconstruction Validation Approach

It is difficult to validate these results. However one possible attempt to do it is to check the recovered data vectors in combination with information from the traceroutes delays. The information is not sufficient and the delay measurements in traceroutes are not accurate, but it is a possibility.

Traceroute Data

The traceroute presents the situation at a given moment in time. The moment might not even be included in the time-bin detected as anomaly. If it is not, then the previous traceroute was considered. The only way to correlate the available data (traceroutes, aggregate delay) with the detected anomalous nodes is to analyze the traceroute RTT delay given in the traceroutes. Traceroute data is known to be inaccurate: erroneous routes might be returned mainly due to load-balancing routers which might determine the appearance of false loops, cycles and diamonds, which do not represent Internet topology [43]. Moreover, traceroute delay is not end-to-end delay, like the ones used for instabilities detection, but it is assumed that there is no problem on the return path. It is proven, [44], that the increase in RTT is more likely to be due to a problem in the forward path, because of the asymmetry of the Internet routes.

In the case K=1, the node detected as having all delay is node 8. This node corresponds to the column with the most number of nodes, so it could be that: it really holds an anomaly or is detected because of the weight of the column. Node 8 corresponds to column 8, which corresponds to IP-ID 21 (the column in the original routing matrix), which is in fact IP: 154.54.31.245. This IP is found on some of the traceroutes for the day on position 6/hop 6 on the paths. By checking this position for any delay abnormality the following is found:

```
Wed, 30 Jun 2010 13:45:22 GMT
Unix time: 1277905522
traceroute to 130.216.5.81 (130.216.5.81), 30 hops max, 60 byte packets
 1  178.164.4.1  0.337 ms  0.416 ms  0.402 ms
 2  217.168.93.101  8.132 ms  8.236 ms  8.223 ms
 3  149.6.116.17  39.296 ms  39.401 ms  39.388 ms
 4  130.117.2.181  24.237 ms  24.458 ms 130.117.2.177  24.327 ms
 5  130.117.49.209  33.815 ms 130.117.49.221  33.800 ms 130.117.49.209  34.137 ms
 6  130.117.51.230  124.930 ms 154.54.31.245  129.116 ms 130.117.51.230  124.822 ms
```

Hops at position 5 and hops in position 6 have a significant delay difference between them, compared to the other transitions. Moreover, these IPs belong to the same ISP: Psinet Inc. from Washington DC. However, all delays differences between hops 5 and 6 are around 100 ms, so there is no abnormal behavior.

In contrast, if two highest delay values of the recovered data vector are considered, K=2 (see table A.10 for reconstructed x), abnormal situations appear on the traceroutes, or on traceroutes before the detected anomalous time-bins. As second highest delay L1eq and L1ls detect node 12, GPSR node 6 and BP node 15.

Node 12	IP ID: 28	IP: 130.117.2.177
Node 6	IP ID: 10	IP: 154.54.30.190
Node 15	IP ID: 36	IP: 154.54.30.186

Table 6.23 NTE – Auckland node mapping

Nodes 6 and 15 are related since their IPs belong to the same hop number in the traceroutes paths, namely position 10. For two of the traceroutes an abnormal behavior is observed, that may confirm anomaly happens. The normal delay values for the day for hops 8, 9 and 10 are:

```
 8  154.54.28.254  155.857 ms  155.835 ms 154.54.2.222  155.708 ms
 9  154.54.0.141  191.349 ms 154.54.0.245  191.846 ms 154.54.0.253  191.853 ms
10  154.54.3.142  191.034 ms 154.54.30.190  191.359 ms  191.343 ms
```

But for these traceroutes:

```
Wed, 30 Jun 2010 13:45:22 GMT
traceroute to 130.216.5.81 (130.216.5.81), 30 hops max, 60 byte packets
8  154.54.2.246  155.492 ms 154.54.5.53  155.388 ms 154.54.29.14  151.422 ms
9  154.54.0.253  244.102 ms 154.54.0.141  191.401 ms  191.509 ms
10 154.54.30.190  191.309 ms 154.54.3.142  191.054 ms 154.54.30.190  191.266 ms
```

The excessive delay does not appear on hops 10, but it appears at one of hops 9. Since traceroute is not exact and this traceroute is just before the detected anomalous time-bin

traceroute, it is possible that this excessive delay found on hops 9 is a prediction of delay on node 15 or on node 10 in the next 2 minutes time-bin. On the traceroute:

```
Wed, 30 Jun 2010 16:16:30 GMT
traceroute to 130.216.5.81 (130.216.5.81), 30 hops max, 60 byte packets
 8 154.54.28.254  331.579 ms 154.54.2.222  331.572 ms 154.54.29.14  332.623 ms
 9 154.54.0.141  256.356 ms 154.54.0.249  191.271 ms 154.54.0.245  191.288 ms
10 154.54.30.190  191.393 ms  191.270 ms  191.388 ms
```

Excessive delay appears on IP: 154.54.0.141, which is node 11, IP-ID: 27, which is next to node with IP-ID: 28 in the routing matrix. Algorithms detect column 12, being heavier, but as shown here, the problem is right next to it. So even if the localization is not perfect, problems might lay one or two hops away counting on the routing matrix columns. This traceroute also suggests that it is possible to have delay on hops 10 in a future time-bin, since there is abnormal behavior delay on hops 8 and 9.

Normal traceroutes behavior for IP-ID: 28, column 12, IP 130.117.2.177, position in traceroute paths: 4, is the following:

```
Wed, 30 Jun 2010 16:16:30 GMT
traceroute to 130.216.5.81 (130.216.5.81), 30 hops max, 60 byte packets
 1  178.164.4.1  0.301 ms  0.388 ms  0.374 ms
 2  217.168.93.101  8.219 ms  8.205 ms  8.431 ms
 3  149.6.116.17  8.527 ms  8.514 ms  8.502 ms
 4  130.117.2.181  24.444 ms 130.117.2.177  24.312 ms 130.117.2.181  24.652 ms
```

However, for two traceroutes abnormalities appear:

```
Wed, 30 Jun 2010 16:14:29 GMT
 1  178.164.4.1  0.362 ms  0.327 ms  0.420 ms
 2  217.168.93.101  8.147 ms  8.256 ms  8.703 ms
 3  149.6.116.17  21.236 ms  21.221 ms  21.319 ms
 4  130.117.2.177  24.475 ms  24.460 ms 130.117.2.181  24.328 ms
Wed, 30 Jun 2010 13:45:22 GMT
 1  178.164.4.1  0.337 ms  0.416 ms  0.402 ms
 2  217.168.93.101  8.132 ms  8.236 ms  8.223 ms
 3  149.6.116.17  39.296 ms  39.401 ms  39.388 ms
 4  130.117.2.181  24.237 ms  24.458 ms 130.117.2.177  24.327 ms
```

The regular delay for hops 3 is around 8.5 msec. For these two cases the delay is increased. It is possible that a few minutes later excessive delay will also appear on hops 4, or that, once more, the peak is detected because column 12 has 4 ones.

Conclusion

The algorithm which retrieves the lowest reconstruction error for sparsity K=1, is BP. It recovers a data vector with the peak position the same as for the other three algorithms. Validation of the correct recovery of the peak position is supported by the occurrence of abnormally high delay on the traceroutes involved in the instabilities for the recovered node position. However, abnormal delay is registered on several paths during the day, therefore it cannot be 100% confirmed that the instability is real.

6.6.3.2 NTE – Cernet Data Vector Reconstruction

There is much difference between NTE- Cernet and NTE- Auckland: the measurement vector is more varied in values, and the matrix is more uniform regarding the weight of the columns.

For this case E1 and E2, for each algorithm, are given in table 6.22. The recovered data vector is included in table A.9, the K=1 sparse and K=2 sparse vectors in table A.10. The error between original y and reconstructed y is a lot larger than the one for NTE – Auckland. Especially for L1eq, 33% is surprisingly high. It reconstructs an x that yields a y far from the original y.

Algorithm / Sparsity		L1eq	GPSR	BP	L1ls
E1 (%)		0.3383	0.0825	0.0825	0.0825
SNR (dB)	K=1	-0.86	-9.59	-8.04	-9.61
E2 (%)		0.7522	0.7497	0.7368	0.7495
SNR (dB)	K=2	8.04	0.56	1.60	0.55
E2 (%)		0.5256	0.5984	0.5884	0.5980

Table 6.24 NTE – Cernet measurement vector recovery results

The SNR of the recovered data vector is minimum -9.61 dB, which is very low, but in section 6.6.2 it was shown that all algorithms identify a data vector with sparsity K=1 for a very low minimum SNR; the algorithms in that section were tested only until SNR -14.43 dB, but it is possible that they recover even noisier data. This limit is safely larger than -9.61 dB, so the algorithms perform within limits. Once again this result is due to the fact that the routing matrix does not have any similar columns. In the K=2 case, the noise limits are also within the ones found in section 6.6.2.

NTE – Cernet Reconstruction Validation Approach

For a validation of the results, the traceroutes for this day are inspected, with the same observation on accuracy as in the previous case.

Both sparsity cases will be analyzed simultaneously. The nodes, columns, IP-IDs, IPs, traceroute hops rows involved in the detected data vectors are structured in table 6.25:

Column/Node	IP-ID	IP	Traceroute hop nr
3	7	217.239.40.117	7
5	17	130.117.14.106	6
1	5	130.117.49.241	5

Table 6.25 NTE – Cernet node mapping

The normal traceroute behavior for hop numbers 4 to 8 is:

```
Wed, 28 Jul 2010 05:22:11 GMT
traceroute to 203.91.120.141 (203.91.120.141), 30 hops max, 60 byte packets
 4  130.117.2.177  24.282 ms  24.383 ms  24.487 ms
 5  130.117.49.241  33.857 ms  33.959 ms 130.117.49.245  33.940 ms
 6  130.117.14.106  33.687 ms 130.117.14.150  33.736 ms 130.117.14.106  33.637 ms
 7  62.154.14.58  310.200 ms 217.239.40.117  297.347 ms 62.154.14.58  309.381 ms
 8  62.153.203.206  367.670 ms  379.533 ms  379.520 ms
```

There is a high increase from hop 6 to 7, but it is normal increase over the day, hops 6 are located in USA and hops 7 in Germany. For the traceroutes involved in the measurements there is only one observed abnormal change:

```
Wed, 28 Jul 2010 16:54:46 GMT
traceroute to 203.91.120.141 (203.91.120.141), 30 hops max, 60 byte packets
 4  130.117.2.177  24.259 ms  24.482 ms  24.581 ms
 5  130.117.49.241  33.834 ms  34.057 ms  34.160 ms
 6  130.117.14.150  33.665 ms  33.669 ms 130.117.14.106  33.773 ms
 7  62.154.14.58  323.574 ms  322.155 ms 217.239.40.117  313.844 ms
 8  62.153.203.206  393.244 ms  367.831 ms  393.154 ms
```

There is slightly increased delay on IP: 217.239.40.117, which is detected as having the peak delay entry by all algorithms. In this traceroute there is increased delay on hops 7 and on hop 8; but hop 8 does not have an entry in the data vector.

Conclusion

The lowest general recovery error is given by L1eq and for K=1 sparsified (make vector components zero except for the peak component) data vector the lowest recovery error is given by BP. Lowest error for sparsity K=2 is given by L1eq. However, compared to the previous path, the anomalous nodes detected on this path cannot be confirmed by traceroute check.

Conclusion Dragon-Lab Data Vectors Reconstruction

The first target of this section was to recover the Dragon-Lab data vectors. In order to theoretically ensure the correct reconstruction, the algorithms have to recover data vectors within the noise levels limits determined in the previous testing section.

The second target of this section was the validation of the recovery, basically to show that nodes flagged as causing instabilities actually cause them.

It was observed that based on the noise levels results from the previous section and on the recovery results from this section, algorithms work within functional noise limits, therefore they are supposed to give proper results. The best performance algorithms are: L1eq, BP and L1LS. GPSR gives slightly higher reconstruction errors.

The validation approach was conducted by inspection of the traceroutes RTT delay in order to find anomalies in delay for the flagged nodes. The validation is inaccurate but it confirms anomalies in some of the tested cases.

6.6.4 Localization algorithm

The Dragon-Lab is aimed to be implemented as network analysis software and the localization technique is an important feature. Therefore an algorithm is needed for the implementation of the localization function.

The two previous sections: testing of recovery algorithms on routing matrices in order to find out the noise levels for a good recovery on various cases, and recovery of the data vectors based on the measurement vectors knowing the applicability conditions for the reconstruction algorithms, offer an overview of how an unstable states localization algorithm should be designed. The scheme of the algorithm can be seen in figure 6.13.

Basically the stages of the algorithms are:

1. select measurement day; select test data vector; construct routing matrix; construct measurement vector.
2. test routing matrix using a reconstruction algorithm in order compute the empirical noise limits for which recovery is guaranteed; empirical noise limits are calculated as in routing matrices testing section; also based on the previous section the best performing reconstruction algorithms are L1eq, BP, and L1LS; any of them can be chosen as reconstruction algorithm.
3. perform recovery of data vector; verify algorithm functional noise limits for the detected positions and confirm successful/unsuccessful localization.

As a task for future work the algorithm will have to be implemented to produce a software application.

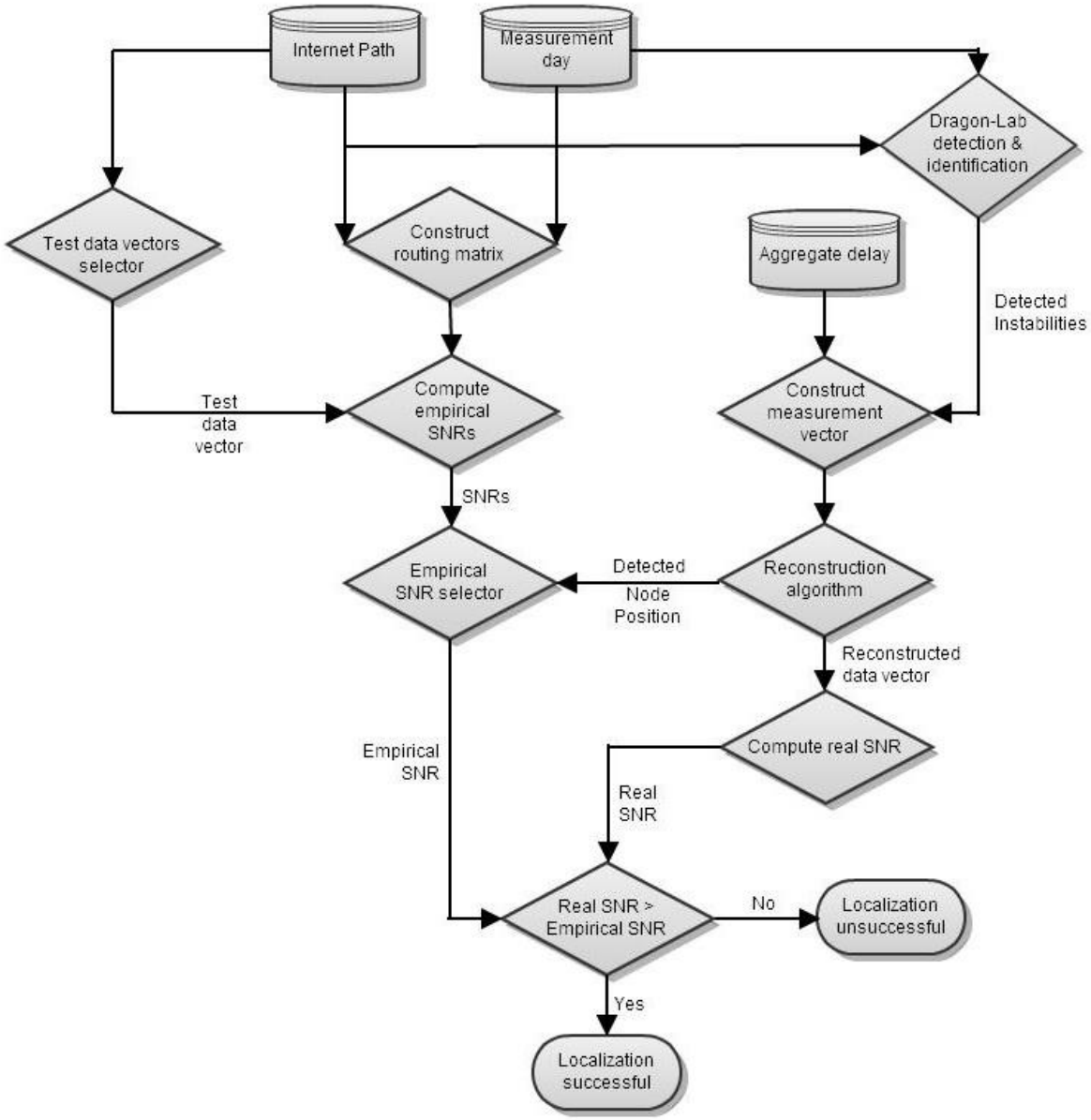


Figure 6.13 Localization algorithm

6.7 Conclusion

This chapter has shown that Compressive sensing reconstruction algorithms can successfully be applied on Dragon-Lab data in order to attain spacial localization of network instabilities.

The motivation for using Compressive sensing reconstruction algorithms was:

- the general form of the Dragon-Lab problem, in essence that it can be arranged as system of equations;
- the presence of an underdetermined linear system of equations;
- the compressibility of Dragon-Lab data vector.

The algorithms for the testing were selected based on:

- the capability of solving problems similar to the Dragon-Lab problem form: $y = \Phi x$;
- the general characteristics of the sensing matrix they use;
- the capability to recover compressible data vectors.

The methodology for the use of the algorithms in order to recover the node/s causing instabilities has been done in three stages with the following results:

- selection of the best algorithms that work on random sensing matrices as well as on binary random matrices; it is aimed that the positions of the data vector main delay components are recovered; the selection is done based on sparsity 1 vectors recovery; the best algorithms selected were: Linprog, L1EQ, GPSR, BP, LASSO, L1LS and BCS;
- replacing of the binary random matrices with the worst and best case scenarios Dragon-Lab routing matrices and selection of the best reconstruction algorithms; selection was based on the tolerated noise limit of the data vector for proper recovery for sparsities 1 and 2; the algorithms chosen to work best using routing matrices were: GPSR, BP, LASSO and L1LS;
- recovery of Dragon-Lab data vectors, basically recovering the positions of the nodes that cause instabilities, has been performed using the measurements vectors, routing matrices and the selected algorithms; taking into consideration the noise limits determined in the previous stage it can be theoretically guaranteed that the algorithms recover correctly at least one position of the data vector; the algorithms that give the lowest recovery error are L1eq, BP and L1LS; validation of the results has been conducted using traceroute data, which is not conclusive in all the verified cases.

The above methodology, used with the target of recovering data vectors under theoretical signal to noise ratio conditions, formed the basis for the proposal of a localization algorithm. Consequently, Dragon-Lab can be implemented as software application for automatic unstable nodes localization.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

As IP networks have become the support of an increasingly varied range of applications, from games, video streaming, to e-commerce and online banking, it is critical to understand the functioning and performance bounds of the network in order to provide a certain QoS. This understanding is based on measurements of various metrics and parameters for being able to control and manage the network. Although many studies have been conducted on Internet measurement the problem of the performance of its paths is still little understood.

As presented in Chapter 2, existing techniques that aim to diagnose network paths performance are focused in two directions:

- they either analyze end-to-end delay in order to achieve as accurate measurements as possible and create delay distributions [1]; or they use end-to-end delay measurements to infer the delay distribution within the network [2] or for bandwidth estimation, admission control and other decision making schemes [3]; these are generally Internet wide measurements;
- or they analyze failures within the network, based on protocols routing information [4], [5], syslog messages, administrator e-mails, router configurations [6], [8], to be able to localize the failure; these techniques are conducted on large scale networks and mainly suffer from scalability problems if applied Internet wide.

Chapter 3 presented Dragon-Lab as a new, alternative approach for Internet paths behavior investigation and for identifying and localizing Internet paths instabilities. The technique is based on detecting and identifying significant unusual changes in the packet delay and packet loss. In this way there is no need for further information regarding the structure, operations or events in the Internet backbone from the ISPs side. The problem is however challenging given the fact that huge amounts of noisy data have to be processed in order to extract meaningful information.

The performance of detection and classification of network instabilities has been analyzed in Chapter 4. It was shown that the framework achieves an overall detection rate of the instabilities of around 90%. The detection rate depends on the type of unstable network states which has to be detected and on the choice of the tuning parameter involved in the principal component pursuit program.

Chapter 5 provided the methodology for Dragon-Lab output information analysis, in order to give a suggestive overview of the behavior of the detected instabilities. Moreover, it offered the support for future Internet paths simulations with respect to the distribution of instabilities. Compared to the techniques presented in Chapter 2, Dragon-lab accomplishes path analysis based only on two metrics: packet delay and packet loss. Additionally, general metrics, such as: availability, fatigue or stability, have been defined in order to give an overview of the impact of the combination of instabilities per path.

Chapter 6 completes the network troubleshooting capabilities of Dragon-Lab with the introduction of instabilities spacial localization. Localization of instabilities, in essence identify the unstable nodes, is a challenge in Internet paths analysis based on only end-to-end delay, packet loss information and traceroutes. The results, performance, conditions and limits of achieving this goal were presented in this chapter. It has been shown that localization is possible under certain conditions. Following the methodology for unstable nodes positions recovery an algorithm for automatic localization has been proposed. The only un-accomplishment is the current validation method of the spacial localization technique, based on traceroutes; it does not give constant results for all tested cases; another validation method will have to be used; this is presented in the next section.

7.2 Future Work

A first clear objective can be set for future work: confident validation of the localization technique. So far the traceroute validation of the reconstruction result is confirmed in just a few of the tested cases. Since traceroute data is not accurate, both situations are possible: that there is an instability but it is not reflected by the traceroute, and that there is no instability but the traceroute presents abnormal behavior. It is also possible that the abnormal delay is caused by the nodes from the beginning or end of the paths, nodes that were assumed stable and consequently eliminated from the analysis. However, algorithms have to detect something as long as the routing matrix is not empty.

A secure way to confirm that localization works is to create a simulation model of the network: the same characteristics of the hops on the path, the same traffic characteristics in the network. On this artificial network controlled delays can be introduced or extra traffic in order to generate the instabilities. Afterwards, the same information as for Dragon-Lab is

acquired in order to apply the framework on it. In this way the problem nodes are known and the validation can be conducted.

A second direction for future work is the software implementation of the localization technique based on the algorithm presented in the previous chapter.

Bibliography

[1] Zhang B., Eugene Ng T. S., Nandi A., Riedi R., Druschel P., Wang G.: Measurement Based Analysis, Modeling, and Synthesis of the Internet Delay Space. *In proceeding of ACM Internet Measurement Conference (IMC)*, 2006.

[2] Duffield N. G., Horowitz J., Lo Presti F. and Towsley D.: Network Delay Tomography from End-to-End Unicast Measurements. *In Springer LNCS 2170 (Proc. IWDC)*, 2001.

[3] Gopalan K., Chiueh T.-c., Lin Y.-J.: Probabilistic Delay Guarantees Using Delay Distribution Measurement. In *proceeding of ACM Multimedia*, 2004.

[4] A. Markopoulou, G. Iannacone, S. Bhattacharyya, CN. Chuah, Y. Ganjali and C. Diot: Characterization of Failures in an Operational IP Backbone Network. *In IEEE/ACM Transactions on networking, Vol.16, No.4* , August 2008.

[5] D. Turner, K. Levchencko, A. Snoeren and S. Savage: California Fault Lines: Understanding the Causes and Impact of Network Failures. *In ACM SIGCOMM*, 2010.

[6] Y. Huang, N. Feamester, A. Lakhina, J. Xu: Diagnosing Network Disruptions with Network Wide Analysis. *In the proceeding of SIGMETRICS*, 2007.

[7] G. Iannaconne, C. Chauh, R. Mortier, S. Bhattacharyya and C. Diot: Analysis of Link Failures in an IP Backbone. In *proceeding of IMW 2002. ACM Press. Marseille (France)*. November 2002.

[8] He Yan, Lee Breslau, Zihui Ge, Dan Massey, Dan Pei, and Jennifer Yates. G-RCA: A Generic Root Cause Analysis Platform for Service Quality Management in Large IP Networks. In Proceedings of ACM CoNEXT, 2010.

[9] A. Abdelkefi, Yuming Jiang: Dragon-Lab, End-to-end Measurement-based Internet Paths States Diagnosis. Under preparation.

[10] A. Abdelkefi, Y. Jiang: A Structural Analysis of Network Delay. *In proceeding of IEEE Conference on Communications Networks and Services Research CNSR*, 2011.

[11] Q. Ke, et al: Robust L1 Norm Factorization in the Presence of Outliers and Missing Data by Alternative Convex Programming. In *IEEE Conf. CVPR*, 2005. Conf. on Computer Vision and Pattern Recognition.

[12] D. Donoho: Compressed Sensing. *In IEEE Trans. on Information Theory*, 2006.

[13] Z. Lin, et al: The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low Rank Matrices. *preprint*, 2009.

[14] B. Choi et al., Analysis of Point-To-Point Packet Delay in an Operational Network. *IEEE INFOCOM'04, Hong Kong*, March 2004.

[15] C. Lee et al., Operating a Network Link at 100%". Passive and Active Measurement 2011, *Lecture Notes in Computer Science*, Volume 6579.

[16] E. Myakotnykh et al, An Analysis of Interdomain Availability and Causes of Failures Based on Active Measurements. unpublished. 2011.

[17] Compressive sensing on www.wikipedia.com

[18] Baraniuk, R., Compressive sensing. *IEEE Signal Processing Mag.*, 24(4), 118–120, 124, 2007.

[19] Candès, E., Compressive sampling. *In Proc. Int. Congress of Math*. Madrid, Spain, August 2006.

[20] Donoho, D., Compressed sensing. *IEEE Trans. Inform. Theory*, 52(4), 1289–1306, 2006.

[21] Candès, E. and Tao, T. (2005). Decoding by linear programming. *IEEE Trans. Inform. Theory*, 51(12), 4203–4215

[22] Donoho, D. and Elad, M., Optimally sparse representation in general (non-orthogonal) dictionaries via minimization, *Proc. Natl. Acad. Sci*., 100(5), 2197–2202, 2003.

[23] R. Berinde, A. Gilbert, P. Indyk, H. Karloff, and M. Strauss, Combining geometry and combinatorics: a unified approach to sparse signal recovery. Preprint, 2008

[24] R. Berinde and P. Indyk, Sparse recovery using sparse random matrices. *MIT-CSAIL Technical Report*, 2008

[25] Candès, E., Wikipedia course available at: http://www.wikicoursenote.com/wiki/Compressive_Sensing_%28Candes%29

[26] Baraniuk, R., Wikipedia course available at: http://www.wikicoursenote.com/wiki/Compressive_Sensing

[27] Baraniuk R., Davenport M., Duarte M., Hegde C., An Introduction to Compressive Sensing. Available at http://cnx.org/content/col11133/1.5

[28] Boufounos, P. and Duarte, M. and Baraniuk, R. (2007, Aug.), Sparse signal reconstruction from noisy compressive measurements using cross validation. In *Statistical Signal Processing, SSP '07. IEEE/SP 14th Workshop on*, 2007.

[29] E. Candes, J. Romberg and T. Tao,: Stable Signal Recovery from Incomplete and Inaccurate Measurements, *Communications on Pure and Applied Mathematics, 59(8), pp. 1207-1223*, 2006.

[30] Matlab release 2011a, documentation available at: http://www.mathworks.com/help/toolbox/optim/ug/linprog.html

[31] L1-MAGIC documentation available at: http://www.acm.caltech.edu/l1magic/downloads/l1magic.pdf

[32] Figueiredo M., Nowak R., Wright S., Gradient Projection for Sparse Reconstruction. Documentation available at: http://www.lx.it.pt/~mtf/GPSR/

[33] SparseLab package for systems of linear equations available at: http://sparselab.stanford.edu/

[34] Tibshirani, R., Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc B.*, 1996. Available at: www-stat.stanford.edu/~tibs/lasso.html

[35] Donoho, D.L., Y. Tsaig, I. Drori and J.L. Starck, Sparse solution of underdetermined linear equations by stagewise orthogonal matching pursuit. *Stanford Technical Report*, 2006.

[36] Yall1 package for L1-minimization problems available at: http://yall1.blogs.rice.edu/

[37] Koh K., Kim S., Boyd S., Simple Matlab Solver for l1-regularized Least Squares Problems, 2008, available at: http://www.stanford.edu/~boyd/l1_ls/

[38] Beck M., Teboulle M., A Fast Iterative Shrinkage- Thresholding Algorithm for Linear Inverse Problems. *SIAM J. IMAGING SCIENCES, Vol. 2, No. 1, pp. 183–202, 2009.*

[39] L1 Homotopy Matlab toolbox available at: http://users.ece.gatech.edu/~sasif/homotopy/

[40] Friedlander M., Saunders M., Basis Pursuit Denoising and the Dantzig Selector. *West Coast Optimization Meeting*, 2007. Available at: http://www.stanford.edu/group/SOL/talks/friedlander-saunders-WCOM07.pdf

[41] Bayesian Compressive Sensing solver available at: http://people.ee.duke.edu/~lcarin/BCS.html

[42] S.Ji, Y.Xue and L.Carin, Bayesian Compressive Sensing. *IEEE Trans. Signal Processing*, vol. 56, no. 6, pp. 2346-2356, June 2008.

[43] F. Viger, B. Augustin, X. Cuvellier, C. Magnien, M. Latapy, T. Friedman, and R. Teixeira, Detection, understanding, and prevention of traceroute measurement artifacts. *Computer Networks*, vol. 52, no. 5, no. 5, pp. 998–1018, 2008.

[44] Pucha, H., Zhang, Y., Mao, Z.M., and Hu, Y.C, Understanding network delay changes caused by routing events. In *Proceedings of SIGMETRICS*. 2007

[45] Mehrotra, S., On the Implementation of a Primal-Dual Interior Point Method, SIAM Journal on Optimization, Vol. 2, pp. 575–601, 1992.

[46] Dai W. , Milenkovic O., Structured sublinear compressive sensing via dense belief propagation. Department of Electrical and Computer Engineering, University of Illinois, 2011.

[47] Kim S., Koh K., Boyd S. An Interior-Point Method for Large-Scale L1-regularized Least Squares, *IEEE Journal of Selected Topics in Signal Processing*, Vol. 1, No. 4., 2007.

[48] X. Jiang, Y. Yao, H. Liu, L. Guibas, Compressive Network Analysis, *Stanford University*, 2011.

[49] Compressed sensing lower bound, http://nuit-blanche.blogspot.com/2008/01/compressed-sensing-lower-bound-for.html

[50] Montemanni R., Heuristic Construction of Constant Weight Binary Codes. *Technical Report No. IDSIA*, Dec 2007.

[51] Vanderbei J. R., Linear Programming. *Springer-Verlag New York Inc*., 2010.

Appendix

A.1 Additional Results

Table A.1 Test vectors form

Ref. Nr.	Data Set 1					Data Set 2					Data Set 3			
	K=1	K=2	K=3	K=4		K=1	K=2	K=3	K=4		K=1	K=2	K=3	K=4
1	3,74	2,57	3,77	100,1		2,37	0,32	4,36	1,82		0,53	2,86	2,88	3,57
2	102,1	1,20	0,66	2,80		4,76	1,79	2,54	2,04		3,73	3,74	0,05	2,01
3	4,06	1,30	1,78	3,06		1,24	1,17	3,94	1,84		3,65	1,60	4,05	4,29
4	1,90	3,79	101,9	1,50		1,93	1,02	2,37	2,34		3,59	2,46	3,04	4,60
5	1,60	4,97	4,43	3,99		2,16	4,07	4,14	2,52		0,67	1,11	2,40	100,9
6	4,93	1,78	0,11	3,98		4,15	1,97	1,61	4,55		2,23	4,70	1,34	100,3
7	3,59	103,7	4,22	3,91		4,12	0,27	4,88	1,03		2,54	2,41	1,29	3,98
8	2,07	0,55	1,44	1,76		2,26	101,8	101,3	1,69		2,65	102,7	102,4	0,71
9	0,49	2,99	1,25	0,27		1,90	103,8	100,3	2,87		4,30	1,11	1,14	100,5
10	3,67	2,15	2,44	3,54		4,63	0,83	103,7	2,43		3,39	0,48	100,2	3,05
11	3,19	3,65	3,65	4,96		3,70	4,56	4,16	1,31		4,03	100,3	0,85	3,52
12	0,37	1,31	1,01	0,81		103,6	1,60	4,61	2,90		2,66	4,10	1,29	1,92
13	0,60	0,47	1,08	100,3		4,73	1,65	1,64	4,39		4,78	3,86	100,9	100,5
14	4,91	2,25	104,8	4,56		2,55	1,02	4,02	0,30		0,33	0,98	3,03	4,44
15	2,48	3,20	2,97	100,9		3,96	3,84	2,69	2,20		2,71	4,48	4,12	0,28
16	0,11	0,66	1,52	4,26		2,26	0,35	2,32	100,9		1,41	3,42	4,05	0,69
17	0,27	2,26	4,84	100,5		4,25	4,75	4,10	100,3		2,40	3,28	4,01	4,32
18	0,70	3,26	4,48	0,93		1,95	0,79	4,76	2,70		3,42	4,95	3,54	2,11
19	4,47	4,13	0,95	1,24		3,69	1,43	0,38	3,84		1,04	0,17	4,30	2,06
20	2,33	101,5	0,01	0,27		4,88	3,44	3,54	1,17		3,04	2,12	3,91	4,80
21	2,80	2,01	3,56	3,04		2,62	0,71	1,17	2,94		1,63	2,45	1,02	3,75
22	2,47	4,42	104,3	3,89		2,15	2,56	1,99	2,29		4,40	2,92	4,97	4,90
23	0,34	3,50	0,59	2,56		1,04	3,61	1,34	100,3		0,67	0,42	0,47	1,17
24	4,49	1,21	0,20	0,14		1,62	4,64	4,16	100,2		100,5	3,30	3,25	0,48

Table A.2 Used test random sensing matrix

0,46	0,61	0,41	0,72	0,16	0,38	0,27	0,88	0,92	0,14	0,33	0,58	0,55	0,08	0,55	0,02	0,23	0,13	0,61	0,34	0,50	0,80	0,88	0,24
0,05	0,68	0,22	0,44	0,53	0,15	0,78	0,43	0,30	0,09	0,22	0,72	0,28	0,08	0,79	0,03	0,12	0,64	0,77	0,90	0,36	0,63	0,59	0,76
0,84	0,99	0,63	0,85	0,46	0,32	0,61	0,63	0,54	0,31	0,52	0,34	0,70	0,07	0,80	0,71	0,99	0,75	0,94	0,24	0,77	0,98	0,30	0,30
0,16	0,99	0,01	0,39	0,38	0,28	0,02	0,59	0,33	0,81	0,35	0,01	0,88	0,90	0,86	0,54	0,07	0,81	0,13	0,22	0,78	0,16	0,17	0,42
0,12	0,76	0,04	0,84	0,09	0,82	0,70	0,23	0,24	0,51	0,86	0,19	0,02	0,97	0,80	0,52	0,06	0,27	0,26	0,99	0,03	0,62	0,34	0,06
0,27	0,28	0,18	0,75	0,26	0,34	0,02	0,78	0,55	0,87	0,26	0,51	0,34	0,39	0,01	0,50	0,66	0,43	0,38	0,95	0,05	0,31	0,07	0,03
0,31	0,95	0,20	0,58	0,34	0,87	0,68	0,29	0,11	0,98	0,84	0,02	0,42	0,31	0,92	0,07	0,34	0,40	0,99	0,64	0,32	0,09	0,65	0,19

Table A.3 Used test binary sensing matrix

1	1	1	0	1	1	1	1	1	1	1	0	1	0	0	1	0	0	0	0	1	0	0	0
1	1	0	1	0	1	1	0	0	1	0	1	0	1	0	0	1	0	0	0	1	0	0	0
0	1	0	1	1	0	0	1	1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0
1	0	1	1	0	1	0	1	0	1	0	0	1	1	1	1	1	0	1	1	1	0	1	0
1	1	1	1	0	0	0	0	1	0	1	1	0	0	0	1	1	1	0	1	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	1	1	1	1	0	1	1	1	1	0	1	0	1
0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1

Table A.4 Complete table of algorithms recovery performance case 1.1 NTE – Auckland

Node	Linprog	L1eq	GPSR	BP	Lasso	L1ls	BCS
1	132,838	174,389	232,805	173,638	4,838	202,644	214,257
2	125,073	172,098	239,359	163,977	0,000	153,294	201,021
3	18,042	21,214	2,024	25,991	338,158	36,859	4,215
4	44,318	33,905	24,401	32,026	0,000	16,093	6,468
5	0,000	0,000	-0,018	0,000	0,000	-0,003	0,166
6	51,250	22,866	0,000	24,571	0,000	26,404	8,393
7	41,694	25,062	34,975	27,314	0,000	32,641	2,402
8	133,413	130,659	104,741	131,584	344,148	116,684	132,743
9	0,000	-45,351	-69,969	-39,979	0,000	-37,271	-17,431
10	0,000	0,000	0,000	0,000	0,000	-0,001	0,000
11	0,000	0,000	0,000	0,000	0,000	-0,004	1,490
12	34,412	18,525	0,683	24,015	0,000	47,771	1,430
13	41,694	25,062	34,975	27,314	0,000	32,641	2,402
14	45,670	41,121	45,048	36,806	0,000	18,483	18,610
15	59,261	121,579	151,235	112,540	9,328	108,960	118,043
16	0,000	0,000	0,000	0,000	0,000	0,000	0,123
17	45,670	41,121	45,048	36,806	0,000	18,483	18,610
18	18,042	21,214	2,024	25,991	0,000	36,859	4,215

19	122,825	103,313	106,987	92,805	0,000	40,232	99,100
20	367,545	303,232	234,746	316,978	950,187	343,340	262,439
21	0,000	0,000	0,000	0,000	0,000	-0,001	0,000
22	0,000	-45,351	-69,969	-39,979	0,000	-37,271	-17,431
23	367,545	303,232	234,746	316,978	0,000	343,340	262,439
24	0,000	0,000	0,000	0,000	12,834	-0,003	-0,003

Table A.5 Test data vectors NTE – Auckland

case 1	case 2.1	case 2.2	case 2.3
54,39	48,15	49,98	45,71
53,96	53,05	49,37	54,58
51,06	46,78	46,68	52,00
62,92	57,41	57,65	55,12
53,36	50,95	51,84	45,05
50,60	53,68	53,17	53,53
45,01	53,19	52,06	49,67
47,51	550,00	550,00	550,00
49,32	45,02	550,00	47,21
49,31	54,45	49,61	53,06
53,75	52,92	49,93	550,00
45,68	48,99	47,74	54,98
48,40	53,73	51,55	54,38
46,77	51,74	49,07	50,54
47,46	47,42	48,00	54,13
52,28	46,68	51,05	51,35
47,88	53,61	52,91	52,80
47,37	48,27	54,71	50,67
46,40	48,25	52,73	46,92
1.000,00	49,00	51,00	54,06
47,21	550,00	49,49	50,15
48,78	46,64	46,53	54,99
51,66	51,88	47,11	48,77
54,36	51,43	52,00	47,06

Table A.6 NTE – Auckland routing matrix, day 11

Row \ Col	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	1	0	0	1	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0
2	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0
3	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0
4	0	1	1	0	0	1	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	0	0
5	0	1	0	0	0	0	0	1	1	0	1	1	0	0	1	0	0	0	0	0	0	1	0	0
6	1	0	0	1	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
7	0	0	0	1	0	0	1	1	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0

Table A.7 NTE – Cernet routing matrix, day 35

Row \ Col	1	2	3	4	5	6
1	0	0	1	1	1	0
2	0	1	0	1	0	1
3	0	0	0	1	1	1
4	0	1	1	1	0	0
5	1	1	1	0	0	0
6	1	0	1	0	1	0
7	1	1	0	0	0	1

Table A.8 NTE – Cernet original routing matrix, day 35

Row \ Col	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	0
2	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	0	1
3	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	0
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
6	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	0
7	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	0	0	1

Table A.9 Data vector reconstruction NTE – Auckland, day 11

Node	L1eq	GPSR	BP	L1ls
1	561,07	471,07	564,78	572,90
2	1,82	0,40	1,04	1,32
3	403,58	451,95	396,94	391,72

4	3,67	0,00	2,78	3,30
5	0,00	0,00	0,00	0,01
6	559,97	547,93	569,34	571,99
7	0,00	0,00	0,00	0,01
8	806,74	912,28	794,41	783,25
9	0,20	0,00	0,15	0,28
10	0,31	0,00	0,38	0,27
11	0,95	0,09	0,88	0,76
12	562,94	475,17	566,84	574,83
13	0,00	0,00	0,00	0,01
14	0,76	0,00	1,16	0,91
15	560,51	542,55	569,90	572,64
16	0,94	0,08	1,15	0,80
17	0,76	0,00	1,16	0,91
18	403,58	451,95	396,94	391,72
19	0,00	0,28	0,00	0,01
20	0,00	0,00	0,00	0,01
21	0,31	0,00	0,38	0,27
22	0,20	0,00	0,15	0,28
23	0,00	0,00	0,00	0,01
24	0,27	0,00	0,20	0,41

Table A.10 Data vector reconstruction NTE – Auckland, day 11, K=1 and K=2

Sparsity	K=1				K=2			
Node	L1eq	GPSR	BP	L1ls	L1eq	GPSR	BP	L1ls
1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
5	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
6	0,00	0,00	0,00	0,00	0,00	547,93	0,00	0,00
7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
8	806,74	912,28	794,41	783,25	806,74	912,28	794,41	783,25
9	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
10	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
12	0,00	0,00	0,00	0,00	562,94	0,00	0,00	574,83
13	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
14	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
15	0,00	0,00	0,00	0,00	0,00	0,00	569,90	0,00
16	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

18	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
19	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
20	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
21	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
23	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
24	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Table A.11 Data vector reconstruction NTE – Cernet, day 35

Node	L1eq	GPSR	BP	L1ls
1	1.024,00	1.353,88	1.281,41	1.356,73
2	512,00	901,78	886,63	901,47
3	1.536,00	1.553,60	1.645,83	1.554,59
4	384,00	985,14	908,18	984,75
5	1.024,00	1.437,25	1.423,09	1.440,18
6	256,00	785,42	881,53	789,65

Table A.12 Data vector reconstruction NTE – Cernet, day 35, K=1 and K=2

Sparsity	K=1				K=2			
Node	L1eq	GPSR	BP	L1ls	L1eq	GPSR	BP	L1ls
1	0,00	0,00	0,00	0,00	1.024,0	0,00	0,00	0,00
2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3	1.536,0	1.553,6	1.645,8	1.554,5	1.536,0	1.553,6	1.645,8	1.554,5
4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
5	0,00	0,00	0,00	0,00	1.024,0	1.437,2	1.423,0	1.440,1
6	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Table A.13 General Dragon-Lab performance absolute values and rates

Link	λ	FP	D	ND	FP rate [%]	ND rate [%]	D rate [%]
Cernet NTE	2	165	562	7	29,36	1,25	98,77
	2,5	60	418	23	14,35	5,50	94,78
	3	31	336	45	9,23	13,39	88,19
NTE Cernet	2	209	1678	127	12,46	7,57	92,96
	2,5	60	875	207	6,86	23,66	80,87
	3	27	518	289	5,21	55,79	64,19
AUCKLA NTE	2	682	1952	6	34,94	0,31	99,69
	2,5	270	1018	17	26,52	1,67	98,36
	3	78	616	44	12,66	7,14	93,33
NTE AUCKLA	2	953	2752	40	34,63	1,45	98,57
	2,5	154	1341	123	11,48	9,17	91,60
	3	36	780	272	4,62	34,87	74,14

where,

FP False positive
D Detected
ND Not detected
C Congestion
Q Queuing building up
F Link failure

D rate = (D / (D +ND)) * 100%

FP rate = (FP / D) * 100%

ND rate = (ND / D) * 100%

Table A.14 a) Dragon-Lab classification performance absolute values and rates

Link	λ	FP - C	FP - Q	FP - F	D - C	D - Q	D - F	ND - C	ND - Q	ND - F
Cernet NTE	2	19	128	18	113	263	186	1	2	4
	2,5	6	49	5	117	147	154	11	4	8
	3	2	28	1	87	98	151	28	3	14
NTE Cernet	2	115	14	80	840	173	665	117	3	7
	2,5	28	6	26	415	131	329	192	3	12
	3	16	1	10	255	77	186	264	4	21
AUCKLA NTE	2	371	57	254	848	123	981	5	0	1
	2,5	143	28	99	362	65	591	13	1	3
	3	38	14	26	151	41	424	31	1	12
NTE AUCKLA	2	484	39	430	1230	93	1429	38	1	1
	2,5	86	13	55	533	40	768	113	7	3
	3	24	0	12	274	15	491	256	4	12

Table A.14 b) Dragon-Lab classification performance absolute values and rates

Link	λ	FP rate – C[%]	FP rate - Q[%]	FP rate - F[%]	D rate - C[%]	D rate - Q[%]	D rate - F[%]
Cernet NTE	2	16,81	48,67	9,68	99,12	99,25	97,89
	2,5	5,13	33,33	3,25	91,41	97,35	95,06
	3	2,30	28,57	0,66	75,65	97,03	91,52
NTE Cernet	2	13,69	8,09	12,03	87,77	98,30	98,96
	2,5	6,75	4,58	7,90	68,37	97,76	96,48
	3	6,27	1,30	5,38	49,13	95,06	89,86
AUCKLA NTE	2	43,75	46,34	25,89	99,41	100,00	99,90
	2,5	39,50	43,08	16,75	96,53	98,48	99,49
	3	25,17	34,15	6,13	82,97	97,62	97,25
NTE AUCKLA	2	39,35	41,94	30,09	97,00	98,94	99,93
	2,5	16,14	32,50	7,16	82,51	85,11	99,61
	3	8,76	0,00	2,44	51,70	78,95	97,61

A.2 Matlab Routines

ROUTINE 1. GENERAL INSTABILITIES IDENTIFICATION AND CLASSIFICATION

```
% Internet paths IDs
% path 5002 = NTE - Cernet
% path 1022 = Cernet - NTE
% path 5003 = NTE - AUCKLA
% path 4003 = AUCKLA - NTE

% Assignment of spreadsheet from the visual inspection of Dragon-Lab
% signaled instabilities according to the path
path=5002;
var_lambda=2.5;
if (path==5002)
    nrsheet=2;
end
if (path==1022)
    nrsheet=5;
end
if (path==4003)
    nrsheet=8;
end
if (path==5003)
    nrsheet=11;
end
% Run PCP scripts
demoUninettv2(var_lambda,path)
demoUninett2v2(path)
demoUninett3v2(var_lambda,path)
% Import measurements data
ag_delay = xlsread('agregdelay.xlsx');
av_delay = xlsread('averagedelay.xlsx');
losses = xlsread('losses.xlsx');
% Read output of PCP scripts
mat1 = xlsread(strcat(num2str(path),'-agregated-delay-120.xlsx'));
mat2 = xlsread(strcat(num2str(path),'-avg-delay-120.xlsx'));
agreg_delay=mat1';
average_delay=mat2';

% Section for filtering physically impossible combinations of the three
% delay space features
[nr_time_bins, nr_days]=size(agreg_delay);
congestions=zeros(nr_time_bins,nr_days);
for j=1:1:nr_days
for i=1:1:nr_time_bins
    if (ag_delay(i,j)~=0)
        if  ((ag_delay(i,j) <0) && (av_delay(i,j) <0) && (losses(i,j)==0))
            ag_delay(i,j)=0;
        end
        if  ((ag_delay(i,j)~=0) && (av_delay(i,j)==0) && (losses(i,j)==0))
            ag_delay(i,j)=0;
        end
        if  ((ag_delay(i,j) <0) && (av_delay(i,j) >0) && (losses(i,j)==0))
            ag_delay(i,j)=0;
        end
        if  ((ag_delay(i,j) >0) && (av_delay(i,j) <0) && (losses(i,j)==0))
```

```
            ag_delay(i,j)=0;
        end
        if  ((losses(i,j)>0) && (ag_delay(i,j)>0) && (av_delay(i,j)<0))
            ag_delay(i,j)=0;
        end
    end
end
end

% Section for processing the remaining elements from the detected
% instabilities matrix - ag_delay; detections due to NTP synchronization
% delay variations are removed
for j=1:1:nr_days
aux=0;
for i=1:1:nr_time_bins
        % Eliminate detections due to NTP synchronization
        if ((aux>=4 && (ag_delay(i,j)==0 || i==nr_time_bins)) && ...
                                            losses((i-aux),j)==0)
        for k=(i-aux):1:(i)
            ag_delay(k,j)=0;
        end
        end

        % Save and eliminate series of detection longer than 4 time bins
        % which fit the congestion definition
        if ((aux>=4 && (ag_delay(i,j)==0 || i==nr_time_bins))  && ...
                        losses((i-aux),j)~=0 && losses(i,j)~=0 && ...
            (ag_delay((i-aux),j)>0) && (av_delay((i-aux),j)>0) && ...
                    ((agreg_delay((i-1),j) > agreg_delay(i,j)) && ...
                    (average_delay((i-1),j) > average_delay(i,j))))
            for m=i-aux:1:i
                congestions(m,j)=ag_delay(m,j);
            end
            for k=(i-aux):1:(i)
                ag_delay(k,j)=0;
            end

        % Eliminate any other series on detections longer than 4
        % consecutive time bins
        elseif ((aux>=4 && (ag_delay(i,j)==0 || i==nr_time_bins)) && ...
                losses((i-aux),j)~=0)
        for k=(i-aux):1:(i)
            ag_delay(k,j)=0;
        end
        end
        if (ag_delay(i,j)~=0)
        aux=aux+1;
        else aux=0;
        end
end
end
% Create the detections matrix with only possible true instabilities
ag_delay=ag_delay+congestions;

% Section for deleting false positive detections based on visual
% inspection saved in a spreadsheet corresponding to the path
fpmatrix=excel_manipulator(nrsheet);
% First step: filter visually inspected data of mistakenly input
% impossible entries
for j=1:1:nr_days
    aux2=0;
```

```
    for i=1:1:nr_time_bins
        if (fpmatrix(i,j)~=0)
        if (((ag_delay(fpmatrix(i,j),j)<0) && (av_delay(i,j)<0) && ...
        (losses(i,j)==0)) || ((fpmatrix(i,j)~=0) && (av_delay(i,j)==0)...
        && (losses(i,j)==0) ) || (ag_delay(fpmatrix(i,j),j)<0) && ...
        (av_delay(i,j)>0) && (losses(i,j)==0) ) || ...
        ((ag_delay(fpmatrix(i,j),j)>0) && (av_delay(i,j)<0) && ...
        (losses(i,j)==0)) || ((ag_delay(fpmatrix(i,j),j)>0) && ...
        (av_delay(i,j)<0) && (losses(i,j)>0) ) || ((fpmatrix(i,j)~=0 )...
        && ag_delay(i,j)==0))
        fpmatrix(i,j)=0;
        end
        end
        if ((aux2>=4 && (fpmatrix(i,j)==0 || i==nr_time_bins)) && ...
            losses((i-aux2),j)==0)
        for k=(i-aux2):1:(i)
            fpmatrix(k,j)=0;
        end
        end
        if (fpmatrix(i,j)~=0)
        aux2=aux2+1;
        else aux2=0;
        end
    end
end
% Second step: remove false positives from detected instabilities
for j=1:1:nr_days
    for i=1:1:nr_time_bins
        if (   (fpmatrix(i,j)~=0 )   )
            ag_delay(fpmatrix(i,j),j)=0;
        end
    end
end

% For better control and generality for the other codes matrices to save
% instabilities of different length are created; TB denotes time bins
anomalies_oneTB=zeros(nr_time_bins,nr_days);
anomalies_twoTB=zeros(nr_time_bins,nr_days);
anomalies_threeTB=zeros(nr_time_bins,nr_days);
anomalies_long=zeros(nr_time_bins,nr_days);

for j=1:1:nr_days
aux=0;
for i=1:1:nr_time_bins
        if ( (aux==1) && (ag_delay(i,j)==0 || i==nr_time_bins))
            anomalies_oneTB(i-1,j)=ag_delay(i-1,j);
        end
        if ( (aux==2) && (ag_delay(i,j)==0 || i==nr_time_bins))
            for m=i-aux:1:i
                anomalies_twoTB(m,j)=ag_delay(m,j);
            end
        end
        if ( (aux==3) && (ag_delay(i,j)==0 || i==nr_time_bins))
            for n=i-aux:1:i
                anomalies_threeTB(n,j)=ag_delay(n,j);
            end
        end
        if ((aux>=4 && (ag_delay(i,j)==0 || i==nr_time_bins)) && ...
           losses((i-aux),j)~=0  && losses(i,j)~=0 && ...
           (ag_delay((i-aux),j)>0) && (av_delay((i-aux),j)>0) && ...
           ((agreg_delay((i-1),j) > agreg_delay(i,j)) && ...
```

```
            (average_delay((i-1),j) > average_delay(i,j))))
             for m=i-aux:1:i
                 anomalies_long(m,j)=ag_delay(m,j);
             end
        end
        if (ag_delay(i,j)~=0)
        aux=aux+1;
        else aux=0;
        end
end
end

% Section for classifying network states that last for one time bin
% they are saved into the cong, qbup and failure matrices
cong_oneTB=zeros(nr_time_bins,nr_days);
qbup_oneTB=zeros(nr_time_bins,nr_days);
failure_oneTB=zeros(nr_time_bins,nr_days);
for j=1:1:nr_days
    for i=1:1:nr_time_bins
        if (((anomalies_oneTB(i,j)>0) && (av_delay(i,j)>0) && ...
           (losses(i,j)>0) ) || ((anomalies_oneTB(i,j)>0) && ...
           (av_delay(i,j)==0) && (losses(i,j)>0)))
        cong_oneTB(i,j)= anomalies_oneTB(i,j);
        elseif ((anomalies_oneTB(i,j)>0) && (av_delay(i,j)>0) && ...
                (losses(i,j)==0))
        qbup_oneTB(i,j)= anomalies_oneTB(i,j);
        elseif (((anomalies_oneTB(i,j)<0) && (av_delay(i,j)==0) && ...
               (losses(i,j)>0) ) || ((anomalies_oneTB(i,j)<0) && ...
               (av_delay(i,j)>0) && (losses(i,j)>0) ) || ...
               ((anomalies_oneTB(i,j)<0) && (av_delay(i,j)<0) && ...
               (losses(i,j)>0)))
        failure oneTB(i,j)= anomalies oneTB(i,j);
        end
    end
end

% Section for classifying network states that last for two time bins
cong_twoTB=zeros(nr_time_bins,nr_days);
qbup_twoTB=zeros(nr_time_bins,nr_days);
failure_twoTB=zeros(nr_time_bins,nr_days);
for j=1:1:nr_days
    for i=1:1:nr_time_bins
        if (((anomalies_twoTB(i,j)>0) && (av_delay(i,j)>0) && ...
           (losses(i,j)>0)) || ((anomalies_twoTB(i,j)>0) && ...
           (av_delay(i,j)==0) && (losses(i,j)>0)))
        cong_twoTB(i,j)= anomalies_twoTB(i,j);
        elseif ((anomalies_twoTB(i,j)>0) && (av_delay(i,j)>0) && ...
               (losses(i,j)==0))
        qbup_twoTB(i,j)= anomalies_twoTB(i,j);
        elseif (((anomalies_twoTB(i,j)<0) && (av_delay(i,j)==0) && ...
               (losses(i,j)>0)) || ((anomalies_twoTB(i,j)<0) && ...
               (av_delay(i,j)>0) && (losses(i,j)>0)) || ...
               ((anomalies_twoTB(i,j)<0) && (av_delay(i,j)<0) && ...
               (losses(i,j)>0)))
        failure_twoTB(i,j)= anomalies_twoTB(i,j);
        end
    end
end

% Section for classifying network states that last for three time bins
cong_threeTB=zeros(nr_time_bins,nr_days);
```

```
qbup_threeTB=zeros(nr_time_bins,nr_days);
failure_threeTB=zeros(nr_time_bins,nr_days);
for j=1:1:nr_days
    for i=1:1:nr_time_bins
        if (((anomalies_threeTB(i,j)>0) && (av_delay(i,j)>0) && ...
           (losses(i,j)>0) ) || ((anomalies_threeTB(i,j)>0) && ...
           (av_delay(i,j)==0) && (losses(i,j)>0)))
        cong_threeTB(i,j)= anomalies_threeTB(i,j);
        elseif ((anomalies_threeTB(i,j)>0) && (av_delay(i,j)>0) && ...
               (losses(i,j)==0))
        qbup_threeTB(i,j)= anomalies_threeTB(i,j);
        elseif (((anomalies_threeTB(i,j)<0) && (av_delay(i,j)==0) && ...
               (losses(i,j)>0)) || ((anomalies_threeTB(i,j)<0) && ...
               (av_delay(i,j)>0) && (losses(i,j)>0)) || ...
               ((anomalies_threeTB(i,j)<0) && (av_delay(i,j)<0) && ...
               (losses(i,j)>0)))
        failure_threeTB(i,j)= anomalies_threeTB(i,j);
        end
    end
end

% The matrices for the three types of instabilities are the superposition
% of the previously found matrices
cong    = cong_oneTB + cong_twoTB + cong_threeTB + anomalies_long;
qbup    = qbup_oneTB + qbup_twoTB + qbup_threeTB;
failure = failure_oneTB + failure_twoTB + failure_threeTB;

savefile='cong.mat';
save(savefile,'cong');
savefile='qbup.mat';
save(savefile,'qbup');
savefile='failure.mat';
save(savefile,'failure');
```

ROUTINE 2. INSTABILITIES DURATIONS CDF GENERATION

```
% Simplified example code for one Internet path and for
% one type of instabilities, namely congestions.
% cong matrix is obtained with Routine 1.
% Introduce one row (one time bin) of zeros so that days
% will be clearly separated after reshaping
[m,n]=size(ag_delay);
zeros_vector = zeros(1,n);
cong=vertcat(cong,zeros_vector);
% Reshape cong matrix into whole measurement period vector
cong_vector=reshape(cong,1,((m+1)*n));

% Section to generate congestions durations vector
aux_vector_cong=cong_vector; cong_durations_vector=[];
cong_durations_vector_aux=[];  aux_cong=0;
% Compute lengths of congestions
for i=1:(length(aux_vector_cong))
            if (aux_vector_cong(i)~=0)
                aux_cong=aux_cong+1;
                if (i==length(aux_vector_cong))
                    cong_durations_vector_aux(i)=aux_cong;
                end
            else
                cong_durations_vector_aux(i)=aux_cong;
```

```
                aux_cong=0;
            end
end
% Eliminate zeros
cong_durations_vector = cong_durations_vector_aux ...
(find(cong_durations_vector_aux~=0));

% Transform time bin durations into minutes durations
cong_durations_vector=2*cong_durations_vector;
figure(1)
cdfplot(cong_durations_vector)
title('Congestions duration CDF')
```

ROUTINE 3. TIME BETWEEN INSTABILITIES CDF GENERATION

```
% Simplified example code for one Internet path and for
% one type of instabilities, namely congestions.
% cong matrix is obtained with Routine 1.

% Leave only the beginning of each detected congestion
[nr_time_bins, nr_days]=size(agreg_delay);
for j=1:1:nr_days
aux_cong=0;
for i=1:1:nr_time_bins
        if ((aux_cong==3 || aux_cong==2 || aux_cong>=4) && ...
            (cong(i,j)==0 || i==nr_time_bins))
            for k=(i-aux_cong+1):1:(i)
                cong(k,j)=0;
            end
        end
        if (cong(i,j)~=0)
        aux_cong=aux_cong+1;
        else aux_cong=0;
        end
end
end

% Process cong.mat for cdfplot
[row_cong,col_cong]=size(cong);
cong_vector=reshape(cong,1,(row_cong*col_cong));
lengths vector cong=[];
lengths_vector_aux_cong=[];
aux_cong=1;
for i=1:(length(cong_vector))
            if  (cong_vector(i)~=0 )
                lengths_vector_aux_cong(i)=aux_cong;
                aux_cong=1;
            else
                lengths_vector_aux_cong(i)=0;
                aux_cong=aux_cong+1;
            end
end

% Eliminates zeros from lengths_vector_aux
```

```
lengths_vector_cong = lengths_vector_aux_cong...
(find(lengths_vector_aux_cong~=0));
% Transform time bins to minutes
lengths_vector_failure_5002=2*lengths_vector_failure_5002

figure(1)
cdfplot(lengths_vector_cong)
title('Time between congestions CDF')
```

ROUTINE 4. GENERAL PATH METRICS COMPUTATION

```
% Compute Internet paths performance metrics: availability,
% fatigue, stability per hour
% cong, qbup and failure are given by Routine 1.

[m,n]=size(ag_delay);
aux_vector_cong=reshape(cong,1,(m*n));
aux_vector_qbup=reshape(qbup,1,(m*n));
aux_vector_failure=reshape(failure,1,(m*n));

% Commpute metrics per hour

% Replace actual values in the instabilities matrices with ones in
% order to be easier to count them
for i=1:(length(aux_vector_failure))
        if (aux_vector_failure(i)~=0)
            aux_vector_failure(i)=1;
        end
        if (aux_vector_cong(i)~=0)
            aux_vector_cong(i)=1;
        end
        if (aux_vector_qbup(i)~=0)
            aux_vector_qbup(i)=1;
        end
end
% Subsection for reshaping the vectors into matrix form which
% contains 30 time bins (one hour) per column, so that afterwards
% to sum the number of network unstable states per hour
vector_failure = reshape(aux_vector_failure,m,n);
vector_cong = reshape(aux_vector_cong,m,n);
vector_qbup = reshape(aux_vector_qbup,m,n);
filling = zeros(1,n);
% Every day one time bin is missing; therefore matrices are filled
% with 0 corresponding to the last time bin of the day
vector_failure = vertcat(vector_failure,filling);
vector_cong = vertcat(vector_cong,filling);
vector_qbup = vertcat(vector_qbup,filling);
vector_failure = reshape(vector_failure,1,((m+1)*n));
vector_cong = reshape(vector_cong,1,((m+1)*n));
vector_qbup = reshape(vector_qbup,1,((m+1)*n));

% 30 = number of time bins per hour
vector_failure_per_hour=reshape(vector_failure,30,(((m+1)*n)/30));
```

```
vector_qbup_per_hour=reshape(vector_qbup,30,(((m+1)*n)/30));
vector_cong_per_hour=reshape(vector_cong,30,(((m+1)*n)/30));

bins_in_cong_per_hour=sum(vector_cong_per_hour);
bins_in_qbup_per_hour=sum(vector_qbup_per_hour);
bins_in_failure_per_hour=sum(vector_failure_per_hour);

% 1968 = number of hours in 82 days of measurement
% 30 = number of time bins per hour
availability_vector = zeros(1,1968);
unavailability_vector = zeros(1,1968);
fatigue_vector = zeros(1,1968);
stability_vector = zeros(1,1968);

for i=1:(length(bins_in_cong_per_hour))

availability_vector(i) = ( ( 30 - bins_in_failure_per_hour(i))/30);
fatigue_vector(i) = ( ( bins_in_cong_per_hour(i) + ...
     bins_in_qbup_per_hour(i)) / (30-bins_in_failure_per_hour(i)));
stability_vector(i) = ( ( 30 - bins_in_cong_per_hour(i) - ...
     bins_in_qbup_per_hour(i) - bins_in_failure_per_hour(i) )/30 );

end

fatigue_vector=fatigue_vector*100;
availability_vector=availability_vector*100;
stability_vector=stability_vector*100;

figure(1)
cdfplot(availability_vector);
figure(2)
cdfplot(fatigue_vector);
figure(3)
cdfplot(stability_vector);
```

ROUTINE 5 GENERATE REPARTITION OF INSTABILITIES PER TYPE PER PATH

```
% Simplified routine: only for link failures; only for one path.
% Routine for dividing link failures for one path into
% frequent/dispersed and short/medium/long.

% failure matrix is known from routine 1.

complete_failure_vector=reshape(failure,1,nr_time_bins*nr_days);
aux_complete_failure_vector=complete_failure_vector; aux=0;
% Leave only the first time bins of the link failure for separation
% into frequent/dispersed
for i=1:length(aux_complete_failure_vector)
    if (aux>1)
        for k=(i-aux+1):1:(i)
        aux_complete_failure_vector(k)=0;
        end
    end
```

```
    if (aux_complete_failure_vector(i)~=0)
        aux=aux+1;
    else aux=0;
    end
end
% If the distance between failures is larger than 15 time
% bins -> dispersed
% If the distance between failures is smaller than 15 time
% bins -> frequent
failure_disp=zeros(1,length(aux_complete_failure_vector));
failure_freq=zeros(1,length(aux_complete_failure_vector)); aux=0;
% Separate into frequent/dispersed
for i=1:length(aux_complete_failure_vector)
    if ( (aux>15) && (aux_complete_failure_vector(i)~=0))
        failure_disp(i)=1;
    end
    if ( (aux <= 15) && (aux_complete_failure_vector(i)~=0))
        failure_freq(i)=1;
    end
    if ( aux_complete_failure_vector(i)==0)
        aux=aux+1;
    else aux=0;
    end
end

% Reconstruction of the vectors of dispersed and frequent anomalies
failure_disp_aux=zeros(1,length(complete_failure_vector));
for i=1:(length(complete_failure_vector))
        if (failure_disp(i)~=0)
            index_disp=i;
            while (index_disp<=length(failure_disp)    &&  ...
                complete failure vector(index disp)~=0  )
            failure_disp_aux(index_disp)=...
                complete_failure_vector(index_disp);
            index_disp=index_disp+1;
            end
        end
end
failure_disp=failure_disp_aux;
failure_freq_aux=zeros(1,length(complete_failure_vector));
for i=1:(length(complete_failure_vector))
        if (failure_freq(i)~=0)
            index_freq=i;
            while (index_freq<=length(failure_freq)    &&  ...
                complete_failure_vector(index_freq)~=0  )
            failure_freq_aux(index_freq)=...
                complete_failure_vector(index_freq);
            index_freq=index_freq+1;
            end
        end
end
failure_freq=failure_freq_aux;
% Split dispersed and afterwards frequent link failures into
% different durations
aux=0; nr_disp_long=0; nr_disp_medium=0; nr_disp_short=0;
for i=1:length(failure_disp)
    if ( (aux>2) && (failure_disp(i)==0))
        nr_disp_long=nr_disp_long+1;
    end
    if ( (aux == 2) && (failure_disp(i)==0))
        nr_disp_medium=nr_disp_medium+1;
```

```
    end
    if ( (aux == 1) && (failure_disp(i)==0))
        nr_disp_short=nr_disp_short+1;
    end
    if ( failure_disp(i)~=0)
        aux=aux+1;
    else aux=0;
    end
end

aux=0; nr_freq_long=0; nr_freq_medium=0; nr_freq_short=0;
for i=1:length(failure_freq)
    if ( (aux>2) && (failure_freq(i)==0))
        nr_freq_long=nr_freq_long+1;
    end
    if ( (aux == 2) && (failure_disp(i)==0))
        nr_freq_medium=nr_freq_medium+1;
    end
    if ( (aux == 1) && (failure_freq(i)==0))
        nr_freq_short=nr_freq_short+1;
    end
    if ( failure_freq(i)~=0)
        aux=aux+1;
    else aux=0;
    end
end

results_failure=[nr_disp_long nr_disp_medium nr_disp_short ...
                 nr_freq_long nr_freq_medium nr_freq_short];
results_failure=(results_failure/(sum(results_failure)))*100;

figure (1)
bar(results_failure,'grouped')
```

ROUTINE 6. CREATE ANOMALIES VECTOR FOR USE IN ROUTING MATRIX

```
% In order to create this vector only the beginning time bin of
% the instabilities is kept; moreover, only congestions and queues
% building up time bins are kept as explained section 5.3

% Matrices cong, qbup and failure are available from routine 1.
% The measurement vector corresponds to one day and one path; here
% day=6, path 4003.

day=6;
path=4003;
% Leave one the first time bin for all instabilities
for j=1:1:nr_days
aux_cong=0;
aux_qbup=0;
aux_failure=0;
for i=1:1:nr_time_bins
    if ((aux_cong==3 || aux_cong==2 || aux_cong>=4) && ...
```

```
        (cong(i,j)==0 || i==nr_time_bins))
        for k=(i-aux_cong+1):1:(i)
            cong(k,j)=0;
        end
    end
    if ((aux_failure==3 || aux_failure==2 || aux_failure>=4) && ...
        (failure(i,j)==0 || i==nr_time_bins))
        for l=(i-aux_failure+1):1:(i)
            failure(l,j)=0;
        end
    end
    if ((aux_qbup==3 || aux_qbup==2 || aux_qbup>=4) && ...
        (qbup(i,j)==0 || i==nr_time_bins))
        for m=(i-aux_qbup+1):1:(i)
            qbup(m,j)=0;
        end
    end
    if (cong(i,j)~=0)
    aux_cong=aux_cong+1;
    else aux_cong=0;
    end
    if (qbup(i,j)~=0)
    aux qbup=aux qbup+1;
    else aux_qbup=0;
    end
    if (failure(i,j)~=0)
    aux_failure=aux_failure+1;
    else aux_failure=0;
    end
end
end
% Consider only
anomalies=cong+qbup;
anomalies=anomalies(:,day);
anomalies=anomalies';

% If queues and congestions merge, still only the first time bins
% will be considered.
aux=0;
for i=1:1:nr_time_bins
      if ( (aux>1) && (anomalies(i)==0 || i==(nr_time_bins*nr_days)))
            for k=(i-aux+1):1:(i)
                anomalies(k)=0;
            end
      end
      if (anomalies(i)~=0)
      aux=aux+1;
      else aux=0;
      end
end
```

ROUTINE 7. CREATE ROUTING MATRIX

```
% Load traceroute file; it will be uploaded as a vector, A.
fid=fopen('traceroute.txt');
A=fread(fid);
A=A';
% Reshape the vector into a matrix; row index is increase when ASCII
% code for new line is encountered.
B=[]; ind1=1; ind2=1;
for i=1:1:length(A)
    if (A(i)~=13)
    B(ind1,ind2)=A(i);
    ind1=ind1+1;
    end
    if (A(i)==13)
    ind2=ind2+1;
    ind1=1;
    end
end
B=B';
% Further processing is done on ASCII characters
% "32" = space   "13" = carriage return    "10" = new line
[m,n]=size(B);
for i=1:1:m
    for j=1:1:n
        if (B(i,j)==32 || B(i,j)==13 || B(i,j)==10 )
            B(i,j)=0;
        end
    end
end
% Eliminate isolated elements
for i=1:1:m
    for j=1:1:n
        if (j~=1)
            if (B(i,j)~=42)
                if ((B(i,j)~=0 && B(i,j-1)==0 && B(i,j+1)==0) || ...
                    (B(i,j)~=0 && B(i,j-1)==0 && B(i,j+2)==0))
                    B(i,j)=0;
                end
            end
        end
    end
end
% C is a matrix that holds the IPs in ASCII code and the timestamps
% in the beginning of each traceroute
C=[];
for i=1:1:m
column=1;
    for j=1:1:n
        if (B(i,j)~=0)
            C(i,column)=B(i,j);
            column=column+1;
        end
    end
end
C_aux=C;
C_increasing_hours=C;
% C_incresing_hours will hold the timestamps with hours increasing until
% 25:55 instead of 23:55, since timing starts at time 02:00.
```

```
% Number of non-zeros elements per row matrix
nr_nnz = sum(C_increasing_hours ~= 0,2);
for i=1:1:size(C_increasing_hours,1)
    if (nr_nnz(i)==8)
        if (C_increasing_hours(i,1)==48 && C_increasing_hours(i,2)==48)
            C_increasing_hours(i,1)=50;
            C_increasing_hours(i,2)=53;
        elseif (C_increasing_hours(i,1)==48 && ...
                C_increasing_hours(i,2)==49)
            C_increasing_hours(i,1)=50;
            C_increasing_hours(i,2)=54;
        end
    end
end
vect_indexes=[];        %holds the indexes of time stamps
vect_time_stamps=[];    %holds the timestamps in ASCII
ind=1;
for i=1:1:size(C_increasing_hours,1)
    if (nr_nnz(i)==8)  %lines with time stamps will have 8 elements
        vect_time_stamps(ind,:) = C_increasing_hours(i,:);
        vect_indexes(ind,1)=i;
        ind=ind+1;
    end
end
% Eliminates zeros columns
vect_time_stamps(:,find(sum(abs(vect_time_stamps)) == 0))=[];


% The detected instabilities may not have a corresponding traceroute; in
% this case the previous traceroute will be taken; this section maps the
% existing traceroutes to the detected instabilities time bins computed
% with code 2.
load anomalies_vector;
vect_indexes_anomalies=[];
ind=1;
for i=1:1:length(anomalies_vector)
    if ( anomalies_vector(i)~=0 )
        hours=floor((2*i)/60); % transform time bin into physical hour
        minutes=mod((2*i),60); % transform time bin into physical minute
    for j=1:1:size(vect_time_stamps,1)
    % if the hours are the same and the minutes belong to the
    % interval of the time bin (min-2;min)
    % if the time of the trace route is the same as the higher
    % limit of the time bin, the seconds (columns 7,8) have to
    % be 0 or smaller then 59
    % for comparison translation from ascii to unicode of the
    % timestamps is done
    if (((hours+2) == str2num(horzcat(native2unicode(vect_time_stamps...
       (j,1)),native2unicode(vect_time_stamps(j,2))))) && ((minutes-2)...
       <= str2num(horzcat(native2unicode(vect_time_stamps(j,4)),...
       native2unicode(vect_time_stamps(j,5))))) && (minutes > ...
       str2num(horzcat(native2unicode(vect_time_stamps(j,4)),...
       native2unicode(vect_time_stamps(j,5))))))
        vect_indexes_anomalies(ind)=vect_indexes(j);
    end
    for p=1:1:length(vect_indexes_anomalies)
    if (vect_indexes_anomalies(p)==0)
    if (((hours+2) == str2num(horzcat(native2unicode(vect_time_stamps...
       (j,1)),native2unicode(vect_time_stamps(j,2))))) && ((minutes > ...
       str2num(horzcat(native2unicode(vect_time_stamps(j,4)),...
       native2unicode(vect_time_stamps(j,5))))) || ((minutes ==str2num...
```

```
        (horzcat(native2unicode(vect_time_stamps(j,4)),native2unicode...
        (vect_time_stamps(j,5))))) && (str2num(horzcat(native2unicode...
        (vect_time_stamps(j,7)),native2unicode(vect_time_stamps(j,8))))...
        == 0) || (str2num(horzcat(native2unicode(vect_time_stamps(j,7))...
        ,native2unicode(vect_time_stamps(j,8)))) <= 59)))))
     vect_indexes_anomalies(p)=vect_indexes(j);
    end  end  end  end
    ind=ind+1;
    end
end
vect_indexes_anomalies; % holds the indexes of the timestamps that
                        % correspond to instabilities;
vect_indexes_real_anomalies=vect_indexes_anomalies...
(vect_indexes_anomalies~=0);

% Attach a column with numerically increasing IDs to the timestamps
column=1:length(vect_indexes);
column=column';
vect_indexes = horzcat(vect_indexes,column);
vect_indexes_real_anomalies = vertcat(vect_indexes_real_anomalies,...
zeros(1,length(vect_indexes_real_anomalies)));
for i=1:size(vect_indexes,1)
    for j=1:size(vect_indexes_real_anomalies,2)
        if (vect_indexes(i,1)==vect_indexes_real_anomalies(1,j))
            vect_indexes_real_anomalies(2,j)=vect_indexes(i,2);
        end
    end
end
% vect_index_anomalous_paths holds indexes of traceroutes with detected
% instabilities, plus a columns with incrementing numerical IDs
vect_index_anomalous_paths = vect_indexes_real_anomalies;

% Continue with the matrix processing; IDs have to be assigned to the IPs;
% until now the time stamps for the detected instabilities have been saved
% 42 = "*"   58 = ":"   46 = "."
C_aux=C;   [m,n]=size(C_aux);    D=zeros(m,n);
% Attribute high numbers to traceroutes with timestamps and stars in order
% to separate them from the IPs later on
for i=1:1:m
    for j=1:1:n
            if(C_aux(i,j)~=46 || C_aux(i,j)~=0 || C_aux(i,j)~=42)
                D(i,j) = str2double( native2unicode(C_aux(i,j)) );
            end
            if (C_aux(i,j)==58)
                D(i,1) = 9;
            end
            if (C_aux(i,j)==42)
                D(i,j) = 8;
                D(i,1) = 8;
            end     end     end
% replaces NaN with zeros
D(isnan(D))=0;
% Concatenate the elements on the rows
% sum_col is the column containing the IPs as integers and
% 9*10^13 for timestamps and 8*10^13 for star lines
[m,n]=size(D);
sum_col=[];
for i=1:1:m
    sum_col(i,1)=D(i,1);
    for j=2:1:n
        sum_col(i,1)= str2num(horzcat(num2str(sum_col(i,1)),...
```

```
        num2str(D(i,j))));
    end
end
% sum_col_aux is the column without the time stamps or stars, just IPs
sum_col_aux=[]; index=1;
for i=1:1:length(sum_col)
    if (sum_col(i) < (7.0000e+013))
        sum_col_aux(index,1)=sum_col(i);
        index=index+1;
    end
end
% mat_unique_elements = column containing the unique IPs and their IDs on
% the second column of the matrix
mat_unique_elements = uunique(sum_col_aux);
for i=1:1:length(mat_unique_elements)
    mat_unique_elements(i,2)=i;
end
% Loop for attributing the ID to the different IPs
[m,n]=size(sum_col); [p,q]=size(mat_unique_elements);
for i=1:1:p
    for j=1:1:m
        if (sum_col(j,1)==mat_unique_elements(i,1))
            sum_col(j,2)=mat_unique_elements(i,2);
        end
    end
end
for i=1:1:m
    if (sum_col(i,1) > (8.5000e+013))
        sum_col(i,2)=300;  % attribute ID 300 for timestamps
    end
    if ((sum_col(i,1)>(7.0000e+013)) && ...
        (sum_col(i,1)<(8.5000e+013)))
        sum_col(i,2)=200;  % attribute ID 200 for stars
    end
end
% Take the second column of sum_col, a long column with 300 for
% timestamps, IP-IDs and 200 for stars
routing_vector=sum_col(:,2);
% Rearrange routing_vector as a matrix, route_ids with 300(timestamp tag)
% in the beginning of each column, basically the traceroutes in IDs one
% next to the other
matrix = routing_vector';  route_ids=[]; ind1=1;  ind2=0;
for i=1:1:length(matrix)
    if (matrix(i)~=300)
        route_ids(ind1,ind2)=matrix(i);
        ind1=ind1+1;
    end
    if (matrix(i)==300)
        ind2=ind2+1;
        ind1=1;
    end
end

% At this point the indexes of the traceroutes that have to be kept for
% the detected anomalies are available. The corresponding traceroutes
% from the route_ids.mat have to be separated. Afterwards paths with
% stars are eliminated and eliminate also the correspondent (by index)
% position from the measurement vector

% Read the real delay vector for a certain day
agg_delay = xlsread(strcat(num2str(link),'-agregated-delay-120.xlsx'));
```

```
agg_delay = agg_delay(day,:);

% Delete what is not detected as anomaly from the real delay vector and
% create a vector named agg_delay which will be used later on
for i=1:1:length(agg_delay)
    if (anomalies_vector(i)==0)
        agg_delay(i)=0;
    end
end
% Eliminate zeros from the agg delay vector (create the measurements
% vector - y)
y = agg_delay(find(agg_delay~=0));
% Keep the routes involved in the anomalies
route_ids = route_ids(:,vect_index_anomalous_paths(2,:));

% Eliminate the routes containing stars and the correspondent measurement
% from the y vector
[r,ind_aux,v] = find(route_ids > 100);
route_ids(:,unique(ind_aux))=[];
y(:,unique(ind_aux))=[];

% Create complete routing matrix
routing_matrix=[];
[m,n]=size(route_ids);
for i=1:1:m
    for j=1:1:n
        if (route_ids(i,j)~=0)
            routing_matrix(j,route_ids(i,j))=1;
        end
    end
end
y=y';
% Save measurements vector
savefile='y.mat';
save(savefile,'y');

% Eliminate columns that contain only ones or only zeros and save indexes
% for later mapping between nodes and columns
[m,n]=size(routing_matrix); ind_aux=[]; ind=1;  aux1=ones(m,1);
aux2=zeros(m,1);
for j=1:1:n
    if (routing_matrix(:,j)==aux1)
        ind_aux(ind)=j;
        ind=ind+1;
    elseif (routing_matrix(:,j) == aux2)
        ind_aux(ind)=j;
        ind=ind+1;
    end
end
routing_matrix(:,ind_aux)=[];
savefile='routing_matrix.mat';
save (savefile,'routing_matrix');
```

ROUTINE 8. FOR COMPUTING THE RECOVERY ERROR AND PLOTING IT

```
function [] = test()
K_l=1; % sparsity low
K_h=4; % sparsity high

% Matrix is a predefined sensing matrix; depending on the case it
% can be a random matrix or a binary random matrix
load matrix
fig_nr=1;
A=matrix; % A will be used throughout the routine
measurement_vect_err=[];
vect_data=[];
index=1;
% K = current sparsity
for K = K_l:K_h

% vect_data will contain the test data vector sets
% depending on the current sparsity, K,
% one data vector will be selected
load vect_data
x0=vect_data(:,K);
data_vector=x0;
% Calculate measurement vector based on known matrix and data vector
y = A*x0;

At=A';
% Define matrix structure because it will be required by some
% algorithms
matrix = struct('A',A,'At',At,'y',y,'N', size(A,2));
M=size(A,1);
matrix.Afun  = @(z) matrix.A*z;
matrix.Atfun = @(z) matrix.A'*z;

% General example for two random algorithms
%%% Algorithm 1
data_vect_1 = algorithm_1(data_vector,matrix);
data_vect_1=abs(data_vect_1);
% Sort vector in order to find the highest elements
sorted_data_vect_1=sort(data_vect_1,'descend');
for i=1:1:length(data_vect_1)
    if (data_vect_1(i)<sorted_data_vect_1(K))
        data_vect_1(i)=0;
    end
end
% Calculate error as in formula (5.14)
measurement vect err(index,1)= (norm(data vect 1-abs(x0)))/...
                                norm(abs(x0));

%%% Algorithm 2
data_vect_2 = recovery('lp',data_vector,matrix);
data_vect_2=abs(data_vect_2);
sorted_data_vect_2=sort(data_vect_2,'descend');
for i=1:1:length(data_vect_2)
    if (data_vect_2(i)<sorted_data_vect_2(K))
        data_vect_2(i)=0;
    end
end
```

```
measurement_vect_err(index,2)= norm(data_vect_2-abs(x0))/...
                               norm(abs(x0));
index=index+1;
end

errors_matrix = measurement_vect_err';
% Plot the errors
K=K_l:K_h;
figure(fig_nr)
hold on
set(gca,'FontSize',16)
ylabel('Relative error')
xlabel({'Sparsity'})
title('Recovery performance')

plot(K,errors_matrix(1,:));
plot(K,errors_matrix(2,:));
end
```

ROUTINE 9. COMPUTE THE ACCEPTED NOISE LEVEL FOR PROPER RECONSTRUCTION

```
% Example for 1 algorithm for sparsity 2
function [] = test_part_2_case_2(percentage, ...
            peak_position1,peak_position2)
% percentage = adjust the level of noise over the data
% peak_position1 = position of the first peak entry in
% the data vector; 2 for the second

fig_nr=1;
load routing_matrix_5003_11
matrix=routing_matrix_5003_11;
% choose sparsity; 2 in this example
K=2;
A=matrix;
measurement_vect_err=[];
vector_de_date=[];
recovered_data_2= [];
index=1;
results_mat=[];
data_vect_noisy=[];
n=size(A,2);

% link 5003_11
noise vect = (2*(percentage * 550)/17) + (45+ 10.*rand(n,1));
noise_vect(4) = noise_vect(4) + 10;
% link 5002_35
% noise_vect = (2*(percentage * 700)/13) + (125 + 20.*rand(n,1));
% noise_vect(4) = noise_vect(4) + 10;
% Over the noise add two peaks at the desired positions
x0=noise_vect;
if(K==2)
    x0(peak_position1)= (1-percentage)*550;
```

```
    x0(peak_position2)= (1-percentage)*550;
end
data_vect_noisy(:,index)=x0; % save generated data vector
y = A*x0; % measurements vector

data_vector=x0;
At=A';
matrix = struct('A',A,'At',At,'y',y,'N', size(A,2));
N=size(A,2);
M=size(A,1);
matrix.Afun  = @(z) matrix.A*z;
matrix.Atfun = @(z) matrix.A'*z;

%%% LINPROG
data_vect_lp = recovery('lp_positive',data_vector,matrix);
% Save the recovered data vector in order to compare with the
% original data vector to see if peak positions are recovered
if(K==2)
recovered_data_2(:,1)= data_vect_lp;
end
data_vect_lp=abs(data_vect_lp);
sorted_data_vect_lp=sort(data_vect_lp,'descend');
for i=1:1:length(data_vect_lp)
    if (data_vect_lp(i)<sorted_data_vect_lp(K))
        data_vect_lp(i)=0;
    end
end
measurement_vect_err(index,1)= (norm(data_vect_lp-abs(x0)))/...
                                norm(abs(x0));

% calculate noise level to know for which level the
% positions were recovered
% noise level dB
sorted_noise = sort(data_vect_noisy(:,1),'descend');
maximum1 = sorted_noise(1);
maximum2 = sorted_noise(2);
sorted_noise([1 2])= [];
MSE = (maximum1^2 + maximum2^2 ) / norm(sorted_noise)^2;
noise_dB= 10*log(MSE);
end
```

ROUTINE 10. RECONSTRUCT DATA VECTOR AND COMPUTE NOISE LEVEL

```
% Example of recovery from measurements vector for
% 1 algorithm and sparsity 2
function [] = test_part_3_K_2()
K=2; % sparsity
% measurement vector NTE - Auckland day 11
load y_5003_11
y=y_5003_11;
load routing_matrix_5003_11
A=routing_matrix_5003_11;

recovered_data_2=[];
```

```
recovered_data_3=[];
measurement_vect=[];

At=A';
size(At);
matrix = struct('A',A,'At',At,'y',y,'N', size(A,2));
matrix.Afun  = @(z) matrix.A*z;
matrix.Atfun = @(z) matrix.A'*z;

%%% L1EQ_PD version
data_vect_l1eq = recovery('lp',matrix); % matrix as structure
                                % includes the measurements vector
recovered_data_3(:,1)= data_vect_l1eq; % original recovered data
% Error with respect to the given measurements vector
measurement_vect(1,1)= norm(matrix.A*data_vect_l1eq-y)/norm(y);
data_vect_abs_l1eq=abs(data_vect_l1eq);
% Absolute value recovered data vector
recovered_data_2(:,1)= data_vect_abs_l1eq;
% Calculate the noise level of the recovered data vector
sorted_1=sort(recovered_data_2(:,1),'descend');
maximum(1,1) = sorted_1(1);
maximum(1,2) = sorted_1(2);
sorted_1([1,2])= [];
MSE(1) = (maximum(1,1)^2 + maximum(1,2)^2)/ norm(sorted_1)^2;
noise_dB(1)= 10*log(MSE(1));
% As this point the noise level can be compared to the ones
% determined with routine 9 in order to check if the algorithm
% performs within limits.
end
```