A study of demand oblivious routing algorithms

Jose Luis Almodovar Chico

PVM 2012-072

A study of demand oblivious routing algorithms

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

by

Jose Luis Almodovar Chico
born in Madrid, Spain

Network Architectures and Services Group
Department of Telecommunications
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

A study of demand oblivious routing algorithms

Author: Jose Luis Almodovar Chico
Student id: 1546694
Email: joseluis.almodovarchico@gmail.com

Abstract

Nowadays, people make use of communication networks (e.g. Internet) everyday and expect high quality from the offered services. Providing a good quality of service and optimizing the utilization of the network resources are the main objectives in designing a routing. It has always been believed that traffic demands play an important role in this design because the better the traffic demands are known, the more properly they can be allocated. However, understanding and estimating these traffic demands is not a trivial task.

In 2002 Räcke asked himself an important question: “how important is accurate knowledge of traffic demands for obtaining good utilization of the network?” This marked the beginning of a new kind of routing algorithms: the *demand oblivious routing algorithms* or *oblivious routing*. These routing algorithms ignore the current status of the network when making routing decisions and they only base their decisions on the source, destination of the flows and, in certain cases, some random values. These kind of algorithms have evolved a lot during the last years in three different directions. Nevertheless, according to the knowledge of the author, there has been no study until now that evaluates the performance of algorithms from these three branches.

Firstly, this thesis will present a complete and exhaustive study based on the three papers that originated the different kinds of oblivious routing. Then, a framework will be developed to compare these three algorithms under the same conditions and with a range of metrics from theoretical calculations to more practical ones. Finally, based on the results obtained, possible improvements for these families of algorithms will be suggested.

Thesis Committee:

Chair:	Dr. Ir. F. A. Kuipers, Faculty EEMCS, TU Delft
Committee Member:	Dr. M. M. de Weerdt, Faculty EEMCS, TU Delft
Committee Member:	Dr. C. Doerr, Faculty EEMCS, TU Delft

Preface

First of all, I would like to thank Joanna Pelc and Hendrik van Antwerpen for all their advices and support. I would also like to thank Javier Martin for his collaboration in this thesis and his comments about my report. And, I would like to thank my supervisor Dr. Ir. Fernando Kuipers for his guidance and support along the months that took me to complete this work.

This MSc means the end of a stage in my life. It would be impossible to name one by one all the persons that have made this time unforgettable for me but I couldn't write this acknowledgement section without thinking about Andre, Ruben, Diogo, Andreia, Isabel, Nerea, Mariana, Melati, Pipe, Mauro and all the people from SoSalsa. I would also like to thank Eliza for all the good times during the last year and a half and all her support.

And last but not least, I want to thank my family for being the way they are, for loving me the way they love me and for believing on me no matter what.

To all of them, THANKS!

Contents

List of Figures

Chapter 1

Introduction

Like other current successful technologies, e.g. GPS, Internet was developed by the USA army for military purposes. During its conception, nobody could have expected the importance that Internet has nowadays, with 2,110 millions of users world-wide in 2011.

The truth is that Internet has been a victim of its own success: it was not meant to be such an enormous network with users' expectations and demands far from the ones at the early days of email and world-wide web (WWW). Nowadays users demand access to Internet anywhere, anytime and with a high quality of service (QoS). To satisfy these demands, Internet service providers (ISP) need to design reliable and high capacity networks.

Figure 1.1: Different uses of the Internet nowadays.

However, it does not matter how well a network is designed if it does not include a proper routing scheme and an adequate policy of flow control. In other words, a good design needs to come with a wise selection of the paths that messages will follow and a control policy of the amount of data we introduce to the network at the same time.

Focusing on routing, the question would be: what is a wise selection? Certainly, this question is not an easy one and it has been the point of many studies and researches. One of the conclusions from these studies is the possibility of calculating an *optimal routing* if the volume of traffic the network is expected to carry is known. In this situation, we can calculate the best possible set of routing paths from a Linear Programming problem (LP problem) according to an objective function (e.g. minimizing congestion, latency ...). Therefore, now the challenge is to deduce the Traffic Matrix (TM) that the network needs to route.

Current Internet applications like voice-over-IP, video-on-demand and peer-to-peer are characterized as having *unpredictable* traffic patterns. Besides, these applications have reduced dramatically the time-scales at which the traffic changes dynamically, making it impossible to extrapolate the past patterns to the future. This makes it hard to measure or estimate traffic demands accurately.

There have been several studies that propose models to estimate traffic demands for a network [27] [21], but the most one can get are innacurate approximations. Hence, ISPs over-provision the capacity of their networks in order to be sure that the expectations from their clients can be satisfied. However, this lead to ISP networks being under-utilized to levels below 30%.

These difficulties in obtaining accurate estimations from the traffic demands raise another question: is it mandatory to have a good knowledge of the traffic matrix to design a good routing? In other words, how well can we design a routing with no knowledge of the traffic? And, how well will it perform? The answers to these questions rely on the study of a new way of routing: *oblivious routing*.

Oblivious routing, or *oblivious demand routing*, refers to routing techniques that ignores the current network status when making routing decisions. In fact, for an oblivious algorithm, a routing path only depends on the source node, the target node and some random bits when using random techniques. This way of routing has generated a lot of literature, especially after Räcke demonstrated in 2002 [24] the existence of oblivious routing techniques with surprisingly good results for general networks.

In general, there are three main different research paths on which most of the literature and studies about oblivious routing based their work. The most representative algorithms that originated these three paths are presented in the following three papers:

1. First of all, the paper "*Minimizing congestion in general networks*" [24], where Räcke developed an innovative framework based on decomposition techniques and random probabilities that achieve a link congestion which is within the bound of polylog(n) times the optimal congestion.

2. The second the paper is the one called "*Making Routing Robust to Changing Traffic demands: Algorithms and Evaluation*" [9], written by D. Applegate and E. Cohen,

which developed a polynomial-size linear programming (LP) formulation that calculate the algorithm with the *best oblivious ratio* for a given network.

3. Finally, in the paper "*Oblivious routing of Highly Variable Traffic in Service Overlays and IP backbones*" [16], Murali Kodialam, T.V. Laksham, James B. Orlin and Sudipta Sengupta extended the two-phase routing technique to general networks.

During this thesis we will present the first comparison, to the author's knowledge, between these three different algorithms. Our focus is to develop a framework to perform realistic tests, according to topologies and traffic demands, from a network administrator's point of view. We will evaluate the performance and robustness of these routing strategies, and compare them to the *Open Short Paths First* (OSPF) algorithm.

Finally, we will conclude whether *oblivious routing* is a suitable technique to be used in real scenarios and we will identify which of the three most representative approaches performs the best.

1.1 Structure of the thesis

The remainder of this thesis will be organized as follows. In **Chapter 2** we will talk about prior art and we will introduce oblivious routing. We will dedicate **Chapter 3** to study and analyze meticulously the three papers mentioned above. We will present in **Chapter 4** the framework used to test the algorithms and the results of these tests. Finally, in **Chapter 5** we will list the conclusions derived from this thesis work and we will include a section to propose future work.

Chapter 2

Oblivious routing

2.1 Routing algorithms

The main purpose of a communication network is to *carry* information between its nodes. In order to exchange this information, a set of *paths* or *routes* between source and destination nodes must be determined.

The mission of a *routing algorithm* is choosing the best paths in order to satisfy a goal. This goal normally consists of *minimizing a cost* (e.g., congestion, latency, ...) and it is generally dictated by requirements of the communication network and the service offered. The main objective is to provide good quality of service and to optimize the utilization of the network resources.

However, choosing a routing algorithm is not a trivial decision and there are several factors that need to be considered [12]:

1. Load balancing: A proper algorithm will balance the load across the network channels, even in situations with non-uniform traffic patterns.

2. Path length: A good routing algorithm also keeps path lengths as short as possible, reducing the number of hops and the overall latency of a message.

3. Resiliency: This feature is critical for systems with high-reliability demands. The ability of an algorithm to adapt in case of failure can make the difference between a system being able to continue working or not.

Selecting a proper *set of routing paths* is an important decision to make, which directly affects the performance of the network. Therefore, this decision needs to be based on reliable parameters, but not all the existing routing algorithms consider the same information at the time of making a decision. According to this point, we classify the routing algorithms in [26]:

- *Adaptive routing algorithms*: These strategies base their decisions on the current network state (e.g., paths chosen, knowledge of the requested traffic, ...). The term *adaptive* refers to their ability of dynamically adjusting the paths to the current state and demands of the network. Although this approach seems ideal, it has severe disadvantages that make its implementation a difficult task and, therefore, a not very cost-effective solution:

- It requires the operator to continuously monitor the network state and distribute this information through the nodes of the network, which produces extra overhead that affects the performance of the network.
- The network information needs to be reliable but the unpredictable Internet traffic patterns make the process complex.
- Adapting routes can take a while depending on the amount of information to process, the number of parameters to take into account, and the size of the network. This processing time directly increases the delay of the traffic carried by the network, which can be unacceptable for some services.

- *Oblivious routing algorithms*: These schemes ignore the network state entirely. The routing decisions do not depend on earlier requests or current traffic at the network. This makes its implementation much easier than in the case of adaptive routing. Besides, there is no overhead added to the network and neither extra delays caused by any path adaptation process because the routes are picked up beforehand and stay fixed.

  The main idea behind *oblivious routing* is providing a routing that, although it may not perform optimally in specific situations, can perform better than *adaptive algorithms* in a range of different cases without modifying the selected routes.

In the next section, we will go deeper in the study of *oblivious routing*. Apart from giving a formal definition, we explain the origins of the idea and its development, which will help the reader to understand the algorithms we will present in Chapter 3.

2.2 Oblivious routing

Oblivious routing, also referred to as *demand oblivious routing*, is a routing technique that picks a set of routing paths between each pair of nodes in advance without knowledge of traffic demands. The path taken by a packet only depends on its source-destination pair and, maybe, a random choice to select one of the options (see Figure 2.1). Formally, *oblivious routing* can be defined in terms of probability as [28]:

"*An oblivious routing strategy is specified by a set of paths P and a function w assigning a weight or flow ratio to every path in P. w has the property that, for every source-destination pair (s,t), the set of paths $P_{s,t}$ for $\sum_{q \in P_{s,t}} w(q) = 1$.*"

Oblivious routing techniques have been under study during a long time. The first approaches applied oblivious routing theory into *specific* network instances. The most successful and famous result from this period was work done by Valiant and Brebner [30],

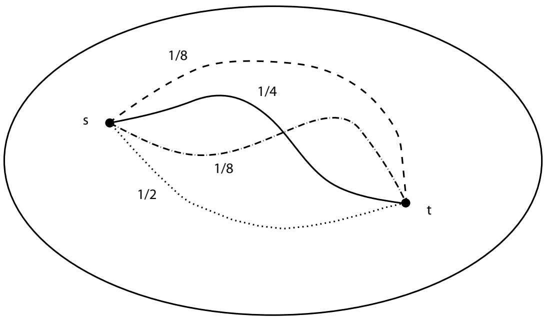


Figure 2.1: A system of optional paths for the pair (s,t). The numbers indicate how a flow from s to t has to be split among the paths.[28]

who performed the first worst-case theoretical analysis for oblivious routing on a *hypercube* network. This oblivious routing technique is known as Valiant's trick and it works in the following way:

Algorithm: *For a network $G = (V,E)$ for every packet with source-destination pair (s,t), the algorithm chooses a random intermediate destination v with probability $cap(v)/cap(V)$ and sends the packet first along a flow path from s to v and then along a flow path from v to t.*

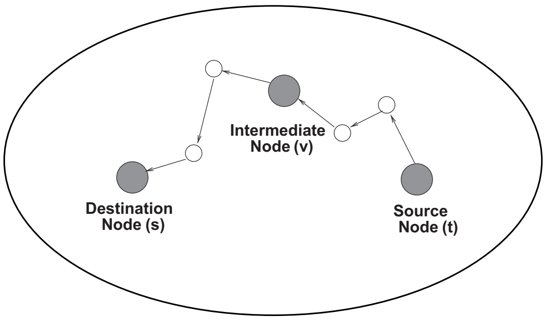


Figure 2.2: For every pair (s,t), the traffic is sent through and intermediate node v.

Where V is the set of nodes from the hypercube network and E the set of links, and the operator $cap()$ represents the capacity[1] of the node v or the node set V, respectively. Using Valiant's trick in a *d-dimensional hypercube*, any *balanced multi-commodity flow problem*[2] (BMFP) can be routed with congestion at most 4d [28].

[1]Given a graph $G = (V,E)$ with links capacities c, we define the capacity of a node $v \in V$ as:

$$c(v) = \sum_{w \in V} c(link(v,w))$$

and the capacity of a node set U is defined as $c(U) = \sum_{u \in U} c(u)$.

[2]Is a multi-commodity flow problem in which the sum of the demands of the commodities originating and terminating in node v is equal to $cap(V)$ for every $v \in V$. We define *multi-commodity flow problem* as a network flow problem with multiple commodities (flow demands) between different source and sink nodes.

In 2002 Räcke presented a framework for minimizing congestion in *general undirected networks* [24]. This framework consisted of transforming any network into a tree with nearly equivalent *communication characteristics* (e.g., the same bandwidth between node-pairs, the same bottlenecks, ...), where the *congestion-related problem* will be solved and then translated back into the original network.

Surprisingly, the use of the framework in general undirected networks achieves a *competitive ratio*[3] of $O(log^3 n)$ with respect to the congestion of the network links, where n is the number of nodes of the network. The main drawback is that the algorithm requires solving *NP-hard problems* and, therefore, is non-polynomial time algorithm.

Nevertheless, Räcke's result represented a milestone in the study of oblivious routing algorithms. This result was improved and extended in various ways by several researchers:

- In 2003, Räcke, Azar et al. [10] presented a polynomial time algorithm based in linear programming techniques that selects the optimal oblivious routing for any network. However, part of the solution consisted on using the complex *ellipsoid method*[4] to cope with the infinite number of constraints produced by the Linear Programming (LP) formulation.

- Applegate and Cohen [9] published an algorithm that computes the optimal routing scheme by an LP problem with polynomial number of variables and constraints, which meant an enormous improvement in running time and complexity compared to the Ellipsoid solution.

- Harrelson et al. [14] presented an improved partitioning algorithm for obtaining trees with nearly equivalent communication characteristics, that works in polynomial time.

Other researchers followed different paths in their works. In 2004, based on Valiant's trick, two-phase routing was proposed as an oblivious routing scheme for general networks [17][31].

In general, most of the researches and innovations done on *oblivious routing* are based on or make direct reference to one of the three researching paths mentioned. In the following chapter, we will explain each of them in detail.

[3]The performance of an algorithm A is evaluated by comparing it to an optimal algorithm OPT, which has full knowledge of the input sequence in advance and can process it optimally. Given an input sequence σ, let $C_A(\sigma)$ and $C_{OPT}(\sigma)$ denote the congestion produced by A and OPT, respectively when processing σ. Algorithm A is called *c-competitive* if there exists a constant a (independent of σ) such that $C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$, for any sequence σ [24]. This method to compare algorithm performances is known as *competitive analysis*.

[4]The *ellipsoid method* is an iterative method for minimizing convex functions, which finds an optimal solution in a finite number of steps. The ellipsoid method generates a sequence of ellipsoids whose volume uniformly decreases at every step, thus enclosing a minimizer of a convex function.

Chapter 3

Oblivious routing algorithms

3.1 Introduction

Our objective in this chapter is to study the three most representative different approaches in the field of *oblivious routing*. We will explain them from a theoretical and a practical point of view. At the end of the chapter, we will present some conclusions that will be useful to establish a comparison framework.

The three *oblivious routing* methods we will study are:

- The first method belongs to the results presented by Räcke in 2002. In the paper “*Minimizing congestion in general networks*” [24], Räcke developed an innovative framework based on decomposition techniques and random probabilities that achieves a congestion within the bound of $polylog(n)$ times the optimal congestion.[1] However, the techniques used by Räcke derive in NP-hard problems and, hence, approximations started appearing. We will work with the approximation developed by Bryan Hughes in his PhD dissertation [15].

- The second method refers to the paper called “*Making Routing Robust to Changing Traffic demands: Algorithms and Evaluation*” [9] written by D. Applegate and E. Cohen. The authors developed a polynomial-size Linear Programming (LP) formulation that gives the *best oblivious routing* for a given network. In other words, the algorithm obtains the oblivious routing that minimizes the congestion on the network compared to the optimal routing the most.

- The last method is called two-phase routing, developed in the paper ”*Oblivious routing of Highly Variable Traffic in Service Overlays and IP backbones*” [16] written by Murali Kodialam, T.V. Laksham, James B. Orlin and Sudipta Sengupta. This approach is based on the work from Valiant [30], which basically consists of routing the traffic through a set of intermediate nodes instead of routing it directly to the respective destinations.

[1]The **congestion** is defined as the maximum relative load over all network links, where relative load of a link is defined as total amount of traffic at the link divided by the capacity of the link.

In the following sections, we will present these algorithms together with their respective pseudo codes and LP formulations. Our implementations were developed in C++. Parameters and data were introduced to the program in xml format and the LP formulations were solved with Cplex [1].

3.2 The network model

The three algorithms work with different kinds of networks: *directed* and *undirected networks*. Therefore, we need a network model that includes both cases and uses each one when it is required.

- *Undirected networks* will be modeled as $G = (V,E)$, where V represents the set of *nodes* of the network and the E the set of *links*. We use n to denote the number of *nodes* and l the number of *links* ($|V| = n$ and $|E| = l$, respectively).
- *Directed networks* are going to be represented by $G = (V,A)$, where V represents the set of *nodes* and A the set of *arcs*. We use n to denote the number of *nodes* and m the number of *arcs* ($|V| = n$ and $|A| = m$, respectively). For any *node* $i \in V$, we denote by $IN[i] \in A$ the sets of *arcs* directed into a *node* i and by $OUT[i] \in A$ the set of *arcs* directed out of the *node* i.

A *weight* function $cap : E \to R_0^+$ describes for a *link* $e \in E$ the *link-capacity*. In the case of a *directed network*, we will be talking about *arc-capacity*.

The demand traffic matrix will be defined by a matrix D: a $n \times n$ nonnegative matrix which diagonal entries are 0. Each entry of the matrix, $d_{i,j}$, specifies the traffic that needs to be carried from the source i to the destination j.

We use the notation f and g to refer to routings and flows respectively:

- The routing is specified by a set of values $f_{i,j}(e) \geq 0$ that represent the *fraction* or *percentage* of demand from i to j ($d_{i,j}$) that is allocated at $e \in E$ ($f_{i,j}(a) \geq 0$ in the case of *directed networks*). To obtain the traffic allocated in a link, we will need to compute $f_{i,j}(e) \cdot d_{i,j}$.
- The shorthand $g_{i,j}(e) \geq 0$ denotes the traffic demand value from i to j present at $e \in E$ ($g_{i,j}(a) \geq 0$ in the case of *directed networks*). Therefore, $g_{i,j}(e) = f_{i,j}(e) \cdot d_{i,j}$

3.3 Räcke's algorithm and its approximation

3.3.1 Introduction

As we already mentioned in previous sections, Räcke developed a mechanism to construct an *oblivious routing algorithm* for any *symmetric network* [24] (networks where link-capacities are the same in both directions of the link, like most of large backbone networks) or, in other words, for *undirected networks*. However, some of the techniques used lead to NP-complete problems and, hence, to exponential-time procedures.

In Bryan Hughes' approximation [15], the author circumvented these exponential-time procedures by replacing them with more resource-friendly techniques. Although these actions invalidate the *guaranteed bounded congestion* from Räcke's method, this loss can still be acceptable for many applications.

In the following sections, we will present both techniques and a detailed implementation of Bryan Hughes' approximation.

3.3.2 Räcke's algorithm

The crux of the algorithm consists of finding a good *decomposition tree* and then showing that it is possible to route in the original graph almost as well as in the tree graph, with some minimal losses. In other words, Räcke transforms a general network into a tree network with the desired *equivalent communication characteristics* (e.g., the same bandwidth between node pairs, the same bottlenecks, etc), finds the tree paths between every pair of nodes, and then transforms these paths back to the general network. Therefore, this framework can be applied to any problem that has an efficient tree solution. This process is called the *bisimulation technique*.

The Bisimulation Technique

This technique is carried out in three steps:

1. Transforming a given problem instance I (i.e. a routing request between nodes $u, v \in V$) on the general graph G to a problem instance I_t on the tree network T: I is translated into T using a predetermined mapping function $\pi_1 : V \to V_t$ (which corresponds to a hierarchical decomposition that generates a *Virtual Tree Network*) resulting on a problem instance I_t. Therefore, when a message is sent between two nodes $u, v \in V$, a corresponding message is sent between two nodes $\pi_1(u), \pi_1(v) \in V_t$.

2. Finding the path between nodes $\pi_1(u)$ and $\pi_1(v)$, which is the solution to I_t: To find a path in a tree network, we only need to find the two independent paths from nodes $\pi_1(u)$, $\pi_1(v)$ to the root node, concatenate them at the root node and then remove any common intermediate nodes.

3. Transforming the solution to I_t on the tree network back onto a general graph G: This transformation is performed by using a randomized mapping $\pi_2 : V_t \to V$, where $\pi_2(\pi_1(v)) = v, \forall v \in V$. This mapping needs to be randomized (i.e. it selects one of many paths randomly), because there are typically more nodes in the tree T than in the graph G. Said differently, one node in G is represented by multiple nodes in T and, thus, there are multiple paths in G given a single path in T.

The Virtual Tree Network

As we already mentioned, the virtual tree network is constructed using a hierarchical decomposition H. H starts with a general graph G and divides it into smaller sub-graphs. These sub-graphs are recursively partitioned until the last sub-graphs contain a single node in G. The partition of the graph is performed in such a way that the cut is of minimum sparsity,[2] which minimizes congestion in the network.

This decomposition process results in a natural tree, T_H, which is referred to as the *decomposition tree corresponding to the hierarchical decomposition H*. Each tree node corresponds to a *set* or *cluster* of nodes $H_V = \{v : V \in V\}$ from the general graph G. This decomposition method is also called *laminar* since the subgraphs resulting from the cuts form a laminar set.

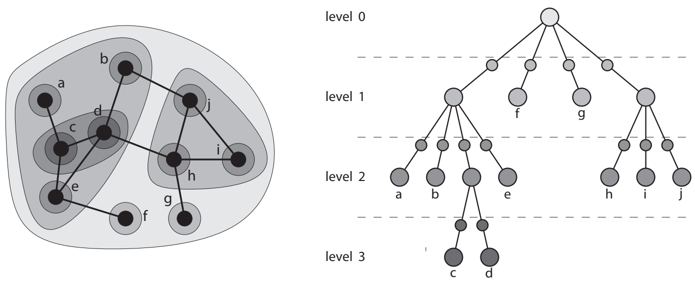


Figure 3.1: A hierarchical decomposition of a graph and the associated virtual tree [25].

The generated tree network has the following properties:

- The *bandwidth* or *capacity* of a link in the tree is defined as:

$$cap_t(x_t, y_t) := cap(X,Y) := \sum_{x \in X, y \in Y} cap(x,y) \tag{3.1}$$

  which describes the total bandwidth that is available between nodes of the subset X and nodes of the subset Y. Where $cap(x,y)$ denotes the bandwidth or capacity for the link $(x,y) \in E$. Besides, $X \subset Y$ or $Y \subset X$ because one subset represents the cluster from a parent tree node and the other represents the cluster from one of its children.

- The *level of a tree node* is defined as $l+1$, where l is the level from the parent tree node. The level of the root node is 0.

- The *weight of a tree level* is defined by the function $w_l : 2^V \rightarrow R_0^+$ for every level $l \in \{0, ..., height(T_H)\}$. Informally, we can express the weight as the capacity of all links leaving the set for nodes in the higher level nodes:

$$w_l(X) := \sum_{e \in V \times V \quad and \quad level(e) \leq l} cap(e) \tag{3.2}$$

One may still ask why a routing performed in this virtual tree network is that efficient. The fact is that routing a *multi-commodity flow problem* on the associated virtual tree network *decreases* the maximum link congestion compared to the optimal flow on the original

[2]The **sparsest cut** problem is to bipartition the set of nodes so as to minimize the ratio of the number of links across the cut divided by the number of nodes in the smaller half of the partition.

graph. Although we will not go into details, the main reason lies in the fact that the tree links represent the upper-bounds on the capacity between a cluster or subgraph and the rest of the graph.

Räcke presented all required mathematical arguments to valid all these methods and results, together with more extensive explanations. This level of detail is beyond the scope of this MSc thesis, but for interested readers we strongly recommend Section 2.3.3 and Section 2.3.4 from Räckes Ph.D. dissertation [25].

3.3.3 Bryan Hughes' approximation

The key of Hughes' approach is to substitute some complex techniques used by Räcke by more resource-friendly techniques. The techniques to be replaced are: *routing tree generation* and *path selection*.

Routing tree generation

The objective is selecting a *partitioning algorithm*, easy to implement, to translate the general graph G into a tree network T_H. Hughes decided to use an algorithm that performs single cuts: the *binary partitioning algorithm* [18].

Before performing any cut, the algorithm calculates the shortest paths, based on hop count, among all the nodes in the graph G. This information will be used to choose the following *cut-points*. Once the calculation is performed, the following routine is repeated at every (sub)graph until the leaves of the tree only contain a single node:

1. Find the shortest path l_{path} from all of the nodes in the (sub)graph and mark the end nodes as u_{end} and v_{end}.
2. Order the remaining nodes in the (sub)graph according to their distance from u_{end} (according to the results from the shortest path calculation performed at the beginning) and set the *cut-point* c in the middle of the list.
3. Execute the "*rule of thirds*": The cut-point c is shifted until the distance between c and u_{end}, identified as c_{dist}, is within the range $l_{path}/3 \leq c_{dist} \leq 2 \cdot l_{path}/3$. This rule prevents that nodes far away from each other are grouped together.

Once the tree network is generated, we calculate the weights for each graph node within every tree node X. For this purpose, the author makes use of the formula proposed by Räcke (Equation 3.2):

$$w_l(u \in X) := cap(u)/cap(X) \tag{3.3}$$

This formula states that the weight is defined as the capacity ratio between node u and all graph nodes in the tree node's parent, represented by the set X. In the case of the *root node*, the capacity is defined as the total capacity of the network (Equation 3.4).These weights will be used to take routing decisions as explained in the following section.

$$w_l(u \in V) := cap(u)/cap(V) \tag{3.4}$$

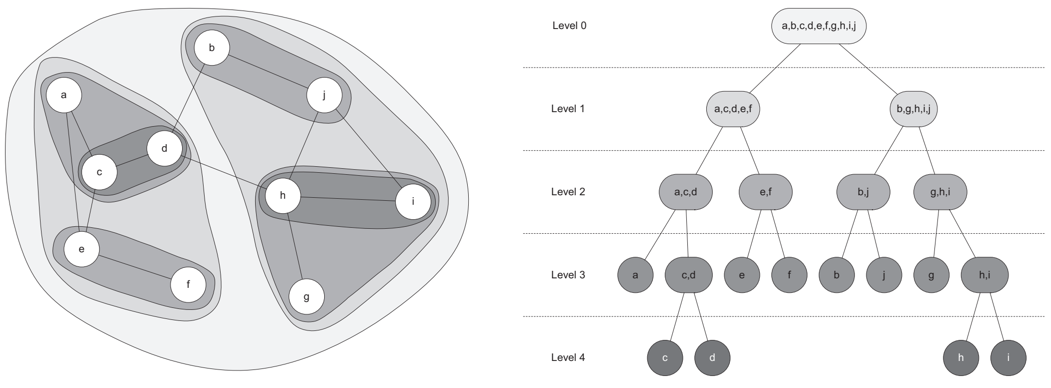


Figure 3.2: A decomposition of a graph using the binary partitioning algorithm and the associated decomposition tree [15]. It would be interesting for the reader to compare this figure with Figure 3.1

Path selection

The first goal is finding the *tree path*: the path from the source node to the destination node in the tree network. For this purpose, after localizing the source and destination leaves, we trace the paths from both of them to the root node and then we concatenate them.

The second step is translating this *tree path* into a *graph path*: we translate the *tree nodes* (which represents a set of graph nodes) from the tree path into single *graph nodes*. This translation is performed by assigning random values to each graph node of a tree node and then selecting based on this randomization and the *weight* of the nodes (Equation 3.3).

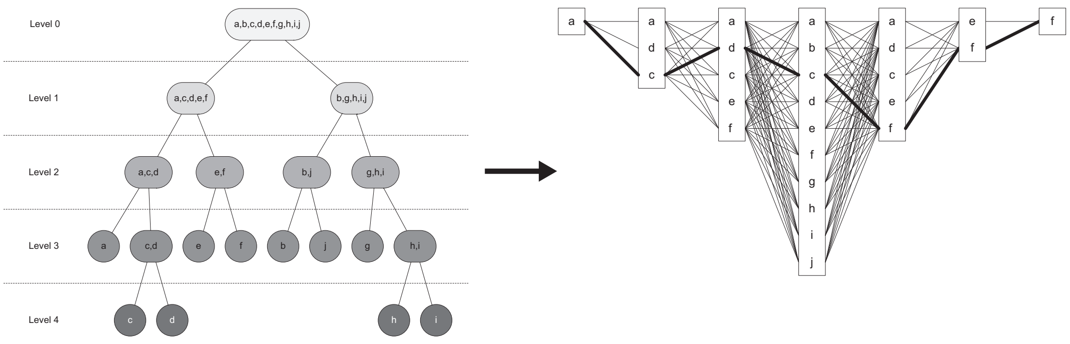


Figure 3.3: Example of network path selection [15].

The drawback of this process is that the path selection algorithm gets "*stuck*" if it chooses a graph node *non-connected* in the original graph to any of the graph nodes of the set from the next tree node. When this situation occurs, the algorithm must step backwards, removes the selected graph node (so it will not be chosen again) and then selects a new one. In the following section, we will introduce an improvement to the algorithm which will easily solve these situations.

Figure 3.4 presents an example of a "*stuck*" situation: in this case, node d is selected in the fifth iteration but it has no connectivity with nodes e and f. Therefore, the algorithm steps backwards, removes d from the options and selects a new node (in this case node e).

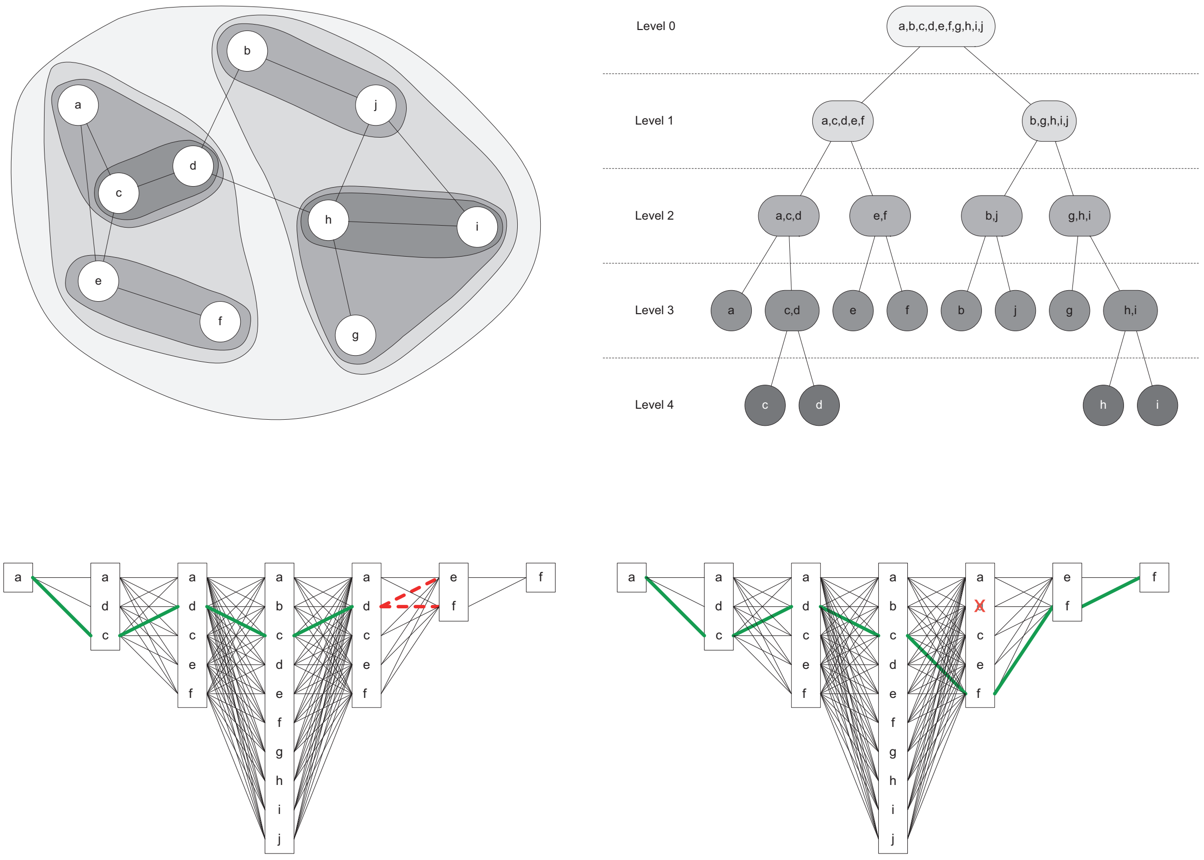


Figure 3.4: Example of a "stuck" situation and its resolution

Finally, we remove any redundant graph node and every possible cycle in the graph path selected.

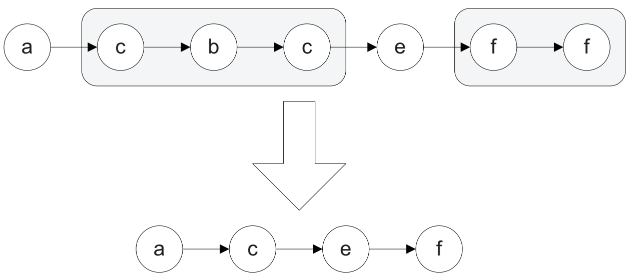


Figure 3.5: Path reduction [15].

The fact that the path selection decision is based on random values means that there is more than one possible solution in most cases. The author suggests running the selection algorithm a high number of times (100 times according to the PhD dissertation [15]) and then choosing the most visited path as the chosen graph path.

3.3.4 Implementation

Our implementation is based on the one presented by Bryan Hughes in his PhD dissertation [15]. We will pay attention to the reasons why some decisions were taken and we will present small changes introduced in the approximation to improve the algorithm performance and results.

We can divide the implementation into three independent parts: *binary partitioning algorithm*, *node weight algorithm* and *path selection algorithm*. Each part will be accompanied by a pseudo code that reflects steps taken and operations performed.

Binary partitioning algorithm

The first step is to calculate the shortest paths among every node from the graph G. We will use a common implementation from the Dijkstra algorithm [23] where we assign a cost equal to 1 to every link. In this way, the cuts would be based on hop counts and not on other factors (e.g. capacity of the links), contributing to obtain the most connected subgraphs possible during consecutive cuts.

There are two important reasons for the shortest path calculation to be the first step of the algorithm:

- On one hand, calculating all the shortest paths at once at the beginning of the process and storing these results so the algorithm can use them later will save time of computation.
- On the other hand, although the author does not specify anything about it, the algorithm is able to perform cuts that, in some cases, result on *non-connected subgraphs* (see Figure 3.6). Dijkstra's algorithm cannot be used in these situations because it does not converge into a solution in non-connected subgraphs. Calculating the shortest paths at the beginning of the process helps us avoid this problem.

All the actions performed within the *decomposition of the graph* and the *generation of the tree graph* are detailed in the following pseudo code. The pseudo code presents a recursive method which runs until the leaves of the tree only contain a single node.

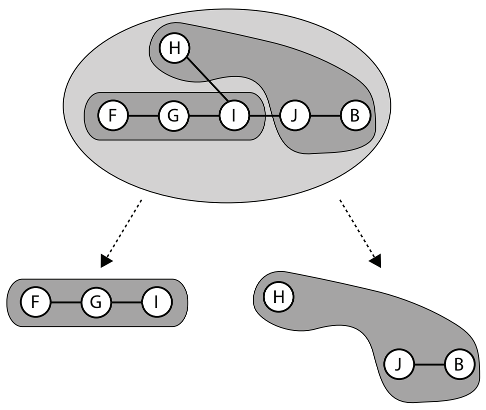


Figure 3.6: Example of non-connected subgraphs resulting from an iteration of the partioning algorithm.

Algorithm 1 Partition algorithm

```
1:  **given** shortest path set $P_{min}$ and (sub)graph $G_p$
2:  **if** $|P| = 1$ **then**
3:      store the node in a leaf (the left and right pointers will be NULL)
4:      return
5:  **else**
6:      find $p_{max} \in P_{min_p}$, where $P_{min_p} \in P_{min}$ is the shortest path set for $G_p$
7:      sort the nodes in $G_p$ according to distance from the endpoint $u_{end} \in p_{max}$
8:      **set** cut point $c$
9:          **if** $|P| = 2$
10:             $c = 1$
11:         **else**
12:             $c = |G_p|/2$ (rounded always to the higher volume)
13:         **end if**
14:     **if** $c_{dist} \leq l_{path}/3$ **or** $c_{dist} \geq 2 \times l_{path}/3$ **then**
15:         shift until $l_{path}/3 \leq c_{dist} \leq 2 \cdot l_{path}/3$
16:     **end if**
17:     group nodes below $c$ with $u_{end}$ and nodes above $c$ with $v_{end}$
18:     create child nodes of $G_p$, $G_{c_1}$, and $G_{c_2}$, containing the two groups
19: **end if**
```

Node weight algorithm

The calculation of the node weights is a simple operation that we detail with the following algorithm.

Algorithm 2 Node Weight Algorithm

```
1: given (sub)graph $G_c$ and its tree node parent $G_p$
2: for all $v \in G_c$ do
3:     set $w_l(v) = 1 + cap(v, G_p)$
4: end for
```

The capacity between a graph node v in the *child tree node* G_c and all graph nodes in the *parent tree node* G_p is calculated and stored as the weight from node. In the case of the root node, the weight will be calculated as the capacity between a graph node v and the rest of the graph nodes.

We will force the weights of all the nodes in the cluster of the tree to be 1 by default. We need to take this action because the algorithm for finding graph paths can get stuck due to the non-connected subgraphs generated by the partitioning algorithm. In this case, the same nodes can be selected until we reach a cluster where there is connectivity with other nodes.

Path selection algorithm

The *path selection algorithm* is responsible for constructing the *tree path* between both node leaves (source and destination) and then translating this *tree path* into a *graph path*. This translation is based on the weights of the nodes and some random values. However, because the partition algorithm can generate non-connected subgraphs, this translation can lead to disconnected graph paths. Therefore, in order to prevent these results, we introduce one more parameter in the decision process within the translation: the connectivity between the possible chosen nodes.

Algorithm 3 Path Selection Algorithm

```
1:  given source node $u$ and destination node $v$
2:  find tree paths $P_{t_u}$ and $P_{t_v}$, and concatenate them (only one root node)
3:  set the current tree node index, $x$, to point to the source node $u$ (i.e. to index 1)
4:  repeat
5:      generate random numbers, $R$, for all nodes in the next tree node
6:      set $P = R \cdot W \cdot connectivity\_to\_node_{x-1}$
7:      select graph node, $g$, with the highest probability $p \in P$
8:      if there is no $p > 0$ then
9:          remove the node from future decisions
10:         decrement $x$
11:     else
12:         store the graph node $g$ into the network path
13:         increment $x$
14:     end if
15: until $x$ point to destination node $v$
16: remove any redundant node and cycle from the network path
```

3.4 Applegate and Cohen algorithm

3.4.1 Introduction

In previous sections, we presented the *oblivious routing algorithm* developed by Räcke that provides a uniform congestion bound of $O(log^3 n)$ for every undirected graph. However, a better oblivious routing solution with a lower *competitive ratio* than the bound of $O(log^3 n)$ may potentially exist for a specific graph.

Based on this possibility, Y. Azar, E. Cohen and Räcke et al. wrote a paper [10] which presents a polynomial time construction that gives the *best possible oblivious routing* scheme for any given network. The techniques used in this paper are based on Linear Programming (LP) and differ completely from everything presented in previous sections. However, although this LP problem has a polynomial number of variables, it has infinitely many constraints (one set of constraints for every possible traffic matrix). To solve this situation, the authors applied the *Ellipsoid method* [2] with a *separation oracle*, but even this method was still too complex.

In [9], Applegate and Cohen concluded that the *optimal oblivious routing scheme* could be computed by an LP problem with a polynomial number of variables and constraints. They developed a simpler model by formulating the *dual problem* from the *separation oracle* (check section 3.4.2) and using it to replace the exponential set of constraints. This new and simpler approach represents an enormous improvement on the running time and simplicity compared to the use of the ellipsoid method.

In the following sections, we will explain these two methods. The explanation and details of the proofs and theorems provided by the authors, together with the *ellipsoid method* itself, are considered to be beyond the scope of this MSc thesis.

3.4.2 Räckes LP problem and the ellipsoid method

As said in the previous section, it was shown in [10] that an optimal oblivious routing can be computed by solving an LP problem with a polynomial number of variables but infinitely many constraints. Before introducing the LP formulation from this approach, it is important to clarify that the authors work with directed networks. They present a well-known reduction from undirected graphs to directed graphs that replaces each undirected link $e = (u, v)$, with capacity $cap(e)$, with the directed gadget $u;x;y;v$ which consists of five arcs: four arcs $e_1 = (u;x); e_2 = (v;x); e_3 = (y;u); e_4 = (y;v)$, all of them with infinite capacity, and the arc $e_5 = (x;y)$ with capacity $cap(e)$. Therefore, we will only consider arcs in this section.

The following formulation defines the LP problem, which will be referred to as "*master LP*". The objective is to minimize the congestion of the network by choosing the routing that uses the lowest amount of capacity in every link (Equation 3.5).

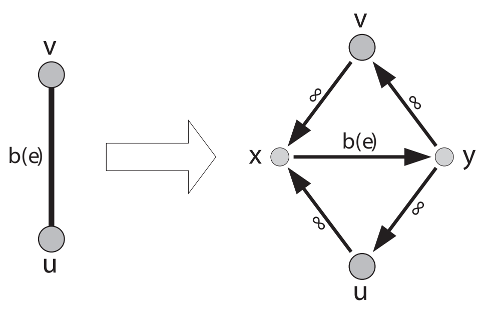


Figure 3.7: The reduction from undirected to directed graphs [25]

$$
\begin{array}{rl}
\textbf{objective} & \textit{minimize} \quad r \\
\textbf{constraints} & \\
1. & \text{f is a routing} \\
2. & \forall \text{arc}(a) \in A, \forall \text{TMs D} \quad \text{with} \quad \text{OPTU(D)} = 1: \\
 & \qquad \sum_{ij} \frac{f_{ij}(a) \cdot d_{ij}}{cap(a)} \leq r
\end{array}
\tag{3.5}
$$

The variables of the LP problem are the set of routing variables $f_{ij}(e)$ and the minimization variable r. The routing variables $f_{ij}(e)$ denote for a link e the fraction of flow from i to j of value 1, which means that the value from the variable has to be within the range $[0,1]$. We can route the traffic demands d_{ij} using a routing scheme f by simply scaling each unit flow f_{ij} by the factor d_{ij}. The demand matrix (or traffic matrix) $D = [d_{ij}]$ is an $n \times n$ non-negative matrix where the diagonal entries are 0 and the rest ≥ 0.

The condition that f is a routing implies a set of linear constraints, known as "*routing constraints*", which defines a unit *single-commodity flow* from i to j of value 1:

$$
\begin{array}{ll}
\forall a \in A \quad \forall i \forall j \neq i: & f_{ij}(a) \geq 0 \\
\forall i \forall j \neq i: & \sum_{a \in OUT(i)} f_{ij}(a) - \sum_{a \in IN(i)} f_{ij}(a) = 1 \\
\forall k \forall i \neq k \forall j \neq k, i: & \sum_{a \in OUT(i)} f_{ij}(a) - \sum_{a \in IN(i)} f_{ij}(a) = 0
\end{array}
\tag{3.6}
$$

Constraint 2 is referred to as the "*congestion constraint*". To explain the meaning of this constraint, we need to introduce some definitions:

- $\sum_{ij} f_{ij}(a) d_{ij}$ represents the traffic on the arc a carried by the routing f. The total flow on the edge divided by its capacity $cap(a)$ is defined as congestion on the arc. The congestion of the network using a routing f and with a demand matrix D is:

$$congestion(f,D) = max_{a \in A} cong(a,f,D)$$

- Constraint 2 is evaluated for every arc of the network and for every D with $OPTU(D) = 1$, where $OPTU(D)$ makes reference to the optimal utilization of the network for the traffic matrix D. We can define the utilization of a network as the maximum *arc utilization* while using a routing f. Hence, the optimal utilization of the network refers to the utilization of the network made by the optimal routing, which is the routing that minimizes the maximum arc utilization $(MxAU)$:

$$OPTU(D) = min_f MxAU(f,D)$$

Finally, we only include traffic matrices with $OPTU(D) = 1$ because the competitive performance ratio is invariant to scaling. Thus, instead of considering all TMs it is sufficient to consider the TMs D with $OPTU(D) = 1$.

If we must evaluate the Constraint 2 for every arc of the network and for every traffic matrix with $OPTU(D) = 1$, it results in an LP problem with infinitely many constraints. In order to solve this problem, the authors applied the *ellipsoid method* using the following separation oracle for the constraints:

- **Input:** A network G, a capacity function $b(.)$, and an assignment of all variables from the master LP.
- **Output:** Either, a confirmation that all constraints in the master LP are fulfilled, or a "violated constraints". This means:
  - a violated routing constraint (master LP constraint 1) that shows that f does not constitute a routing scheme
  - a violated congestion constraint (master LP constraint 2)
- **Implementation:** the constraints from the master LP can be tested by solving the following "*slave LP*" problem for each arc a and testing if the objective is $\leq r$ or not:

$$\begin{array}{rlll}
\textbf{objective} & max & \sum_{ij} f_{ij}(e) \cdot d_{ij}/cap(e) & \\
\textbf{constraints} & & & \\
1. & \forall & pairs \quad i \to j: & \{g_{ij}(a) | a \in A\} \text{is a flow demand } d_{ij} \\
2. & \forall & edges \quad e: & \sum_{ij} g_{ij}(e) \leq cap(e) \\
3. & \forall & pairs \quad i \to j: & d_{ij} \geq 0
\end{array} \tag{3.7}$$

Summarizing, this method gives the oblivious routing scheme that achieves the *best possible* competitive performance ratio for a given network G. In the next section we will explain how, from the *master* and *slave* LP formulations, Applegate and Cohen derived a simpler LP problem.

3.4.3 Applegate and Cohens LP formulations

The main objective of Applegate and Cohen [9] is to deliver a simpler LP with a polynomial number of variables and constraints that allows to efficiently process larger networks. The easiest way is to directly combine the *master* and *slave* LPs, which will result in a single polynomial size LP instance. However, such an approach results in a quadratic problem where we cannot apply the solving methods for LP problems anymore. The authors tackled this obstacle by taking the *dual problem* from the separation oracle (*slave problem* from the previous section and using it to replace the exponential set of constraints of the master LP problem.

This method works for *undirected networks*, but is based on the LP problem from section 3.4.2. Therefore, a relation between *links* and *arcs* has to be introduced: the term $link-of(a)$ is the link between the two end points of an *arc a*. The capacity of the *link* is the sum of the capacities from the *arcs*. Regarding flows and routing, they used the following notation for the sum of flows on both directions of a link:

$$f_{i,j}(e) = \sum_{a:link-of(a)=e} f_{i,j}(a) \tag{3.8}$$

All these modifications lead to the following LP formulation:

$$\begin{array}{rl}
\textbf{objective} & \textit{minimize } r \\
\textbf{constraints:} & \\
1. & f \text{ is a routing} \\
 & \forall\ links\ e \in E: \\
2. & \quad \sum_{ij} cap(h) \cdot \pi_e(h) \leq r \\
3. & \quad \forall\ pairs\ i \to j : f_{ij}(e)/cap(e) \leq p_e(i,j) \\
4. & \quad \forall\ nodes\ i, \forall\ arcs\ a = (j,k): \\
 & \qquad \pi_e(edge\text{-}of(a)) + p_e(i,j) - p_e(i,k) \geq 0 \\
5. & \quad \forall\ links\ h \in E : \pi_e(f) \geq 0 \\
6. & \quad \forall\ nodes\ i : p_e(i,i) = 0 \\
7. & \quad \forall\ nodes\ i,j : p_e(i,j) \geq 0
\end{array} \tag{3.9}$$

This LP has $O(ln^2)$ variables and $O(nl^2)$ constraints. The set of variables are the routing variables $f_{ij}(e)$, the minimization variable r, the link weights $\pi_e(h)$ and the variables $p_e(i,j)$, defined as the length of the shortest path from i to j according to the link weights. The variables $\pi_e(h)$ and $p_e(i,j)$ belong to the *dual LP problem* from the *slave* LP.

The first constraint makes reference to the set of "*routing constraints*", presented in Section 3.4.2. In this case, the constraints will be evaluated over arcs and not over links. The translation will be possible by introducing the equation for the sum of flows on both directions of a link to this set of constraints.

Constraints 2, 3 and 4 are the result from the dual problem from the slave problem. The calculation of the dual problem is out of the scope of this MSc thesis, but we encourage the reader to consult Sections 6c and 6d from [9]. This allows the authors to replace the exponential number of constraints (for all possible paths) from the previous approach with a small polynomial number of constraints. Constraints 5, 6 and 7 are bounding constraints for the variables of the problem.

As we already mentioned in the introduction of this chapter, we used CPLEX [1] to solve the LP formulations. In general, other LP solvers can be used but it is important to be sure that the solver includes the *Interior-Point algorithm*, cause Applegate and Cohen specified this algorithm to solve their LP problem.

The main disadvantage of LP problem is the number of variables that the problem generates: $O(l \cdot n^2 + l^2)$. As the networks we use have more nodes and links, the number of variables will grow really fast and, hence, we require a computer with high memory capacity to allocate such a problem.

3.5 Two-phase algorithm

3.5.1 Introduction

This algorithm represents a totally different approach from the previous algorithms presented in this chapter. While in Section 3.3 and Section 3.4 the traffic from source to destination is carried along “direct” paths, here we present a routing through a set of *intermediate nodes* (also called *two-phase routing*). The origin of this approach can be traced back to Valiant’s Load Balancing [30], where Valiant proposed a randomized scheme for processor interconnection networks. However, Valiant’s work was only valid for hypercube networks.

In 2004 [17], *two phase routing* was proposed for general networks as an oblivious routing scheme for handling traffic variations subject to aggregate ingress-egress constraints. We will work with one of the most recent implementation of two-phase routing, included in the paper presented by Kodialam, Laksham, Orlin and Sengupta [16]. In the following sections we will present the theory of such routing, an implementation method and the LP formulations to calculate the routing.

3.5.2 Traffic variation model and Two- phase routing theory

Traffic variation model

In this case, although the specific current traffic load on the network is unknown like in previous approaches, they use a constraint traffic model: the Hose model [13]. This model bounds the total amount of traffic that enters (leaves) a node in the network by the total capacity of all external egress (ingress) links at that node.

We will identify the upper bounds on the total amount of traffic entering and leaving the network at node i by R_i and C_i, respectively. The allowable traffic matrices $D = [d_{ij}]$ for the network are constrained by these ingress-egress link capacity bounds, so:

$$\sum_{j \in N, j \neq i} d_{ij} \leq R_i \quad \forall i \in N \qquad \sum_{i \in N, i \neq j} d_{ij} \leq C_i \quad \forall i \in N$$

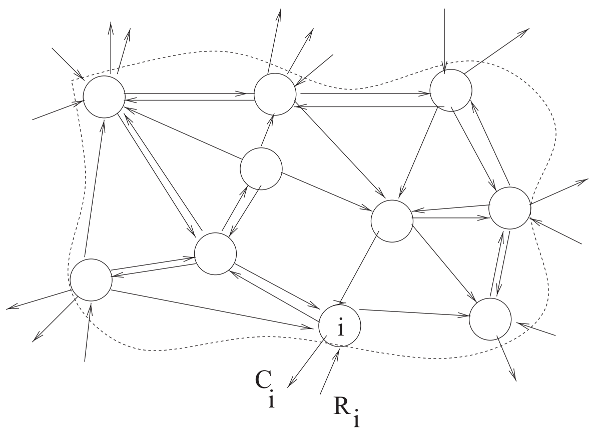


Figure 3.8: Ingress-egress constraints from Hose's model [16].

Therefore, for given R_i and C_i, we can denote the following set of traffic matrices as feasible candidates to be routed within the network:

$$D(\overrightarrow{R},\overrightarrow{C}) = \{[d_{ij}] \,/ \sum_{j\in N, j\neq i} d_{ji} \leq R_i \quad and \quad \sum_{i\in N, i\neq j} d_{ij} \leq C_i \quad \forall i \in N\} \tag{3.10}$$

The authors use $\lambda \cdot D(\overrightarrow{R},\overrightarrow{C})$ to indicate the set of all the traffic matrices in $D(\overrightarrow{R},\overrightarrow{C})$ whose entries are multiplied by λ. The Hose model defines the throughput as the maximum multiplier λ such that all the matrices in $\lambda \cdot D(\overrightarrow{R},\overrightarrow{C})$ can be feasibly routed under given link capacities (by some routing scheme).

Two-phase routing

Two-phase routing has been proposed [31], [16] as an oblivious routing scheme for general networks to handle arbitrary traffic variations subject to aggregate ingress-egress constraints (i.e., the Hose model). As the name indicates, the routing scheme operates in two phases:

- **Phase 1:** Predetermined fraction α_j of the traffic entering the network at any node is distributed to every node j independent of the final destination of the traffic.

- **Phase 2:** Each node has to route the traffic received during the first phase for a different destination than itself to that respective destination.

Note that the traffic split ratios $\alpha_1, \alpha_2, ..., \alpha_n$ in Phase 1 are such that $\sum_{i=1}^{n} \alpha_i = 1$. In [31] the split ratios were constrained to be equal while the authors from [16] considered a general scheme with possibly unequal split ratios.

3.5.3 Implementation

A simple method of implementing this routing is creating fixed bandwidth paths between the nodes. Basically, we can set up tunnels between the respective nodes, differentiating the ones used in Phase 1 and the ones used in Phase 2. The critical reason why this scheme works is because the bandwidth required for those tunnels depends on the ingress-egress capacities R_i, C_i and on the traffic split ratios α_j but never on the (unknown) individual entries of the traffic matrix.

For a better understanding, we will now calculate the bandwidth requirements that we will have to satisfy for Phase 1 and Phase 2 tunnels.

- Phase 1: Let us consider a node i with max incoming traffic R_i. Node i sends an amount of $\alpha_i \cdot R_i$ from this traffic to each intermediate node $j \in N$. At the end of Phase 1, the node has received $\alpha_i \cdot R_k$ traffic from another node k.

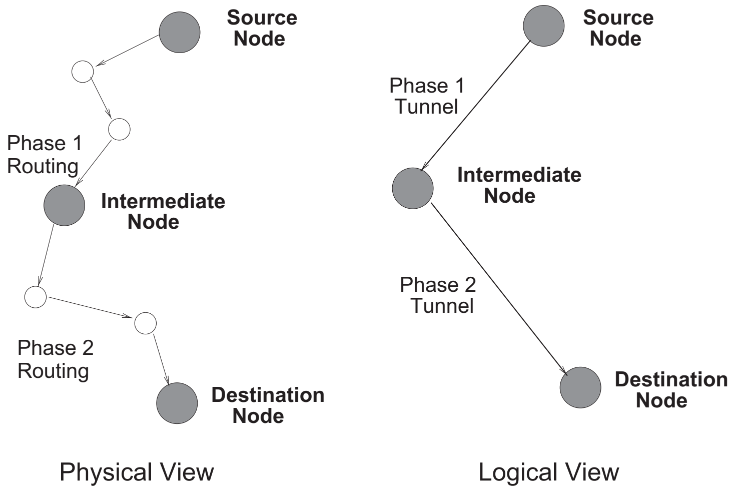


Figure 3.9: Two-phase routing [16].

- Phase 2: We can compute the traffic at node i from node k and destination j as $\alpha_i \cdot d_{kj}$, where d_{kj} is the demand from node k to node j. Then, we can consider that the summation of all the traffic at node i and destination to j at the beginning of Phase 2 is $\sum_{k \in N} \alpha_i \cdot d_{kj} \leq \alpha_i \cdot C_j$. Therefore, the traffic demand from node i to node j, as a result, of Phase 2 is at most $\alpha_i \cdot C_j$.

In conclusion, traffic demands in Phase 1 and Phase 2 result in a maximum total demand from node i to node j of $\alpha_i \cdot R_i + \alpha_i \cdot C_j$. In other words, in order to implement the two-phase routing, we only need to effectively route the fixed matrix $D = [d_{ij}] = [\alpha_i \cdot R_i + \alpha_i \cdot C_j]$ which does not depend on any specific traffic matrix $D \in D(\overrightarrow{R}, \overrightarrow{C})$.

3.5.4 LP formulations

From section 3.5.1 we know that the only assumptions about the traffic are the limits imposed by the ingress-egress constraints at each node. From section 3.5.2 we know that we need to effectively route the fixed matrix $D = [d_{ij}] = [\alpha_i \cdot R_i + \alpha_i \cdot C_j]$. Now, we formulate the linear programming problem to solve these two issues and choose a routing scheme that maximizes the throughput λ (calculated as $\lambda = \sum_{i \in N} \alpha_i$). At the end, the resulting split ratios α_i will be divided by λ to normalize them (in order to accomplish that $\sum_{i=1}^{n} \alpha_i = 1$).

$$
\begin{array}{rlll}
\textbf{objective} & max\ \sum_{i \in N} \alpha_i & & \\
\textbf{constraints} & & & \\
& \forall i, j, k \in N & & \\
1. & & \sum_{e \in OUT(k)} g_e^{ij} - \sum_{e \in IN(k)} g_e^{ij} = \alpha_j \cdot R_i + \alpha_i \cdot C_j & \mathit{If\ k{=}i} \\
& & \sum_{e \in OUT(k)} g_e^{ij} - \sum_{e \in IN(k)} g_e^{ij} = -\alpha_j \cdot R_i - \alpha_i \cdot C_j & \mathit{If\ k{=}j} \\
& & \sum_{e \in OUT(k)} g_e^{ij} - \sum_{e \in IN(k)} g_e^{ij} = 0 & \mathit{other} \\
2. & \forall e \in E & \sum g_e^{ij} \leq cap(e) & \\
3. & \forall i \in N & \alpha_i \geq 0 & \\
4. & \forall e \in E, \forall i, j \in N & g_e^{ij} \geq 0 &
\end{array}
\tag{3.11}
$$

There are two sets of decision variables: the fraction of traffic (split ratio) that will be routed to node i in the first phase denoted by α_i and the flow value from source i to destination j over *link e* denoted by g_e^{ij}.

Constraint 1 corresponds to the *multi-commodity flow* problem to route a flow of $\alpha_i \cdot R_i + \alpha_i \cdot C_j$ from source i to destination j. Constraint 2 represents the link capacity limitations of the network.

There are three aspects of this formulation, known as "*link flow based formulation*", that need to be taken into consideration:

- This LP formulation does not minimize the congestion on the network; hence, no *competitive ratio* is guaranteed like in Section 3.3 and Section 3.4. Instead, what the authors guarantee is a *throughput* that the network will be able to offer.
- This formulation generates a high number of variables: $O(l \cdot n^2)$, which will require a computer with high memory capacity to be solved.
- The authors do not specified any concrete algorithm to solve this problem. Therefore, we decided to use the *simplex* method for LP problems.

3.6 Conclusions

In this chapter we have presented 3 different oblivious routing algorithms which represent 3 totally different approaches. While in Section 3.2, the method is based on decomposition techniques and random values, Section 3.3 and Section 3.4 are based on LP formulations. However, Section 3.3 is based on calculating direct paths between the respective source and destination, while Section 3.4 proposes routing the traffic through a set of intermediate nodes. These LP formulations introduce a high number of variables and, hence, we need a computer with high memory capacity to solve these LP problems.

Secondly, while the first two algorithms try to minimize the congestion at the network and give some bounds in the competitive ratio performance, the third algorithm just tries to maximize the throughput at the network and it is not able to guarantee any competitive ratio. However, while the Applegate and Cohen algorithm provides the best oblivious routing regarding to congestion, Räcke's algorithm just offers a congestion bound for general networks. Besides, every approach is based on a paper which presents its own performance results with their own parameters to compare the oblivious routing with other optimal routings (e.g. shortest path routing).

In conclusion, after a detailed study of the three approaches proposed, the challenge for the next chapter is to create a framework that enables us to perform tests on the three approaches under the same conditions to be able to compare them. To our knowledge, this is the first comparison between these algorithms.

Chapter 4

Simulations and results

4.1 Introduction

The main objective of this thesis is to develop a common framework to test the *demand oblivious routings* presented in chapter 3 under the same conditions. According to the knowledge of the author, this is the first time that these three algorithms are compared.

The three papers studied use different metrics in the performance analyses of their algorithms:

- In the paper "*Minimizing congestion in general networks*" [24], Räcke performs a *competitive analysis*[1] to support his results. The author presents an algorithm that guarantees a *competitive ratio* of $O(log^3 n)$ with respect to the congestion links.
- In "*Making Routing Robust to Changing Traffic demands: Algorithms and Evaluation*" [9], D. Applegate and E. Cohen define a new metric: the *oblivious ratio*. This ratio measures the performance of the algorithm relative to the best possible solution under all possible traffic matrices (TMs) for the chosen network.
- The two-phase routing aims to maximize the *throughput* at the network, factor also used as main metric of their evaluation. Murali Kodialam and et al. define in "*Oblivious routing of Highly Variable Traffic in Service Overlays and IP backbones*" [16] the *throughput* as the maximum multiplier λ such that all matrices in $\lambda \cdot \tau(\overrightarrow{R}, \overrightarrow{C})$ can be feasible routed. The matrix $\tau(\overrightarrow{R}, \overrightarrow{C})$ refers to the Hose Model [13] studied in section 3.5. Nevertheless, the *throughput* is the reciprocal metric from the *maximum link utilization* (MxLU) and, therefore, directly related to the link congestion.

Nevertheless, these three different analyses are based on theoretical calculations and, in some case, only on some simulations. One may ask, how would these algorithms perform in real life? How will they behave with different traffic matrices (TMs)? Are they as good in practice as in theory? And if I were a network administrator, which one would I choose? During the rest of this chapter we will try to give an answer to these and more questions about the algorithms studied.

[1]Competitive analysis has been explained in section 2.2

4.2 Framework

4.2.1 Objectives

The framework presented in this thesis has three main objectives:

- Analyzing the three algorithms presented in chapter 3 under the same conditions.
- Extracting as much information as possible from different tests in order to increase the knowledge about the performance of these algorithms and being able to make recommendations.
- Presenting realistic scenarios, according to topologies and traffic demands or traffic matrices (TMs).

4.2.2 Tests: metrics and reference routing algorithms

In Chapter 2 we already studied the three main factors that a network administrator should consider when choosing a routing: *load balancing* and *path length* (which will be reflected in the *performance of the routing*) and the *resiliency* (or, in other words, the *robustness of the routing* against failures in the network).

In the following subsections, we will present the metrics used in our tests and we will talk briefly about the *Open Shortest Paths First* (OSPF) algorithm, included in the analysis to compare how the studied algorithms perform with respect to algorithms used in daily life. All the formulation we present can be used in the case of *directed networks* just translating the metrics used from *links* to *arcs*.

Performance metrics

Our analysis comprises *theoretical* and *practical metrics*. *Theoretical metrics* can give a really good overview on the general performance of the routing, because they are not linked to any specific case. However, they are based on worst case scenarios and never on specific *traffic matrices* (TMs). Hence, they do not provide realistic pictures of the routing performance in real situations. *Practical metrics* can focus in specific situations, because they are linked to TMs, but their reliability is associated with the reliability of the TMs. Including both kind of metrics in our framework, we can cover more possible scenarios and we will be able to extract reliable conclusions.

- Theoretical metrics

  1. <u>Oblivious ratio</u>: The *oblivious ratio* measures in the *worst-case scenario* for a routing, how far it is from the optimal routing calculated for that situation. In other words, it tells us that for any possible situation, the routing performance will be in a range of [1, oblivious ratio] times the optimal routing performance.

     We choose to add this metric to our framework because it provides us with an idea of how the routing performs in general. However, it does not show how the routing behaves inside the interval [1, oblivious ratio]. This means that routings with worse oblivious ratio can perform better in real situations with real traffic matrices (TMs) than routings with a lower oblivious ratio.

The oblivious ratio is defined as the *performance ratio* (check Equation 4.2) of a routing with respect to all possible TMs (**D**).

$$PERF(f,D) = max_{D \in \mathbf{D}} PERF(f,D) \tag{4.1}$$

To calculate the *oblivious ratio* of a routing, we introduce the *arc flows* to the linear programming problem (LP) from Section 3.4.3 and solved the problem for this fixed values.

Finally, calculating the *oblivious ratio* is reciprocal to calculating the highest possible *throughput* in the network because the throughput is reciprocal to the *maximum link utilization* (MxLU). In other words, the routing with the lowest oblivious ratio experiences the highest possible throughput in the network. Therefore, there is no need to introduce special metrics for the two-phase routing algorithm case.

2. Quality of the routing: In [20] the author defines the *quality of the routing* based on two factors: the *path dispersion* and the *path variation*. The first one is concerned with how many paths exist between each Source-Destination (SD) pair as specified by the routing. The second one refers to how much the paths between each SD pair differ from the shortest path of the SD pair and how much they differ from each other.

   At the end, these parameters are translated into the *number of flows* allocated at every link and the *average percentage of traffic* that these flows carry.

- Practical metrics

  Performance ratio: For a specific TM, the *performance ratio* evaluates the maximum link utilization (MxLU) of the routing f divided by the minimum possible link utilization given a TM D.

$$PERF(f,D) = \frac{MxLU(f,D)}{OPTU(D)} \tag{4.2}$$

  The MxLU for a routing f given a TM D is defined as the maximum of the total flow on a link divided by the capacity of that link, over all the links in the network.

$$MxLU(f,D) = max_{(i,j) \in links} \quad \frac{\sum_{a,b} d_{a,b} \cdot f_{a,b}(i,j)}{cap_{ij}} \tag{4.3}$$

  Finally, an optimal routing for a certain TM D is a routing which minimizes the maximum link utilization. This routing is computed by solving the LP formulations corresponding to the multi-commodity flow problem for the specific TM. Therefore, we can define the optimal utilization (OPTU) for a routing f give a TM D as the minimum MxLU possible.

$$OPTU(D) = min_{f|f\_is\_a\_routing} \quad MxLU(f,D) \tag{4.4}$$

Robustness metrics

First of all, we define *routing robustness* as the routing response to perturbations or changes in the network. In our case, these perturbations will be characterized by a number of links removed from the network due to random unintentional failures.

The routing robustness will be measured in terms of *availability*. We define *availability of the routing* as the number of Source-Destination pairs (SD pairs) over the total number of possible SD pairs in the network which are still connected by a remaining complete path. In other words, how many SD pairs the routing still can connect after the removal of a number of links.

Although availability is simple to measure in most situations, in our case there is a situation we need to face: the routing algorithms provide flows[2] and not paths[3] as results. The problem appears with Applegate and Cohen's algorithm, from which flow results we cannot extract the paths that the traffic will follow because there are cases where multiple sets of paths give the same flow results. However, we can easily calculate the flows generated in a network from a set of paths.

In this framework, we have decided to solve the mentioned situation calculating the availability over the flows for all the routings: when a number of links fails, we check whether the remaining links where is allocated traffic from i to j (so $f_{i,j}(e) \neq 0$) we can construct a complete connected path or not. Although, we lose capacity on measuring robustness for concrete situations, the suggested framework provides a fair comparison between all the routing analyzed because they will be treated in the same way.

We need to measure this availability for different situations: first for all the scenarios where one link is removed from the network, then for all the scenarios where two links are removed from the network and so on until all the links are removed.

In [22] Van Mieghem and Doerr et al. presented a definition and a framework to compute topological network robustness. They interpret network robustness as a compatible measure for network response to perturbations or challenges (such as failures or external attacks) imposed on the network. Here, we present an adaptation of this framework to compute the robustness of routing algorithm solutions[4].

The framework consists of measuring a proposed $R-value$ parameter, which is function of a routing r to analyze. The network will be expressed by a graph G, defined by a set N of n nodes and a set L of l links.

[2]The implemented routing algorithms provide a list of variables $f_{i,j}(e)$ (check section 3.2) which reflect the percentage of traffic between i and j allocated in link e.

[3]We understand by *path* the route that a package follows to go from a source do a destination.

[4] This work has been developed jointly with Ir. Javier Martin Hernandez, Phd. candidate at Network and Services researching group, Faculty EEMS, TUDelft

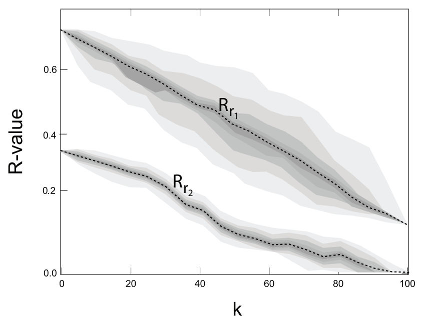


Figure 4.1: Example where r_1 is considered more robust routing than r_2

We will measure this R-value along a set of perturbations. We identify the perturbations or challenges P as the sets of $|P_k|$ links removed from the network G. Every perturbation $|P_k|$ represents a state of the network and it has an associated R-value. Therefore, any realization can also be expressed as a sequence of R-values, denoted $R[k]_{0 \leq k \leq n}$, where $R[0]$ represents the absence of perturbations. Then, we plot this sequence of R-values obtaining an contour plot along the vector of perturbations.

The $R-value$ represents the *availability* of the routing r as links are randomly removed from the graph. This $R-value$ will help us to compare the *robustness* of two or more routings: if r_1 and r_2 are routings solutions for a network G and $R_{r_1} > R_{r_2}$, then r_1 is considered more robust than r_2.

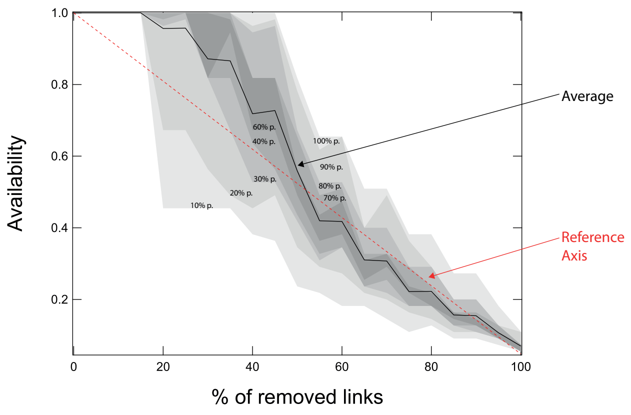


Figure 4.2: Detailed explanation of the contour plot

As we already mentioned, there are multiple possible scenarios for every perturbation $|P_k|$. The number of these possible scenarios grows exponentially with the number of links the network consist of and, hence, we need to fix a number of iterations at every P (1000 iterations per P in our case). In the end, we have a set of R-values for every perturbation and a *contour plot* representing the function $R-value$, as illustrated in figure 4.2.

Figure 4.2 shows the robustness of an example routing for a network. The x axis represents the percentage of links removed from the network and, the y axis determines the

Räcke	Racke
Applegate and Cohen	AC
Two phases	Two_P
Shortest Paths (Cost = Inv. Capacity)	SP_W
Shortest Paths (Cost = Inv. Capacity + Distance)	SP_L

Table 4.1: Algorithms and acronyms.

robustness measured in means of availability. The *contour plot* comprises different colored regions representing different percentiles. The continuous line represents the mean or average of the availability values in the contour plot. We also provide a red dotted line as reference axis, which illustrates a linear function between the availability and the percentage of links removed (the availability decreases linearly to the number of links removed).

Algorithms

As we already mentioned, the framework presented will evaluate the performance and robustness of the three algorithms presented in chapter 3. In our analysis we will also introduce the *Open Shortest Paths First* (OSPF) algorithm to have an idea if oblivious routing algorithms can perform better than algorithms already used in daily life or not.

Nevertheless, OSPF algorithms provide shortest paths for every SD pair based on a given cost associated to the links of the network. This cost is fixed by the network administrator and, hence, it is unknown for us. Therefore, we will work with two different approaches:

- Cost based on capacity: based on the recommendations made by Cisco, we calculate the weights of the links as the inverse of their capacities.

$$w_l = \frac{10^6}{cap(l)} \tag{4.5}$$

- Cost based on capacity + length of the paths: we extend the previous cost by adding the length of the paths (in km) to the equation, so delays are also take into account.

$$w_l = \frac{10^6}{cap(l)} + length(l) \tag{4.6}$$

In the following table we present the algorithms that will be part of our analysis next to the acronyms we will use in the rest of this chapter.

4.2.3 Scenarios

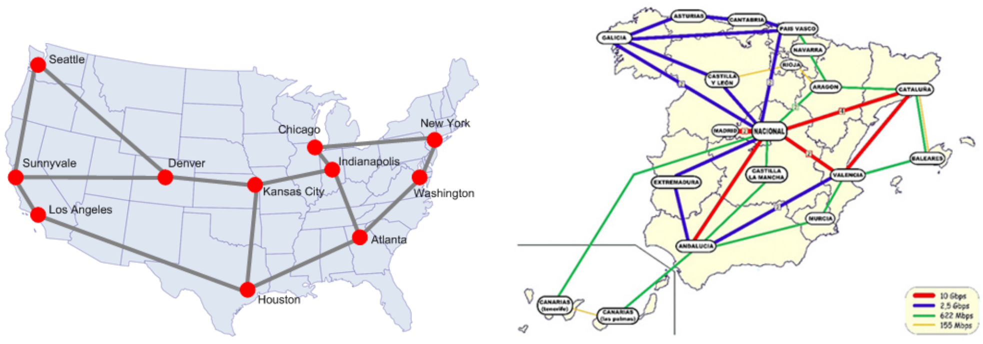
Topologies

Two of the papers studied in Chapter 3 make use of topologies from the project Rocketfuel [29] in their tests. Although these topologies belong to real networks, the authors modify them in ways that are not explained with enough detail (i.e., approaching Point of Presence, reducing links, using old topologies, ...). This fact creates two problems: the impossibility of directly relating results presented in the different papers and enormous difficulties in reproducing and validating[5] the experiments from the paper.

We have chosen four real research networks for our analysis: Abilene network [3] (America), Rediris network [6] (Spain), RediMadrid network [5] (Madrid) and Surfnet network [7] (The Netherlands). The main reasons for this choice are:

1. They are research networks and, in most of the cases, the state owns the network. This point is important because in their respective webpages one can find much more information about the network (specially detailed topology maps) than in the case of private networks.

2. They represent a good combination of *mesh* and *ring* networks, the two mostly used topologies over Internet nowadays.

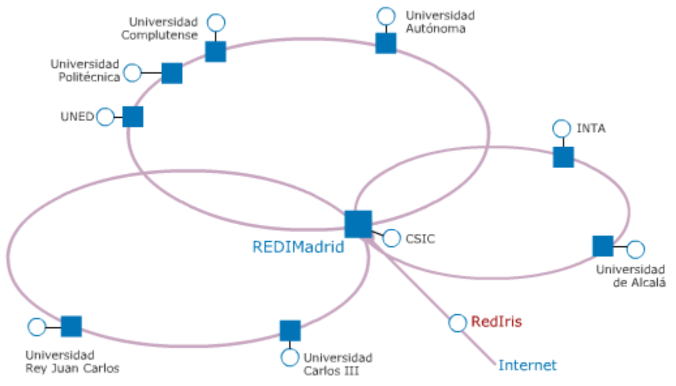


(a)*Abilene* (b)*Rediris*

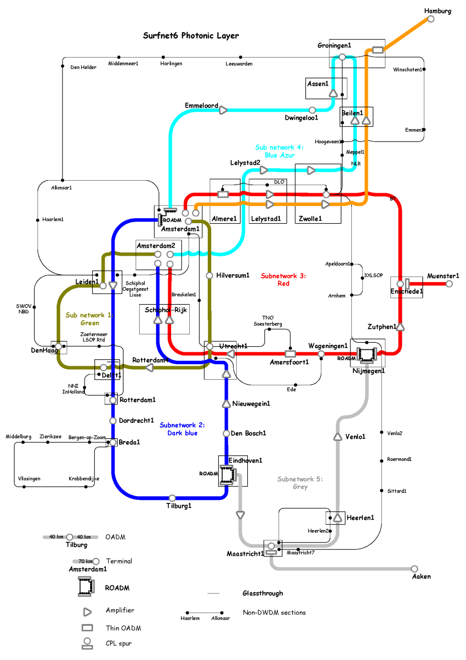
Figure 4.3: Mesh networks.

Table 4.2 presents the most important characteristics of these networks for our analysis. We fix the capacities of all the links in the four networks to *10gbps*, which means that we ignore the different link capacities in the case of RediMadrid. The main reason to act this way are the limitations of our Traffic Matrices model (explained in the next section). The model we use assign traffic randomly between SD pairs, not taking into account that nodes connected to links with less capacity should receive less traffic. Therefore, as future work, it would be interesting to extend the analysis to traffic models that take into account this

[5]In our specific case, the validation of the implemented algorithm is not that important in the case of two-phases routing and Applegate and Cohens algorithm because they ar based in linear programming formulations



$(a) RediMadrid$



$(b) Surfnet$

Figure 4.4: Ring Networks.

	Abilene	Rediris	RediMadrid	Surfnet
number of nodes	11	17	9	20
number of links	14	29	11	23
min. node degree	2	2	2	2
max. node degree	3	10	6	8
av. node degree	2.54	3.35	2.4	2.3
topology	mesh	mesh	ring	ring
network dimesion	national	national	regional	national

Table 4.2: Research networks parameters.

particularities of this kind of networks randomly and study if this situation produces any change in the behavior of the analyzed routings.

Finally, in the case of two-phase routing, it is necessary to know the incoming and outcoming capacities at every node of the network (R_i and C_i respectively, section 3.5.2). We consider that the nodes from the networks are connected to outside nodes by links with the same capacities from the ones in the network. Therefore, the values from R_i and C_i will be *10gbps* in every case.

Traffic matrices

Different techniques have been developed to estimate traffic matrices (TMs) [21]. In this thesis, we use the *bimodal family* of synthetic TMs, where the traffic demands are generated by a mixture of two Gaussians. For each Source-Destination pair, a biased coin is flipped deciding which of the two Gaussians distributions will give value to the flow.

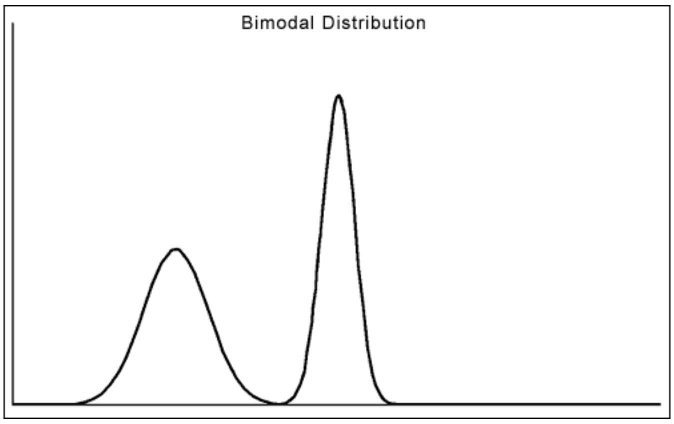


Figure 4.5: Bimodal distribution.

The model allow us to simulate the traffic behavior in network backbones studied and described in [11], where a small number of flows generate a large fraction of the total traffic, while a large number of flows generate a small fraction of it. This phenomenon is known as "*elephants and mice phenomenon*" and the authors demonstrated that it is a stable pattern in the network throughout the day.

To simulate this phenomenon, we will make use of the bimodal distribution. Based on this distribution we generate six sets of TMs in our analysis in each scenario (TM1, TM2, TM3, TM4, TM5 and TM6), each one of them contains 1000 matrices. In the first set (TM1), the average of 20% of the flows will be three times the average of the rest. In the second set (TM2), the average of 20% of the flows will be four times the average of the rest, and so on.

This model of TMs has already been used by Applegate and Cohen in their paper [9], which also included the gravity model in their test. However, we discard using this second model because the inference of traffic flows without real data does not give reliable results. The other two papers do not present any test where TMs are used.

4.2.4 Traffic engineering toolbox TOTEM

To perform the test and evaluate the results, we use the open source traffic engineering toolbox TOTEM [19],[8]. The toolbox TOTEM is designed and developed by the E-NEXT[6]

[6]E-Next was a Network of Excellence (NoE) funded by the European Commission under the 6th IST program

partners. The functionality of TOTEM needs to be extended to be able to introduce the routing results from the oblivious routing algorithms. Besides, TOTEM already incorporates implementations of the multi-commodity flow routing that will be useful to develop more complete comparisons and evaluations. Totem also provides different methods for traffic matrices generation which include the bimodal family.

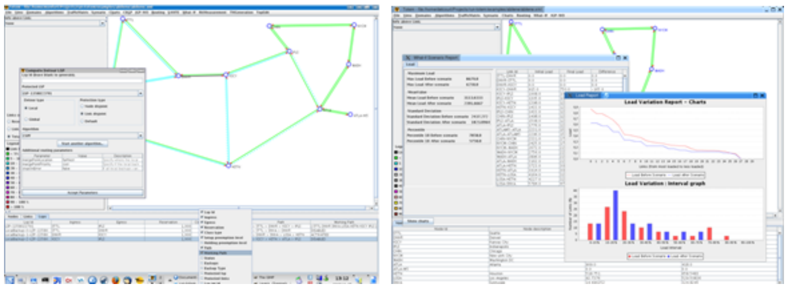


Figure 4.6: Totem snapshots.

However, the toolbox does not adapt exactly to our needs and we need to modify aspects of the algorithms results:

1. TOTEM works with *directed networks*, while two of the studied algorithms work with *undirected networks*:

    - In the case of Räcke's algorithm, the results can be directly used for directed networks, but the competitive ratio of $O(log^3 n)$ is not guaranteed anymore. However, the limit is already not guaranteed because we are working with an approximation of the algorithm.
    - In the case of Applegate and Cohen's algorithm, the authors introduced a lemma [9] (lemma 5.4) to derive directed networks from undirected networks without affecting the optimal oblivious ratio. We only need to replace each edge e by two anti-parallel arcs that have the same capacity as e. However, one of the conditions is to apply symmetric TMs, which are not really common in the real world. Therefore, we can use the results obtained from the LP formulation, but the oblivious ratio promised by Applegate and Cohen cannot be guaranteed anymore.

2. TOTEM also includes a complete implementation of the OSPF (Open Shortest Paths First) routing. However, when the OSPF routing is performed TOTEM activates some measure against the congestion over which we have no control (e.g. choosing more than one shortest path if possible,...). Therefore, we implement our own version of the shortest paths algorithm, based on the Dijkstra's algorithm [23].

One may ask why we chose TOTEM. First of all, it is much easier to use than other simulators, like ns [4]. It is really handy for network administrators who want to test their network in some situations. Secondly, TOTEM already includes many traffic and topology generation tools, routing mechanisms and a powerful engine to process results. And last but not least, TOTEM is an open source traffic engineering toolbox created by the research community for the research community.

The main drawback is that evaluations are restricted to the TM introduced. This method is not as flexible as other simulators, like ns, where you have the control to modify the traffic from any SD pair at any moment without introducing a new TM.

Finally, TOTEM provides a command-line mode where the tool executes the actions included in an xml format file called "*scenario*". This mechanism allow us to automatize the tests. In [8] the reader can find manuals on how to use TOTEM and to develop new tools for it.

4.3 Performance results

We start with the performance analysis based on the theoretical metric mentioned: *oblivious ratio* and *quality of the routing*. During this section we will use the acronyms presented in Table 4.1. In the case of the network RediMadrid, we will not analyze the *SP_L* routing because is a regional network and the link length is not long enough to be taken into consideration. In Table 4.3 we present the *oblivious ratios* of the routings in the different networks.

As expected, *AC* has the lowest oblivious ratio in all the cases. This result was easy to predict, because the objective of the LP formulation from Applegate and Cohen's algorithm is to minimize this value. However, having the lowest *oblivious ratio* does not guarantee that the routing is going to perform the best in every situation.

The rest of the algorithms cannot compete with *AC*, although SP_W and SP_L oblivious ratios are not that far in the case of ring network. Actually, all the oblivious ratios from the algorithms are lower in ring networks that in mesh networks. The reason behind this behavior is than the path diverse in ring networks is lower than in mesh networks, and, therefore, the routing choices are lower.

Table 4.4 evaluates the quality of the routings in the different networks. The first conclusion we can extract is that *AC* and *Two_P* allocate many more flows at the links than the rest of the routings. This situation is a consequence of the multipath nature of these two algorithms. *Racke* behaves similarly to the shortest paths algorithms, which is expectable because both of them provide single paths for every SD pair.

There is a trade-off between the congestion in a network and spreading the traffic between more than one path. In most of the cases, spreading the traffic into a higher number of links helps to reduce the general congestion of the network. But there is a limit after which you are flooding the network and the congestion of some links can start being really high. This is a good concept to keep in mind during our next test.

Figure 4.7 illustrates our practical performance analysis. For each network, we present two graphs: the left one shows the behavior of the *performance ratio* of the routings for the different TMs, and the right one shows the *maximum link utilization* (MxLU). The first one allows us to compare the performance of the routing with the optimal routing for those specifics TMs, while the second one gives us an idea of how fast the network congests with each algorithm.Due to the random nature of the traffic matrixes used in our experiments, we

	Racke	AC	Two_P	SP_W	SP_L
Abilene	4	1.85057	8.66667	4	3
Rediris	7	1.9672	12.87	4	6
RediMadrid	4	1.6	4.9	2	X
Surfnet	3	1.71	8.5	2	2

Table 4.3: Oblivious ratio of the routings for the different networks

	Abilene (110 SD pairs)		RediMadrid (72 pairs)	
	Av n flows	Av % traffic per flow	Av n flows	Av % traffic per flow
Racke	10.25	100.00	7.65	100.00
AC	43.78	30.50	20.68	40.50
Two_P	37.6	37.50	37.86	48.7
SP_W	9.5	100.00	6.6	100.00
SP_L	9.7	100.00	X	X

	Rediris (272 SD pairs)		Surfnet (380 SD pairs)	
	Av n flows	Av % traffic per flow	Av n flows	Av % traffic per flow
Racke	10.59	100.00	20.82	100.00
AC	81.32	21.67	87.3	38.09
Two_P	61.46	43.67	152.43	40.11
SP_W	10.28	100.00	25.48	100.00
SP_L	11.17	100.00	25.57	100.00

Table 4.4: Quality of the routings.

perform 1000 iterations for each TM set (1 iteration per traffic matrix at each TM set).[7]

From a general view, we can extract the following conclusions:

- The routing *AC* is the *oblivious routing* algorithm from the ones studied in Chapter 3, that gives the best performance in all the cases. In most of the cases, its performance can beat and even win the performance from *SP_W*.
- Surprisingly, the approximation of Räcke's algorithm, being a randomized algorithm, does not perform that far from the optimal routing. Its results are quite promising, specially in the mesh networks we analyzed where its performance ratio is at most 1.6 for the traffic model we have work with.
- The routing *Two_P* gives the worst results from the three *oblivious routing* algorithms studied.
- The *SP_W* gives performances similar to *AC*. We have not studied this algorithm as a *demand oblivious routing* algorithm, but it also does not take any traffic demands into account when picking the paths.

[7]The statistical results (mean and standard deviations) from this experiment are shown in appendix A.

Now we will study the performance analysis for each network:

- Abilene: In this case, shortest paths seems to be the best option. However, *AC* does not perform that far from it. If we take a look at the MxLU, both algorithms are quite similar in most of the cases. It is interesting to point out that in this case, *Racke* performance is not that far from *AC* and it is even better than *SP_L*.

- Rediris: It is remarkable that the behavior of most of the routing stays stable along the different TMs. Only *SP_C* breaks this tendency. Rediris is a bigger and more dense network than Abilene. The degree of the nodes is higher and, in this specific case, *AC* performs much better than *SP_C* in most of the cases. It would be interesting to use similar networks and different TMs generation methods in future analyses to see if this behavior remains.

- RediMadrid: The behaviors of all the routings seem to be quite correlated. A possible explanation for this effect is that in a ring network the possibilities of choosing different paths which contains different links are lower than in mesh networks. Once more, *AC* and *SP_C* perform the best. RediMadrid is a regional network, quite small in comparison with the rest of the networks, so we do not perform the *SP_L* routing in this case.

- Surfnet: Like in the case of RediMadrid, the behavior of the routings seems also to be correlated. The network is bigger than RediMadrid and all the routings give better performances in this case.

4.4 Robustness results

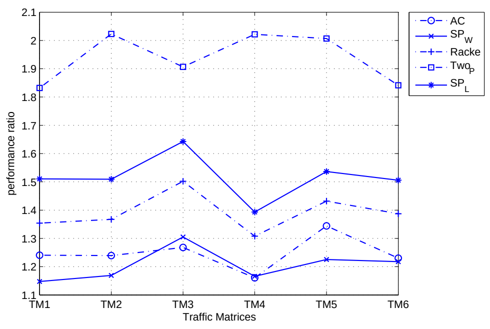
We will only perform a robustness analysis of the routing in two of the four networks: Abilene and RediMadrid. Our objective is to establish a comparison of the robustness from the routings and study if the topology affects to its behavior.

Figure 4.8 presents the *availability* of the different routings according to the percentage of links removed from the networks. The first conclusion we can extract is that two-phase routing and AC routing are more robust that the rest because their averages are higher than the rest, which makes sense because both routings are multipath. More in detail, AC seems to behave more robust that two-phases because until the 20% of removed links, the mean stays at 1 (all SD pairs connected).

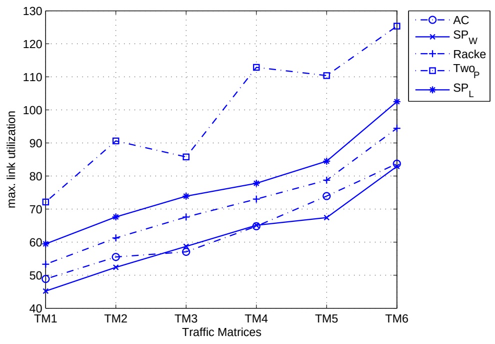
Another insightful variable to evaluate is the number of links that need to be disconnected for the availability of the routing to drop below 0.5. In the case of two-phases and AC it is necessary on average to disconnect at least the 50 percent of the links for the availability to go under 0.5, while in the rest of the cases it is only 40 percent.

Figure 4.9 represents the robustness study of routings for the ring network RediMadrid. The results are quite similar to the case of Abilene, where two-phase routing and AC routing are more robust than the rest again. However, it is interesting to point that the behavior from the two-phase and AC routing look more similar to each other in this network, as well Racke and Shortest paths do with each other. This fact makes sense because in ring networks the path diversity is lower and, therefore, the removal of links affects in an equal way to the availability of all of them. Besides, we can see how in this case, the averages of the contour plots tend to approach the reference linear function (dotted red line), where disconnecting a link means disconnecting a SD pair.

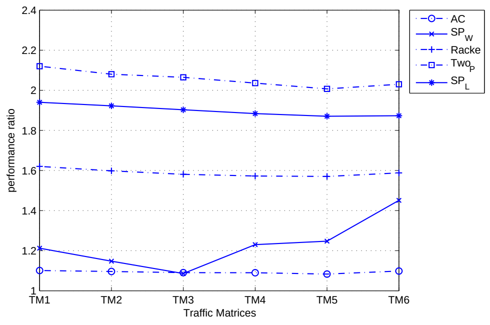
Again, it is necessary to remove at least 50% of the links in average so the availability of the multipath's routing goes under 0.5.



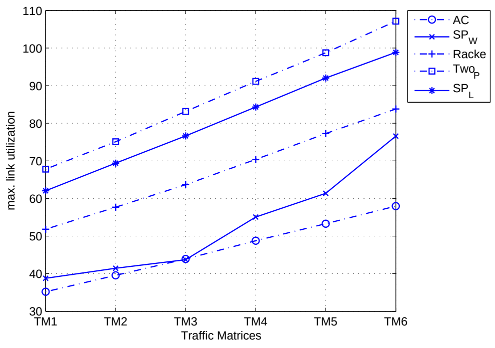
(a)*Abilene network: Performance Ratio*



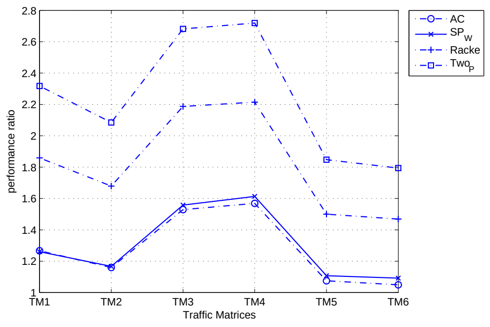
(b)*Abilene network:Max Link Utilization*



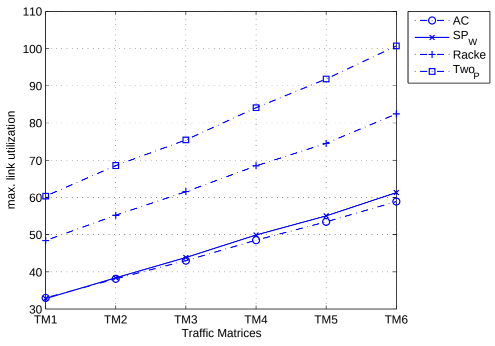
(c)*Rediris network: Performance Ratio*



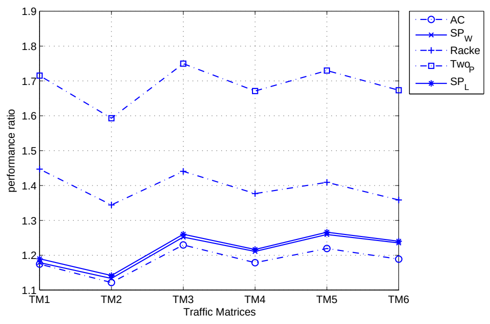
(d)*Rediris network:Max Link Utilization*



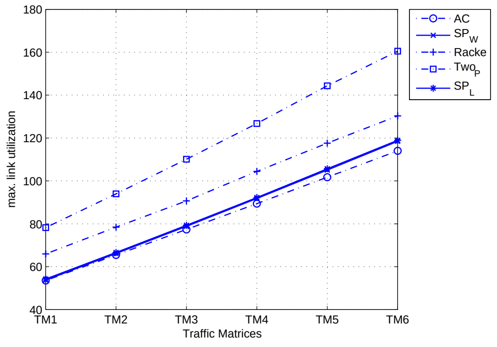
(e)*RediMadrid network: Performance Ratio*



(e)*RediMadrid network:Max Link Utilization*

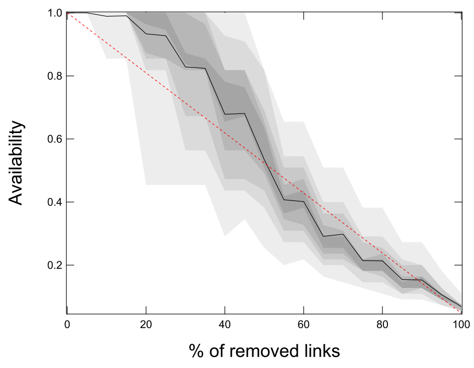


(f) *Surfnet network: Performance Ratio*

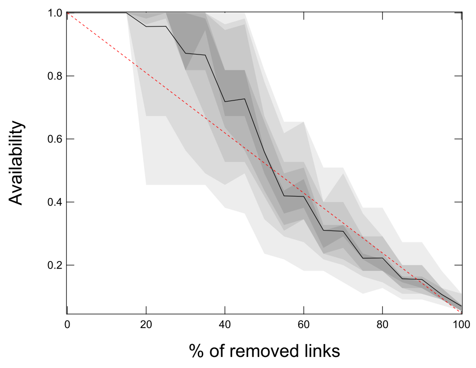


(g)*Surfnet network:Max Link Utilization*

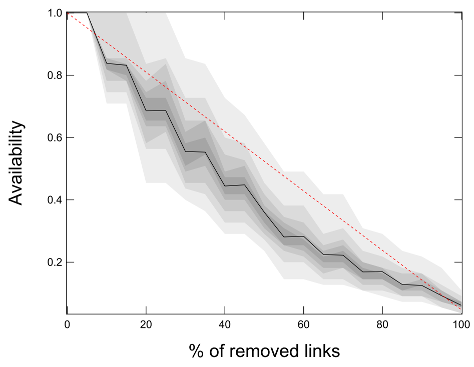
Figure 4.7: Performance analysis



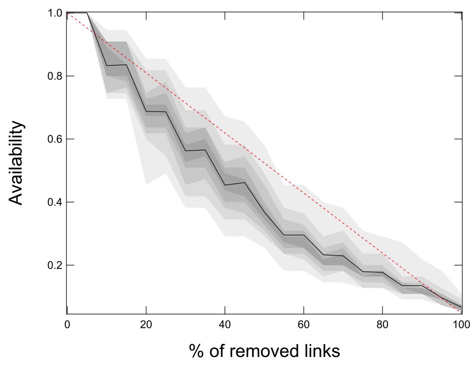
(a)*Two_P*



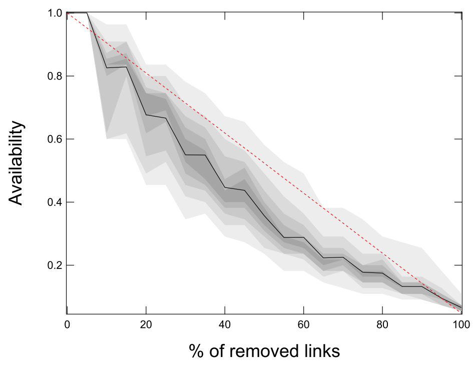
(b)*AC*



(c)*Racke*

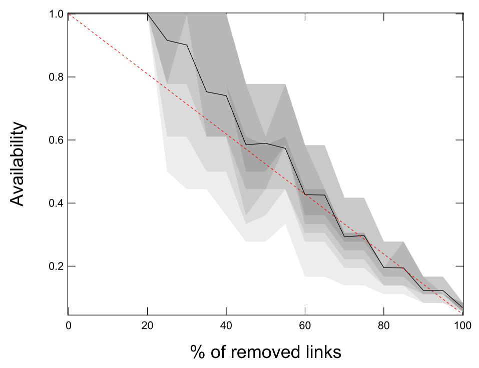


(d)*SP_W*

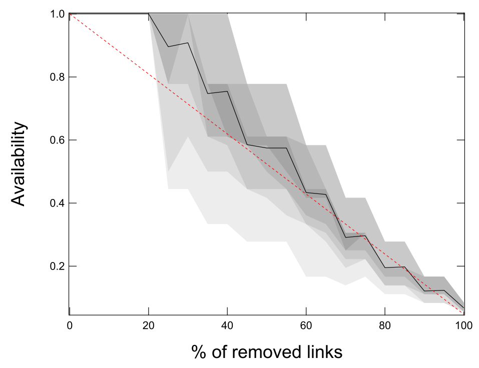


(e)*SP_L*

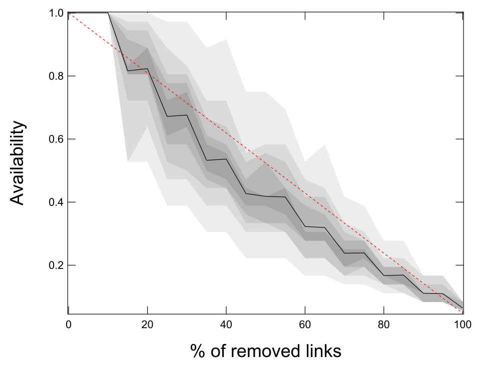
Figure 4.8: Robustness contour plots for Abilene's network



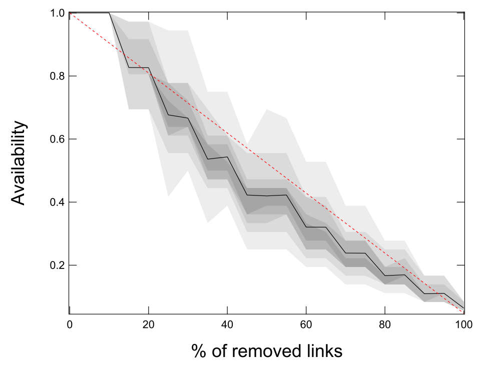
$(a)Two\_P$



$(b)AC$



$(c)Racke$



$(d)SP\_W$

Figure 4.9: Robustness contour plots for RediMadrid’s network

Chapter 5

Conclusions and Future Work

We have presented the first comparison, to the author's knowledge, between the most representative *oblvious routing* algorithms: Räcke's algorithm, Applegate and Cohen's algorithm, and two-phase routing algorithm. After a detailed theoretical study of each of algorithm, we have tested their performance and robustness and compared them to the Open Shortest Path First (OSPF) algorithm. Our general conclusion is that *oblivious routing* algorithms represent an interesting choice to take into consideration by network administrators when they have to select a routing method for their networks.

We conclude from our theoretical study that Applegate and Cohen's algorithm and two-phase routing are the easiest to implement, although the linear programming (LP) formulation from the first one required more time to be understand and introduce into the solver. However, both LP formulations generate a high number of variables, which represent a problem if we want to run this algorithms in a regular computer.

On the other hand, Räcke's algorithm is hard to implement and its techniques derive to NP-hard problems. Therefore, approximations to this algorithm are needed. In this thesis we have used the one developed by Hughes. The good side of the algorithm is that it does not present problems with memory consumption as the other two.

From our performance analysis we conclude that Applegate and Cohen's algorithm gives the best performance results in all the scenarios. It is interesting to remark that the OSPF performance can compete with Applegate and Cohen's algorithm, and it also does not take into consideration traffic when picking the paths.

Surprisingly, the approximation from Räcke's algorithm gives a performance not as far from the *optimal routing* as we would expect from a randomized algorithm. On the contrary, two-phase routing performs really bad. The main reason behind this bad performance is that routing paths are calculated from source to intermediate nodes and from intermediate nodes to destination nodes independently. Therefore, in many cases there are links repeated over the same path, so the traffic goes over the same link two times.

Regarding to robustness, Applegate and Cohen's algorithm and two-phase routing algorithm are the most robust ones because they provide multiple paths for every source-destination pair.

Finally, if we had to choose one of the three *oblivious routing* algorithms presented, we would choose Räcke's algorithm. The main reasons for this choice are that it does not present problems of memory (so it is able to run in regular computers), a simple approximation gives performance results not extremely far from the optimal routing, and there are plenty possibilities of improving Hughes' approximation (e.g. searching for a tree that looks like more like the one with equivalent communication characteristic of the original graph, make the algorithm choose more than one possible path for every SD pair, . . .).

5.1 Future work

We can classify the possible future work in three different paths:

1. Extending and improving the analysis framework: We suggest using real traffic data and synthetic traffic models different than the bimodal distribution, studying of the worst case scenario for every routing, and introducing the Quality of Service (QoS) to the analysis.

2. Developing a new approximation for Räcke's algorithm: The two main points would be developing a multipath routing version and looking for a more suitable routing tree generation technique than binary trees.

3. Studying possibilities of reducing the number of variables of the LP formulation from Applegate and Cohen's algorithm. If we could reduce this number, we could also think about introducing the Hose's model to the LP formulation from Applegate and Cohen.

Bibliography

[1] Cplex large-scale mathematical programming software. http://www.cplex.com.

[2] Ellipsoid method. http://www.utdallas.edu/ dzdu/cs7301/ellipsoid-1.pdf.

[3] The internet2 network. http://www.internet2.edu/network/.

[4] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[5] Redimadrid. http://www.redimadrid.es/.

[6] Rediris. http://www.rediris.es/.

[7] Surfnet. http://www.surfnet.nl/.

[8] Toolbox for traffic engineering methods. http://totem.run.montefiore.ulg.ac.be/.

[9] David Applegate and Edith Cohen. Making routing robust to changing traffic demands: algorithms and evaluation. *IEEE/ACM Trans. Netw.*, 14(6):1193–1206, December 2006.

[10] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. Optimal oblivious routing in polynomial time. pages 383–388, 2003.

[11] Supratik Bhattacharyya, Christophe Diot, and Nina Taft. Geographical and temporal characteristics of inter-pop flows: View from a single pop. *European Transactions on Telecommunications*, 13, 2002.

[12] Door William J. Dally and Brian Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann., 2003.

[13] N. G. Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, K. K. Ramakrishnan, and Jacobus E. van der Merive. A flexible model for resource management in virtual private networks. *SIGCOMM Comput. Commun. Rev.*, 29(4):95–108, August 1999.

[14] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. 2003.

[15] Bryan Hughes. *An Oblivious Routing Scheme for Parallel Computing in General Embedded Networks*. PhD thesis, Texas Technical University, 2010.

[16] Murali Kodialam, T. V. Lakshman, James B. Orlin, and Sudipta Sengupta. Oblivious routing of highly variable traffic in service overlays and ip backbones. *IEEE/ACM Trans. Netw.*, 17(2):459–472, April 2009.

[17] Murali Kodialam, T. V. Lakshman, and Sudipta Sengupta. Efficient and robust routing of highly variable traffic. 2004.

[18] Hang T. Lau. *A Java Library of Graph Algorithms and Optimization*. Chapman and Hall/CRC, 2006.

[19] G. Leduc, H. Abrahamsson, S. Balon, S. Bessler, M. D'Arienzo, O. Delcourt, J. Domingo-Pascual, S. Cerav-Erbas, I. Gojmerac, X. Masip, A. Pescapè, B. Quoitin, S. P. Romano, E. Salvadori, F. Skivée, H. T. Tran, S. Uhlig, and H. ímit. An open source traffic engineering toolbox. *Comput. Commun.*, 29(5):593–610, March 2006.

[20] Yuxi Li, Janelle Harms, and Robert Holte. A simple method for balancing network utilization and quality of routing. In *In Proceedings of ICCCN*, pages 71–76, 2005.

[21] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: existing techniques and new directions. *SIGCOMM Comput. Commun. Rev.*, 32(4):161–174, August 2002.

[22] P. Van Mieghem, C. Doerr, H.Wang, J. Martin Henandez, D.Hutchison, M. Karaliopoulos, and R.E.Kooij. A framework for computing topological network robustness. 2011. http://www.nas.its.tudelft.nl/people/Piet/papers/RobustnessRmodel_TUDreport20101218.pdf.

[23] Piet Van Mieghem. *Data Communications Networking*. Techne Press, 2006.

[24] Harald Räcke. Minimizing congestion in general networks. pages 43–52, 2002.

[25] Harald Räcke. *Data Management and Routing in General Networks*. PhD thesis, Universitt Paderborn, 2003.

[26] Harald Räcke. Survey on oblivious routing strategies. In *Proceedings of the 5th Conference on Computability in Europe: Mathematical Theory and Computational Practice*, CiE '09, pages 419–429, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. Experience in measuring backbone traffic variability: models, metrics, measurements and meaning. pages 91–92, 2002.

[28] Christian Scheideler. Lecture 3: Basic routing theory i. course: Theory of network communication. Course: Theory of Network Communication, 2002.

[29] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. In *In Proc. ACM SIGCOMM*, pages 133–145, 2002.

[30] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. pages 263–277, 1981.

[31] Rui Zhang-shen and Nick Mckeown. Designing a predictable internet backbone with valiant load-balancing. pages 178–192, 2005.

Appendix A

Appendix

In this appendix we present the statistical results from the performance test described in section 4.3.

Abilene						
	MCF		Two_P		AC	
	Mean	Stdev	Mean	Stdev	Mean	Stdev
TM1	39,372	4,0511	72,13	6,746	48,843	4,659
TM2	44,796	5,32	90,63	8,39	55,517	6,018
TM3	44,99	6,9277	85,78	10,719	57,055	7,877
TM4	55,83	7,96	112,87	12,767	64,8	9,09
TM5	55	9,4	110,38	14,49	73,93	10,944
TM6	68,07	10,835	125,36	15,686	83,73	12,37
	Racke		SP_W		SP_L	
	Mean	Stdev	Mean	Stdev	Mean	Stdev
TM1	53,306	6,7114	45,178	5,2576	59,476	7,1273
TM2	61,249	8,358	52,375	6,82	67,609	9
TM3	67,579	10,68	58,714	8,7075	73,898	11,9757
TM4	73,016	12,83	65,085	10,26	77,786	13,93
TM5	78,749	14,72	67,40	11,88	84,5	16,2
TM6	94,44	16,61	82,89	14,21	102,51	18,9

Table A.1: Abilene statistic results

Rediris						
	MCF		Two_P		AC	
	Mean	Stdev	Mean	Stdev	Mean	Stdev
TM1	31,974	2,5103	67,797	4,9362	35,199	1,9301
TM2	36,094	3,5072	75,086	6,7726	39,559	2,7123
TM3	40,266	4,3801	83,15	8,3680	43,896	3,2238
TM4	44,755	5,3182	91,129	9,7839	48,771	4,5581
TM5	6,43	98,739	11,42	53,282	5	77,247
TM6	52,775	6,943	107,14	12,999	57,953	5,966
	Racke		SP_W		SP_L	
	Mean	Stdev	Mean	Stdev	Mean	Stdev
TM1	51,809	5,6356	38,765	2,8274	62,04	4,261
TM2	57,684	6,1673	41,425	4,0535	69,394	6,2925
TM3	63,664	7,1854	43,72	5,2734	76,62	8,179
TM4	70,378	9,52	55,055	6,85	84,314	9,8
TM5	10,976	61,36	7,439	92,019	11,18	
TM6	83,801	12,21	76,565	9,64	98,855	13,27

Table A.2: Rediris statistic results

RediMadrid						
	MCF		Two_P		AC	
	Mean	Stdev	Mean	Stdev	Mean	Stdev
TM1	26,045	3,46	60,362	7,14	32,988	4,05
TM2	32,885	4,56	68,565	9,86	38,137	5,45
TM3	28,135	5,58	75,455	11,14	43,017	6,56
TM4	30,925	6,78	84,093	13,65	48,51	7,98
TM5	49,705	7,62	91,836	15,61	53,431	9,08
TM6	56,135	8,77	100,70	17,7	58,9	10,46
	Racke		SP_W		SP_L	
	Mean	Stdev	Mean	Stdev	Mean	Stdev
TM1	48,412	7,87	32,819	4,83	X	X
TM2	55,218	10,12	38,386	6,34	X	X
TM3	61,546	12,56	43,837	7,66	X	X
TM4	68,492	14,66	49,896	9,25	X	X
TM5	74,542	16,74	55,032	10,49	X	X
TM6	82,45	19,45	61,31	11,74	X	X

Table A.3: RediMadrid statistic results

Surfnet						
	MCF		Two_P		AC	
	Mean	Stdev	Mean	Stdev	Mean	Stdev
TM1	45,59	0,988	78,203	1,83	53,55	1,079
TM2	58,335	2,2815	93,948	3,98	65,44	2,5
TM3	62,94	3,59	110,13	6,13	77,37	4,08
TM4	75,835	4,99	126,75	8,26	89,39	5,65
TM5	90,04	6,14	144,34	10,73	101,74	6,91
TM6	95,915	8,13	160,50	12,73	114,03	8,99
	Racke		SP_W		SP_L	
	Mean	Stdev	Mean	Stdev	Mean	Stdev
TM1	65,97	2,18	53,746	1,55	54,253	1,513
TM2	78,435	4,82	66,121	3	66,62	3,01
TM3	90,67	6,94	78,796	4,89	79,32	4,96
TM4	104,42	9,62	91,82	6,91	92,27	6,99
TM5	117,58	11,71	105,07	8,2	105,63	8,28
TM6	130,32	14,81	118,47	10,82	118,95	10,945

Table A.4: Surfnet statistic results