Scalable Mesh Networks and The Address Space Balancing Problem

Andrea Lo Pumo
Girton College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: al565@cl.cam.ac.uk

May 31, 2010

Declaration

I Andrea Lo Pumo of Girton College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 14980

Signed:

Date:

Abstract

Mesh network architectures are reliable and efficient. They maximize the network throughput with multiple paths and adopt alternative routes when a component fails. Moreover, network applications can optimize their performances by exploiting updated routing informations.
Large scale versions of mesh networks are attractive both for ISPs, as a mean to lower the management cost of their infrastructure, and also for communities, as they can build and sustain city-wide wireless networks without requiring any third party support.
Hierarchical routing protocols are natural candidates for implementing scalable mesh networks. However, when the network is dynamic, the hierarchical topology must be reconfigured after each event. In order to reduce the installation and management costs of a hierarchical mesh network, we propose distributed protocols for automatically creating and maintaining the routing architecture. Also, we derive a set of rules for solving the address space balancing problem, we study their behavior under different network conditions and evaluate their performance as the network becomes larger and more dynamic. We find that, in the worst case, the number of address changes is upper-bounded by $\tilde{O}(N)$, but in a network with a constant churn, the number of reconfigurations increases at least linearly as N grows.

Acknowledgements

I would like to thank my supervisor Jon Crowcroft for always clarifying my doubts, pointing interesting papers and encouraging me throughout the entire project. I am also deeply in debt with Eiko Yoneki for listening to my long explanations of how the little nodes migrate back and forth and for speeding up my simulations with her 8-CPUs machine. Finally, my gratitude goes to Tim Griffin for his frank advice and optimism.

Contents

Chapter 1

Introduction

Mesh network architectures are reliable and efficient: every node acts as an independent router and when a path becomes broken due to a link or a node failure, the network automatically adopts alternative routes. Moreover, nodes can increase their throughput by exploiting the presence of multiple paths.

Thanks to the high availability of low-cost wireless devices, Wireless Mesh Networks (WMN) are becoming the prevalent form of mesh networks. Their applications are numerous and include broadband home LANs, security surveillance systems and metropolitan area networks for transportation systems[1]. WMNs are also effective for extending Internet access to remote rural areas[4] and for combating the digital divide. Among the various application scenarios, community WMNs spanning one or more city neighborhoods are the most interesting. In fact, they support almost all kinds of common network services, such as web servers, multiplayer games, file sharing systems and VOIP, and they can be viewed as a localized small scale version of the Internet. Currently, two major community WMNs are the Athens Wireless Metropolitan Network[2], which reached 2000 nodes in 2008, and the Berlin Freifunk[3].

A large scale version of a mesh network would not only be similar to the current Internet, it would be much better. In fact, in a mesh network each node is a router and every network application can have access to and exploit the information

regarding the routing infrastructure. Thus, applications would be able to make more informed decisions that would improve latency and throughput. For example, multiple idle paths could be simultaneously utilized for increasing the bandwidth between two nodes, or in the case of content distribution networks, the clients themselves could decide what is the nearest replica that is accessible through the least congested path. Routing informations would be particularly useful for distributed services like P2P applications: the virtual links of overlays could be directly replaced with physical optimal paths.

However, with current routing protocols, mesh networks are still not ready for growing to truly large sizes. In fact, they face serious scalability problems: as the number of nodes grows the demand imposed on routers increases rapidly, until a point where they are forced to dedicate all their resources. As an example, in 2009, the mesh network of Internet autonomous systems comprised 350 thousands nodes and, in order to execute the BGP protocol, a router needed 400Mbytes of memory and a 1.1Ghz processor [32]. WMNs are even more sensitive: routers are generally small devices with constrained resources, f.e. Access Points with 32Mbytes of memory and a 200Mhz processor. Additionally, in WMNs the overhead caused by routing packets can heavily decrease the network's throughput[5].

A classic approach for solving the problem of routing scalability is to structure the network into a hierarchical topology. The aggregation induced by the hierarchy allows to achieve routing tables with small size and to reduce the routing update overhead. In this dissertation, our main concern will be the design of distributed protocols for automatically creating and maintaining a hierarchical routing architecture. The automatic configuration of a hierarchical network presents several benefits: first of all, the network installation and management costs are greatly reduced, secondly the network can rapidly adapt to changes: nodes or links can be easily added or removed and in the case of a global attack or a system update the network can be quickly reconfigured from scratch. Furthermore, in the case of WMNs, the ability to automatically configure the network becomes a necessity: the network is dynamic, nodes may be added or removed frequently and the hierarchical constraints may affect other nodes and force their reconfiguration. A manual intervention will require a high cost and the response times will be too

slow for guaranteeing an appropriate continuity of service.

The dissertation is structured as follow: in Chapter 2, we will introduce the background and related works concerning wireless mesh networks. In Chapter 3, we will derive a set of rules for solving the address space balancing problem and in the next chapter, we will describe the protocols for implementing the rules in a distributed way. Finally, in Chapter 5, we will evaluate the performance of the balancing rules and study their behavior as the network becomes larger and more dynamic.

In the next section, we present a concise summary of the work undertaken in the course of the MPhil project.

1.1 Methodology

The fundamental research goal of the project was to verify to what extent hierarchical architectures are applicable to large dynamic networks, such as city-wide WMNs. To this end, we focused on designing distributed rules for automatically reconfiguring the hierarchy after network changes and on evaluating their cost.

Initially, we decided to use some very simple and intuitive rules for obtaining a first understanding of the dynamics involved in the topology maintenance and for discovering their main issues. We wrote a high level simulator for aiding our study of the rules on small topologies. As a result, we collected different examples showing impossibility results and situations that were not correctly covered by a naive approach.

The study of the simple rules helped us to exactly define our problem and to reformulate it in a more general form. After reviewing the literature, we understood that it was strictly related to classic NP-complete problems of graph partitioning and clustering. However, since we needed distributed and non-reset based protocols, we could not apply any known solution.

Before proceeding further in the design of the rules, we theoretically investigated the problem space, trying to understand what were the fundamental limitations

that constrained our design choices. Next, for analyzing the complex dynamics generated by rules, we independently studied the gnode split and the gnode saturation problems. Our main intuition for analyzing the gnode split problem was that a cluster could be approximated as a random graph[1] and that the expected number of reconfigurations could be derived from the size of its giant connected component. For the saturation problem, we analyzed the performance of the rules on simple topologies first and then we generalized the results on arbitrary graphs.

For estimating the overall cost of the rules we derived a natural upper bound on the number of reconfigurations due to dynamic network events. Obtaining a lower bound was harder: the rules were too dependent on the underlying graphs and it was not possible to obtain a meaningful estimate by theory alone. Thus, we started to write a second simulator and designed an experiment for evaluating the rules under churn conditions. This time, writing the software was simpler: we proved that in order to get a lower bound it was sufficient to simulate the rules only on the first level of the hierarchy. This simplification allowed us, not only to get the lower bound, but also to study the behavior of the rules under different network conditions.

Finally, writing the dissertation was a task in itself: we reassembled in a coherent form all the notes written during the project.

[1] this intuition was later confirmed by simulations

Chapter 2

Hierarchical Networks

In a hierarchical network, the nodes are aggregated in groups (or clusters). Each node knows a route to reach any node of its own group, but it does not store all the routes required for reaching outside nodes. Routing update packets are propagated as usual, but when they exit from a group they drop all its internal information. This solution provides a marked saving in the routing table size and in the overhead caused by routing updates.

2.1 Background and Related Works

Kleinrock and Kamoun[6] were the first to study the theoretical properties of hierarchical networks. They showed that the introduced stretch[1] is sufficiently small for networks where the average distance grows as N^v, for a fixed $v > 0$ and where N is the number of nodes in the network. This is the case for wireless

[1] the stretch of a network measures the distortion caused by a non-shortest path routing protocol and is defined as is

$$\max_{x \neq y} \frac{d_R(x,y)}{d(x,y)}$$

where $d_R(x,y)$ is the length of the shortest path from x to y returned by the routing protocol, while $d(x,y)$ is the length of the actual shortest path

networks in a two-dimensional space[8]: when the node density is constant, the average distance is proportional to $N^{\frac{1}{2}}$.

The Internet itself is a hierarchical network of two levels where clusters are represented by Autonomous Systems, and –as observed by Krioukov [7]– almost all proposals for a clean-slate design of a scalable Internet architecture are based, sometimes implicitly, on the concept of hierarchical routing. Also in wireless networks the only routing protocols that have been able to scale are based on hierarchical concepts[9]. However, differently from wired networks, where the topology is designed a priori, in a wireless network new nodes may be added or removed over time and if the nodes are mobile, new links may be established or old ones may become broken. For these reasons, the hierarchical topology must be dynamically constructed and maintained. Moreover, since the routing addresses are automatically assigned a separation between the *location* and the *identity* of a node is introduced. The location of a node is a label used only for routing purposes, while the node's identity is the actual name that network users will have to refer to for contacting the node.

There are two main approaches for automatically configuring a hierarchical topology: defining clusters as neighborhoods of a restricted subset of nodes or creating clusters as groups of nodes of bounded size. The first approach is adopted by single-level or multi-level clustering protocols. In single-level protocols like LANMAR[10], particular nodes called *cluster heads* are selected and their neighborhood of radius r forms a cluster. Multi-level protocols[11],[12], go one step further by extending the hierarchy recursively: among the level l cluster heads, some are elected as level $l+1$ cluster heads.

The second approach for constructing hierarchical topologies is adopted by DART[14] and Netsukuku[16]. Using a graph-partitioning protocol, nodes are merged in connected groups of bounded size S and, recursively, groups are merged into higher level groups. Since the size of a group is bounded, it is possible to associate to each node a routing address that requires a minimum amount of space, namely $O(\log N)$ bits. In comparison, the addresses assigned by multi-level protocols require $O(\log^2 N)$ bits. No virtual links are defined between the groups and distance

vector-like protocols are used for discovering pro-actively the routes.

In multi-level protocols the main difficulty arises in electing and maintaining cluster-heads. When a cluster head dies, all the members of the cluster will have to choose another cluster head as their leader. This is particularly severe when the cluster head belongs to a high level, since all the nodes of that level will have to change their address and update their *identity* $\mapsto$ *location* mapping. The traffic generated by mappings handoff has been estimated as the dominant overhead factor in multi-level cluster networks[19].

In DART and Netsukuku the groups do not depend on the existence of a single node. However, in some situations a group may become *saturated* and new nodes will not be able to join the network. In [15], the authors pose the problem of designing a mechanism for avoiding the saturation of groups, but they leave it as an open problem, which we call the *address space balancing problem*. This is the focus of our project. In essence, it is necessary to balance the clusters each time an upper bound is imposed on their size. For example, in MMWN[12] the authors fix a preferred cluster size and for this reason they are forced to split a cluster in two when its size becomes too big.

The hierarchical model that we will adopt assumes a multi-level topology with L levels and groups of bounded size S. The only constraint imposed on the logical topology is the connectivity constraint: all groups of all levels must be internally connected. This model is strictly related to the DART/Netsukuku hierarchical architecture, but it can be easily adapted to any other with connected groups of bounded size.

We will now give a formal description of the hierarchical model.

Definition 2.1.1. Let $G = (V, E)$ be a connected graph representing the network. Suppose that the maximum number of nodes that can ever join the network is $N_{\max} = S^L$, that is $|V| \leq N_{\max}$, with S, L positive integers and $S \geq 2$. Then we can assign to each node x an address $\overline{x}$ of the form

$$x_0.x_1 \ldots .x_{L-1}$$

where $0 \leq x_i \leq S - 1 \;\; \forall i = 1, \ldots, L - 1$.

For routing purposes, we can allow situations where a node has more than one address but we cannot assign the same address to different nodes. In other words, a proper address assignment is a partial surjective function $\alpha : S^L \longrightarrow V$.

Define the following equivalence relation on V:

$$y \sim_l z \quad \Leftrightarrow \quad y_{\geq l} = z_{\geq l}$$
$$\text{where } x_{\geq i} = x_i \dots x_{L-1}$$

that is, we are identifying nodes with the same l-suffix. We can represent the equivalence class $[y]_l$ of y as follow:

$$[y]_l = *.y_l.\dots.y_{L-1}$$

Note that $[y]_0 = \{y\}$ and $[y]_L = V$.

For each level l and address x, we can then consider the subgraph of G induced by the nodes of $[x]_l$, which is is the subgraph of G formed by all the nodes that have the same l-suffix of x. We will call $[x]_l$ the *network at level* l of x.

By contracting the nodes of $[x]_l$ that have the same $(l-1)$-suffix we obtain the graph called the *group node* (gnode) of x of level l:

$$g_l(x) = (V_l(x),\ E_l(x))$$
$$V_l(x) \ = \ [x]_l/\sim_{l-1} = \ \{[y]_{l-1} \mid y \in [x]_l\} = \{*.y_{l-1}.x_l \dots x_{L-1} \in V \mid 0 \leq y_{l-1} \leq S-1\}$$
$$[y]_{l-1}\,[z]_{l-1} \in E_l(x) \quad \Leftrightarrow \quad \exists y' \in [y]_{l-1}\ \exists z' \in [z]_{l-1} : \ \ y'z' \in E$$

In other words, we subdivide the network in groups and then we recursively proceed to subdivide each group (see figure 2.1). An alternative representation of the group nodes can be given using the language of trees: each $g_l(x)$ is a vertex of a tree T and the elements of $g_l(x)$ are its children, or in other words, the nodes with an address of the form $*.y_l \dots y_{L-1}$ are children of the node $*.y_{l+1} \dots y_{L-1}$.

We will continue to call an element $x \in g_l$ a *node*, while we will call *single nodes* the elements of the original graph G.

A group node g is of level l if there exists a node x such that $g = g_l(x)$. We define $\text{lvl}(g) = l$. The graph formed by all the gnodes of level l is:

$$[G]_l = \bigcup_{x \in G} [x]_{l+1}$$

The links in $[G]_l$ are those induced by the single nodes, i.e. $g, h \in [G]_l$ are linked if a single node $x \in g$ is linked to a single node $y \in h$. With $\Gamma_l(g)$ we indicate the

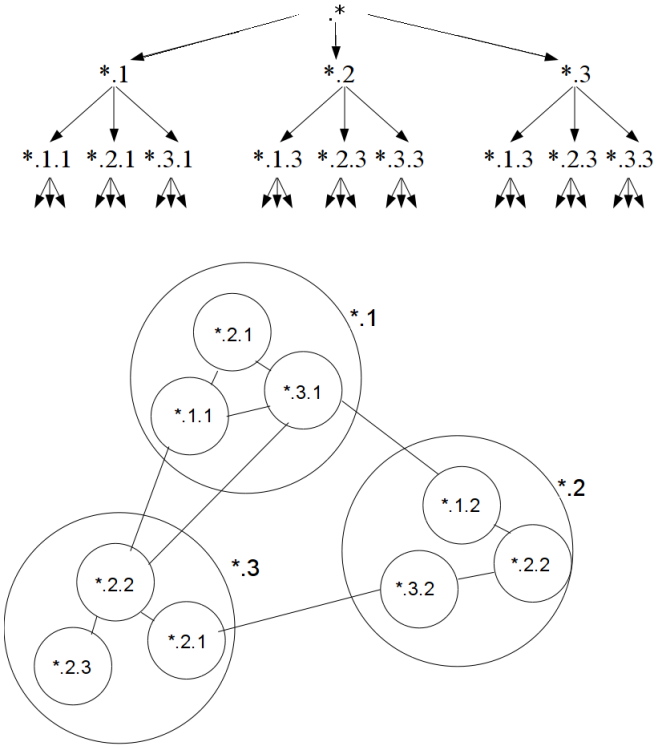

Figure 2.1: A hierarchical topology represented as a tree and as nested groups. The figure represents the first two highest levels $(L-1, L-2)$. The group size is set to $S = 3$.

neighborhood of g in the graph $[G]_l$:

$$\Gamma_l(g) = \{h \in [G]_l \mid g, h \text{ are linked}\}$$

It will be sometimes useful to consider a further level L by fixing for all nodes x, $x_L := \eta$, where η is a constant called *network id*. The group $g_L(x)$, called *network group*, is equal for all the nodes and contains all the group nodes of level $L-1$.

The *size* of level $m < l$ of a group g_l is the number of m-level groups contained in g_l, that is:

$$\begin{aligned} \text{size}_l(g_l) &= 1 \\ \text{size}_k(g_l) &= \sum_{Y \in g_l} \text{size}_k(Y) \end{aligned}$$

With $\text{size}(g_l)$ we indicate $\text{size}_0(g_l)$, i.e. the number of single nodes contained in g_l. Since m-level nodes in $g_l(x)$ have addresses of the form $*.y_m.y_{m+1}\dots y_{l-1}.x_l\dots x_{L-1}$, it follows that $\text{size}_m(g_l) \leq S^{l-m}$ and in particular $\text{size}(g_l) \leq S^l$.
We say that g_l is *full* if $\text{size}(g_l) = S^l$. g_l is *free* if $\text{size}(g_l) = 0$.

If $h = *.y_l\dots y_{L-1}$, then all the gnodes $*.y_i\dots y_{L-1}$, with $i > l$, will be called the *higher gnodes* of h. We define $\text{up}(h) = *.y_{l+1}\dots y_{L-1}$. Analogously, we will talk of *lower gnodes* of h and, by abuse of notation, sometimes we will write $y \in h$ to indicate that y is any lower gnode of h.

A node $y \in h$ is called a *border node* of h if it is linked to at least one node $z \in h'$, with $h' \neq h$ and $\text{lvl}(h) = \text{lvl}(h')$. For example, in figure 2.1, the node $*.2.2$ is a border node of $*.3$. The set $\text{bnode}(h, h')$ contains the border nodes in h that are linked to h'. Notice that $g, h \in [G]_l$ are linked iff $\text{bnode}(g, h) \neq \emptyset$.

We will say that an address assignment forms a *valid hierarchical topology* if for all levels the graph of each group node is connected. For this reason, we will call this requirement the *connectivity constraint* for group nodes.

We will now proceed to describe the main components required for a complete implementation of the above hierarchical network architecture. Later on we will focus on the problem of constructing a valid address assignment for creating a self-configuring network.

2.2 Routing

The main benefit of the connectivity constraint comes from the following proposition

Proposition 2.2.1. *When the network is full, the routing table of each single node contains at most $LS = S\log_S N$ entries.*

Proof: Before proceeding, we give the following definition: let x, y be two nodes, then

$$hdl(x, y) = \min\{0 \leq l \leq L-1 \leq \ | \ x_{\geq l+1} = z_{\geq l+1}\}$$

If $l = hdl(x, y)$, then $g_{l+1}(x) = g_{l+1}(y)$ is the lowest gnode where both x and y belongs. In the language of trees, g_{l+1} is the nearest common ancestor of x and y.

For all levels $l = 0, 1, \dots, L-1$, and for each group g of level l, run a distributed route discovery algorithm on the graph of g, in such a way that at the end of the discovery, each route starting from a gnode g' and contained in $\text{up}(g')$ is known

and stored by all the single nodes of g'. Moreover, for each $l \geq 1$ and $h \in \Gamma_l(x)$, x must also know at least one border node $b \in \text{bnode}(g_l(x), h)$.
With the above protocol a packet can be correctly routed to any destination: suppose x wants to forward a message to z. Let $l = hdl(x, z)$. Both x and z belong to $g = g_{l+1}(x) = g_{l+1}(z)$. By the connectivity constraint, g is connected and there is a path $(g_l(x), y_l, \ldots, g_l(z))$ in g connecting $g_l(x)$ to $g_l(z)$. Since the routing protocol has explored all the gnodes of the networks, and in particular g, the route $(g_l(x), y_l, g_l(z))$ has been discovered and x knows it. Now, the problem of routing the packet from x to z has been reduced to the problem of routing a packet from x to any node of the group y_l and then to z. To reach y_l, x will forward the message to its known border node $b \in \text{bnode}(g_l(x), y_l)$.
We now count how many entries a single node $x = x_1 \ldots x_{L-1}$ stores in its routing table. With the above routing protocol, x stores a routing entry for each $1 \leq l \leq L$ and $y \in *.x_l \ldots x_L$. When the network is full, each gnode $*.x_l \ldots x_L$ has S elements, thus the total number of entries becomes

$$| \{(l, y_{l-1}.x_l \ldots x_L) \mid 1 \leq l \leq L, \;\; 0 \leq y_{l-1} \leq S-1\} | = LS$$

The space required for storing the routing table is thus

$$SL^2 \log_2 S$$

bits. Note that in order to mark a node as a border node, x needs only an additional bit.

□

We now give some remarks on how to implement such a routing protocol.

Any Distance Vector or Link-State routing protocol can be converted into a hierarchical version as follow: when a node $x = x_1 \ldots x_{L-1}$ receives a route (x, y, z), where y is a neighbor of x and z is the destination, x will install the following entry in its routing table:

$$\begin{aligned} &\text{gateway} = y, \;\; \text{destination} = *.z_l.z_{l+1} \ldots z_{L-1} \\ &\text{where } l = \min \{0 \leq l \leq L-1 \leq \; \mid x_{\geq l+1} = z_{\geq l+1}\} \end{aligned}$$

Distance Vector routing protocols do not require any further modification. Instead, Link State protocols are more complicated to implement, as they require an appropriate definition for the weight of the virtual link that connects two groups.

With a hierarchical topology it is also possible to prevent loops of flooding packets in a simple way: each time a node forwards a routing discovery packet, it appends its address at the end of the packet. A node will discard a packet if it finds its address in the appended list. There is no risk that the list will become too large: when the packet exits from a group node, all its internal addresses are discarded,

i.e. when the packet contains a list of the form

$$
\begin{aligned}
&* .x_l^1.x_{l+1} \dots x_{L-1},\\
&* .x_l^2.x_{l+1} \dots x_{L-1}\\
&\vdots\\
&* .x_l^m.x_{l+1} \dots x_{L-1}\\
&* .y_l.y_{l+1}.x_{l+2} \dots x_{L-1}
\end{aligned}
$$

it is rewritten to

$$
\begin{aligned}
&* .x_{l+1} \dots x_{L-1}\\
&* .y_l.y_{l+1}.x_{l+2} \dots x_{L-1}
\end{aligned}
$$

This means that once a packet exits from a group node, it will not return inside. A gnode can have a maximum of S nodes, thus its diameter is also bounded by S. It follows that in the worst case the list appended in a packet will contain $(S-1)L$ entries.

2.3 Hierarchical Distributed Hash Table

The addresses of the hierarchical topology are used to encode connectivity information and are thus not arbitrary. For this reason, a separated mechanism is needed in order to give an *identity* to nodes.

An easy way to solve the problem is to set up a classic Domain Name System where few single nodes function as DNS servers. A much better way is instead to exploit the hierarchical topology and build a Distributed Hash Table (HDHT) that will store the associations between names and addresses.

We now describe how to construct a HDHT. Let S^L be the address space, V_t the set of nodes of the network at time t and $\alpha_t : S^L \longrightarrow V_t$ the address assignment at time t. Let K be the *key space*, which we can assume to be larger than the address space ($S^L \subseteq K$). The aim of a DHT is to maintain at each time t a function $d_t : K \to D$, where D is the *data space*, f.e. strings of few bytes. The function d_t is *distributed* among the nodes of the network: each node stores in its memory a subset of d_t, i.e. a small set of mappings $\{k_1 \mapsto d_t(k_1),\ \ k_2 \mapsto d_t(k_2),\ \ \dots,\ k_m \mapsto d_t(k_m)\}$. Also, d_t varies through time: a node might request to change the mapping $k \mapsto d(k)$ to $k \mapsto d'$. The basic idea for building the HDHT is to let the node $\alpha_t(h(k))$ store the mapping $k \mapsto d(k)$, where $h : K \to S^L$ is a hash function. Thus, in order to

retrieve such a mapping or to change it, the nodes will contact the node $\alpha_t(h(k))$. However, if the network is not full ($V_t \subsetneq S^L$), α_t might be a partial function not fully defined on S^L, i.e. some addresses might not be assigned to any node. This matter is solved with another dynamic function $H_t : S^L \longrightarrow \mathrm{dom}(\alpha_t)$. Given an address x, H_t returns another address $H_t(x)$ that has been already assigned to a node. H_t can be implemented in a distributed way as follow:

1. define $H_t(x)$ as the nearest address to x associated to an alive node:

$$H_t(x) = \mathrm{minarg}_{x' \in \mathrm{dom}(\alpha_t)} \mathrm{abs}(x - x')$$

   (note[2]) x, x' are considered as vectors and are compared using the lexicographic order where the most significant digit is the last one (x_{L-1}). abs is defined component-wise.

2. a node does not need to have the complete knowledge of $\mathrm{dom}(\alpha_t)$, i.e. it does not need to know if for an address there's a corresponding alive node in the network. A node $y = y_0 \dots y_{L-1}$, by looking at its routing table constructed with the routing protocol described above, knows what are the alive nodes of $y_l \quad \forall l = 1, \dots, L$ (note [3]). If $y_{l-1} \in y_l$ is an alive node, then there is an alive single node with an address of the form $*.y_{l-1}$. Let $\mathrm{dom}_y(\alpha_t)$ be the set of all the alive nodes known by y.

   When a packet has to be sent to $H_t(x)$, it will be *routed* to $H_t(x)$ using a greedy algorithm: when y receives the packet, it forwards it to the group node with address

$$H_t^y(x) = \mathrm{minarg}_{x' \in \mathrm{dom}_y(\alpha_t)} \mathrm{abs}(x - x')$$

In sum, the node associated to a key k is $\alpha_t(H_t(h(k))$.

Notice that a HDHT is more efficient than a classical DHT like CHORD[17]. In fact, in DHTs, the request is forwarded multiple times between nodes. For instance, CHORD requires $O(\log N)$ forwardings. Instead, a HDHT is built on top of the routing infrastructure of the network. Each read/write request is directly routed to the correct node, so that the number of required forwardings is 1.

The latency for querying and updating a mapping can be further optimized by extending the HDHT: suppose the single node z wants to read or update a mapping $k \mapsto d$, where k is the key and d is the data. Instead of querying directly the node $h(k)$, z will do the following:

[2] there can be two addresses x', x'' that minimize $\mathrm{abs}(x - x')$, in this case we pick $\min\{x', x''\}$

[3] $y_L = \eta$ is the network group

1. let $h(k) = h_0 \dots h_{L-1}$ and $z = z_1 \dots z_{L-1}$.
2. z will query in order:

$$
\begin{aligned}
&h_0.z_1 \dots z_{L-1} \\
&h_0.h_1.z_2 \dots z_{L-1} \\
&\vdots \\
&h_0.h_1 \dots h_{L-1} = h(k)
\end{aligned}
$$

With the above schema, the HDHT is sliced in levels: a node will initially query nodes in its same group of level 1, then nodes in its same group of level 2 and so on, until it finds a result. Each time it goes up of one level, the destination node may be potentially located farther, in terms of routing distance, and vice-versa, finding an answer in lower levels may be more profitable.

Chapter 3

Balancing the Address Space

In this chapter, we will discuss how to construct and maintain a proper address assignment that structures the network in a hierarchical topology.

3.1 Related Problems and Works

The problem of constructing a proper address assignment from scratch is not easy, in fact, it is strictly related to the Bounded-Connected-Graph-Partitioning (BCGP(G, M, k)) problem: given a graph $G = (V, E)$ and two integers $M, k > 0$, decide if there is a partition of its vertices $V = V_1 \cup \ldots \cup V_m$ such that

1. $m \leq M$
2. each component V_i is connected
3. $1 \leq |V_i| \leq k \;\; \forall i = 1 \ldots m$

When k is fixed to 4, the above problem is called Bounded Component Spanning Forest (BCSF) and it is known to be NP-Complete[18].

Proposition 3.1.1. *Given a graph $G = (V, E)$ and $S > 0$, the problem of deciding if there is a proper address assignment $\alpha : S^L \longrightarrow V$ such that*

1. $L = \min\left\{L \geq 1 \mid |V| \leq S^L\right\}$
2. *each node has no more than one address*

3. the number of gnodes of level 1 *is no more than* M

is NP-hard.

Proof: With an assignment as above and $S = k$, the nodes are partitioned in connected groups of level 1 and $|g| \leq S = k \quad \forall g \in [G]_1$. Thus, it is possible to decide if BCGP(G, M, k) is true or not. □

Different solutions and heuristics have been proposed for constructing a solution to the BCGP problem or one of its variations. The solution presented in [21] selects a spanning tree rooted at a random vertex. By traversing the tree from the leafs, vertices are aggregated in connected components. The authors show that in the case of Random Geometric Graphs their algorithm can achieve a small number of common vertices between two components. The distributed version of the algorithm works by creating the tree with flooding. Finally, their algorithm is reset based: there are some cases where it is necessary to rebuild the clustering from scratch.
By requiring that the size of all components is almost the same ($0 \leq |V_i - V_j| \leq 1 \quad \forall i, j$), the BCGP becomes closely related to the problem of Graph Partitioning, which has been extensively studied for applications such as VLSI circuit layout, image processing and matrix computations[22]. For generating an initial partition, graph partitioning algorithms generally resort to either spanning tree techniques or to graph growing. Graph growing algorithms initially form groups of size one by selecting a random subset of vertices (seeds), afterwards neighboring vertices are iteratively added, enlarging the groups. If a group becomes too large, the procedure is recursively applied to the group.

Unlike the above works, in this project we are interested in an incremental, distributed solution to the address assignment problem: as the network evolves the address assignment must be updated and the nodes must be able to change their address without having a global knowledge of the network. Also, as explained in Chapter 5, the update has to minimize the number of address changes.

3.2 Dynamic Balance

In a distributed implementation of a self-configuring network, the nodes have to choose their own addresses. For this reason, from now on, we will view the task of finding a proper address assignment as an evolving distributed process. We will say that a node *joins* a group node $g_l(y)$ when it chooses an address x such that $x \sim_l y$. Analogously, a node can *leave* a group and can *migrate* from a group to another. Further, we say that x *creates* or *allocates* a gnode g_l, if x joins g_l and it is its unique node, i.e. $g_l = \{x\}$.

The connectivity requirement for having a valid hierarchical topology is a strong constraint and it is the cause of the *gnode split* and of the *address space balancing* problems.

The first problem arises while trying to maintain a proper address assignment. The removal of a node or a link may disconnect a gnode g_l in multiple connected components $g_l = A_1 \cup \ldots \cup A_m$ (gnode split). When this happens, the connectivity constraint is not satisfied anymore. See figure 3.1 for an example.

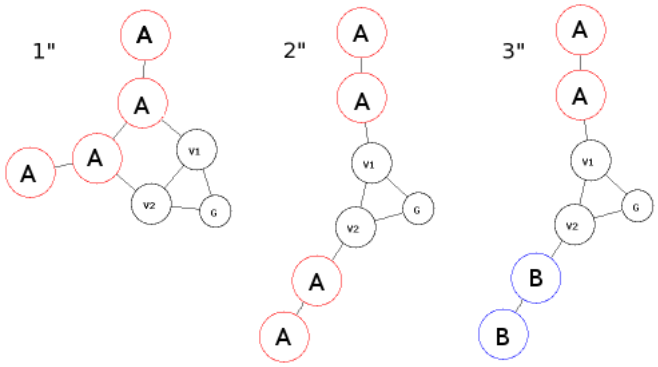

Figure 3.1: The removal of a link disconnects the group node A in two connected component. The nodes of one component will have to change their membership by migrating into another gnode B.

Notice that a split of a group $*.g_l \ldots g_{L-1}$ may induce a split of one of its higher gnodes $*.g_{l'} \ldots g_L - 1$, with $l' > l$. This happens when g_i is an articulation point[1] in g_{i+1}, $\quad \forall i = l, \ldots, l' - 1$.

[1] a vertex x of a connected graph is an articulation point if there are two distinct nodes such that all the paths that connect them pass through x

The only solution to the gnode split problem is to promptly repair the address assignment: the single nodes of the components $A_2, \ldots, A_m$ are forced to change their address and to migrate into other groups.

The address space balancing problem arises when a new node joins the network:
Proposition 3.2.1. *As a consequence of the connectivity constraint, there are some configurations where a new node cannot join to a gnode g_l, even if* $\text{size}_0(g_l)$ *is not full. In this case we say that g_l is* saturated.
Moreover, the address space of the whole network can be saturated with just $(S-1)L+1$ nodes.

Proof: Let $g_l = *.y_l \ldots y_{L-1}$ and suppose that $\text{size}_1(g_l)$ is full, i.e. g_l contains all the possible gnodes of level 1, or in other words, $*.y'.y_2 \ldots y_{L-1}$ has at least one node, $\forall 0 \leq y' \leq S-1$. Suppose further that a gnode $h = *.y'_1.y_2 \ldots y_{L-1}$ is full. Finally, suppose that a node x is linked to nodes of h only. Since h is full, in order to preserve the uniqueness of addresses, x cannot join to h. Moreover, by the connectivity constraint, x will not be able to choose any other address of g_l, i.e. it will not be able to join to any $*.y.y_2 \ldots y_{L-1}$, with $0 \leq y \leq S-1$.
The whole network g_L can be saturated as described above, however we need much less than $|\,\text{size}_1(g_L)| = S^{L-1}$ nodes. In fact, we can saturate it as follow:
1. First, turn on only $S-1$ nodes and to each of them assign an address of the form $*.y$, with $1 \leq y \leq S-1$. Ensure also that they are connected.
2. Consider other $S-1$ nodes and ensure that they form a connected graph. Let them join the network, using an address of the form $*.y.0$, with $1 \leq y \leq S-1$.
3. continue recursively: $*.y.0.0$, $*.y.0.0.0$,..., until adding the nodes $y.0.0 \ldots 0$.
4. finally, add the node $0.0 \ldots .0$

In this network, a node x that is only linked to nodes of the gnode $*.0.0 \ldots 0$, will not be able to join. □

In order to avoid network saturation, we need to balance the address space: the address assignment has to be updated over time in order to let any new node acquire a proper address if the network is not full.

The requirement of having a valid address assignment restricts the choice of how a balancing protocol reconfigures the network.
Proposition 3.2.2. *Consider the situation described in Proposition 3.2.1, where there are no more free gnodes left and a node x is forced to join to a full group h. In this case, if there is a proper address assignment, then we have only two solutions:*

1. *either a node migrates from h*
2. *or a gnode g is emptied*

With solution 1., *x is then able to join to h, instead, with solution* 2., *x can re-create the gnode g.*

Let M_2 be the minimum number of migrations required for applying the solution 2., *and M_1 that required for applying* only *the solution* 1., *then we have:*

1. $M_1 = \min\{\text{length}(P) - 1 \mid P$ *is a migration path starting from* $h\}$
   where a migration path *is defined in the proof below*
2. $M_2 \geq \min\{|g| \mid g$ *gnode*$\}$

Proof: Consider any new proper assignment and let $y_1, \ldots, y_S$ be the nodes in the group h of the old assignment. Let $g(y_i) = \text{up}(y_i)$. In the new assignment we have two cases: either $\forall i, j\ g(y_i) = g(y_j)$, or not.
In the former case, x has necessarily joined to a gnode g different from h. However, since x is only linked to nodes of h, by the connectivity constraint x is the only node of g. In other words, x has created g. By hypothesis, all the groups were not free, thus the old nodes of g have migrated, i.e. g has been emptied and x has re-created it.
In the latter case, at least one node has migrated from h.
The number of migrations required in the former case is at least $|g|$. It can be larger, f.e. if the migrations from g force other migrations. Thus, we have

$$M_2 \geq \min\{|g| \mid g \text{ gnode}\}$$

Call a path P starting from h a *migration path* if

$$P = (p_1, \ldots, p_m),\ \ p_1 = h,$$
$$\forall i \leq m-1 \quad p_i \text{ is full},\ \ p_i \text{ is a gnode linked to } p_{i+1},\ \ \text{lvl}(p_i) = \text{lvl}(h),$$
$$p_m \text{ is not full},\ \ \text{lvl}(p_m) \geq \text{lvl}(h)$$

If solution 2. is forbidden, we do not have any other choice than to repeatedly apply solution 1.. That is, a migration path is selected and at least a node migrates from p_i to p_{i+1}, for all $i \leq m-1$. As we will see in Prop. 3.2.11, we can force the migration of exactly one node from p_i to p_{i+1}. Thus, the number of migrations required is $\text{length}(P) - 1$ and it is minimized by the shortest migration path. □

Remark 3.2.3. Between the two solutions presented in Prop. 3.2.2, we prefer to adopt the first, for two main reasons:

1. The second solution forces all the nodes of a gnode g to migrate. This implies that some other gnodes will increase their size. As a consequence, the network may reach the saturation point quicker.
2. In a distributed implementation where a group does not know the size of the other groups, the first solution requires less communication overhead: in

order to find a shortest migration path, the gnode h queries its surrounding gnodes using a BFS[2]-like exploration, which is stopped as soon as a shortest path is found. Instead, in the second solution, at least the size of all group nodes has to be discovered[3].

Remark 3.2.4. In Proposition 3.2.2, we described what are the necessary solutions for fixing a saturated network. However, instead of fixing the network at the last minute, we might try to avoid to fill up a group, unless it is strictly necessary, and try to always keep the network saturation-free. Also in this case, we do not have much choice on how to reach a new address assignment. In fact, we cannot predict in what group the new nodes will join, i.e. we must assume that any group can increase its size. Thus at some point, we must decide if the size of a group is too large, and force at least one node of the group to migrate. This means that we have to use a condition $\rho(|g|)$ that depends on the size of the group g, and possibly on other parameters. When $\rho(|g|)$ is true, a border node of g will be forced to migrate. The simplest condition is obtained by fixing

$$\rho(|g|) \equiv (|g| > S)$$

In this case, a migration will occur only when g is full and a new node x joins. If x joins to g only when it is forced to do so, then this becomes the same solution 1. of Prop. 3.2.2. Instead, by fixing

$$\rho(|g|) \equiv (\exists h : \ |g| \geq |h| + 2)$$

a migration will occur only if g is bigger than one other gnode[4]. This condition is the opposite of the previous one: as soon as possible a node will migrate.

We will later see in more details the above two *balancing rules*.

Notice that, in any case, the migration of a node from g can make $\rho(|g|)$ false, but $\rho(|h|)$ true, for some other gnode h. Thus, in order to avoid infinite back and forth migrations from g to h and from h to g again, migration paths become a necessity.

[2]Breadth First Search

[3]We say "at least", because finding the group g that minimizes the number of migration is not just a matter of knowing its size. In fact, in Prop. 3.2.2, $\min\{|g| \mid g \text{ gnode}\}$ is a lower bound of M_2.

[4]if $\text{size}(g) \geq |h| + 1$ is used instead, a loop can occur: a node may endlessly migrate back and forth from g to h

They are redefined as follow:

P is a migration path $\Leftrightarrow P = (p_1, \ldots, p_m)$, $p_1 = h$, p_i is a gnode linked to p_{i+1}
$\forall i \leq m-1 \quad \rho(|p_i|)$ is true
$\rho(|p_m|)$ is not true

Proposition 3.2.5. *Suppose that all the lower gnodes of h are allocated, then it is possible to change the address of a node $x \in h$, only if x is a border node of h.*

Proof: In fact, by the connectivity constraint, a node $y \in h$ linked only to nodes of h is forced to remain in h. Thus the only nodes that can migrate are the border nodes. The vice-versa does not always hold, because even if y is a border node, if all its neighboring gnodes are full, then it cannot join them. □

The migration of a border node b may affect the topology of the higher levels. For example, suppose that b is the unique border node in $\text{bnode}(g, h)$. If b migrates to $f \neq h$, then g will become linked to f but will loose its link with h. In general, the following proposition holds:
Proposition 3.2.6. *Let $g, h \in [G]_l$. If a border node b migrates from g to h and $|g| \geq 2$, the resulting gnodes g', h' satisfy:*

$$1 \leq |\Gamma_l(g')| \leq |\Gamma_l(g)|$$
$$0 \leq |\Gamma_l(h)| \leq |\Gamma_l(h')|$$

Also, g may lose one of its links and $\text{up}(g)$ *may become split, but $[G]_l$ remains connected.*

Proof: Since $|g| \geq 2$ and $b \in g$, the connectivity constraint implies that b is linked to at least one other node $x \in g$. Thus when b migrates, x will become a border node and $|\Gamma_l(g')| \geq 1$.
Since the border node b is a new node in h', it follows that $|\Gamma_l(h')| \geq |\Gamma_l(h)|$. Equality holds when h was already linked to all the neighboring gnodes of b.
Finally, if h was the unique border node in $\text{bnode}(g, h)$, then g' is no more linked to h'. Thus, $|\Gamma(g')| < |\Gamma(g)|$.
Let $\Gamma(g) = \{h_1, \ldots, h_m\}$, with $h = h_1$. $[G]_l$ remains connected because a broken link (g, h_i) can be replaced by the path (g, h_1, h_i). However, if $h_1 \notin \text{up}(g)$, then (g, h_1, h_i) is not a path contained in $\text{up}(g)$. If this was the only path connecting g to h_i, then $\text{up}(g)$ becomes split. □

We have a result similar to the previous proposition in the case of gnodes migration:
Proposition 3.2.7. *Suppose that a gnode g of level l migrates, then the graph $[G]_{l-1}$ does not change, and the new $[G]_l$ is isomorphic to the old one.*

Proof: This follows directly on how the gnodes migrate. Let $g = *.g_l \dots g_{L-1}$. When g migrates, it will assume another address of the form $*.g'_l \dots g'_{L-1}$, thus a node of level $l-1$ will change its address from $x = *.g_{l-1}.g_l \dots g_{L-1}$ to $x' = *.g_{l-1}.g'_l \dots g'_{L-1}$. If a single node was a member of x it will still be a member of x'. In other words, the "inside" of g_{l-1} has not changed. Thus, the links between $*.g_{l-1}$ the others $*.h_{l-1} \in [G]_{l-1}$ are still the same. (What could have changed are the links between $*.g_{l+1}$ and another $*.h_{l+1}$.)
Finally, $[G]_l$ is isomorphic to $[G']_l$ because their only difference is the name of the gnode $*.g_l$, which has been changed to $*.g'_l$. □

It is not always possible to solve the address balancing problem, i.e. in some cases, some nodes of the network will not be able to join:
Example 3.2.8. Not all graphs admit a proper address assignment.

Proof: Consider the string topology formed by $N = S^L > 1$ nodes, that is, if the nodes are $v_1, \dots, v_N$, then $v_i v_{i+1} \quad \forall i = 1, \dots, N-1$ are all the edges. Now, attach a dangling node q to v_S, i.e. $v_S q$ is a link. Then this network does not have a proper address assignment, in fact, suppose by contradiction that it has one.
First observe that since the network is full ($N = S^L$), all the gnodes of any level are full too, i.e. all the addresses have been used and

$$|g| = S \quad \forall g \text{ gnodes} \qquad (1)$$

This means that $v_S \in g$ for some gnode g of level 1. Let j be such that

$$j = \min \{i = 1, \dots, S \mid v_i \in g\}$$

Consider the case where $j > 1$. We have,

$$v_1, \dots, v_{j-1} \notin g \quad (2)$$

Let $H \subseteq \{v_1, \dots, v_{j-1}\}$ be a maximal subset such that $\forall x, y : \ g_1(x) = g_1(y)$, that is all the elements in H are in the same level 1 gnode and all the other $v_i \notin H$ are not. Since $|H| \leq j - 1 < S$, by (1) it follows that H cannot be a complete gnode, that is
$\exists w \notin H : \ g_1(w) = g_1(x) \ \ \forall x \in H$
H is by definition maximal $\Rightarrow w \notin \{v_1, \dots, v_{j-1}\} \Rightarrow w = v_{j+h}$
it is not possible that $w = v_j$, otherwise $g_1(H) = g_1(v_j) = g$ and this is in contrast with (2)

$$\Rightarrow \qquad w = v_{j+h}, \ h > 0$$

so, we have found a w which is in the same group of the elements in H, but is not linked to any of them. This contradicts the connectivity constraint.
Consider now the case where $j = 1$. Since $|g| = S$, we have

$$\{v_1, \dots, v_S\} = g \qquad (3)$$

Now recall that the dangling node q is linked only to v_S, thus by the connec-

tivity constraint, the only gnode where it can belong to is $g = g_1(v_S)$. But this contradicts (3).
In any case, we have shown a contradiction. Therefore no proper address assignment is possible. □

Example 3.2.9. Reaching a valid assignment is sometime impossible through local address changes.

Proof: Consider the case where a group g has only one border node b with a group h. Suppose also that b is forced to migrate to h. If b is an articulation point of g, then its migration will disconnect g. Suppose h is full, then one component of g might not be able to follow b and migrate to h and it will remain completely disconnected from the network. □

Definition 3.2.10. Clearly, the problems exposed in the above two examples can be solved at the cost of increasing the parameter S. Another workaround is to add new links: suppose a migrating node b splits a group g, then *virtual links* will be established between the old neighbors of b belonging to g. In this way, g remains connected. A virtual link between $x, y \in g$ is removed when x or y leaves g, or when a new link reconnects g.

Creating virtual links in g can be still seen as changing the parameter S, but only locally to g: when the node b migrates from g to h, it assumes two addresses and belongs at the same time to g and h. In g, b is seen as a *virtual node*, with a non standard address of the form $b' = *.(S + k).g_{l+1} \dots g_{L-1}$. The virtual node b' does not act as a border node, i.e. it does not maintain links with nodes outside g, and when its neighbors leave g or g becomes reconnected, it is removed from the network.

With the use of virtual links, the following proposition holds.
Proposition 3.2.11. *If one node migrates from a group, the group will not be disconnected and no other migrations will occur.*

Although, there are pathological cases where an arbitrary number of virtual nodes are added, we will later see in simulations that their use is rarely needed for random networks with $L = 1$.

3.3 Last Minute and Preemptive Balancing

The two balancing rules presented in Remark 3.2.4 adopt two different strategies,

1. Last-Minute: the addresses of some nodes are changed only if the network is saturated and a new node cannot join,

2. Preemptive: at each network event, the addresses of some nodes are changed so that a new node can immediately join without requiring a further reconfiguration of the network.

The Preemptive strategy seems attractive because the network is constantly kept saturation-free, however it also requires a migration each time a new node joins, while the Last-Minute rule will force one migration only when necessary.

We will now describe the Preemptive Balancing rule (PB-rule) and later the (LM-rule). The protocols for ensuring their distributed implementation are presented in Chapter 4.

The PB-rule works as follow:

1. At first fix $l = L - 1$, and iteratively apply the following procedure, lowering l by one each time it ends, until $l = 1$.
2. For any gnode h of level l ($h \in [G]_l$), let $\Gamma(h) = \Gamma_l(h)$.
3. If all the neighbors $y \in \Gamma(h)$ are such that $|h| \leq |y|$, then stop. Otherwise,
4. if there's any neighbor y such that $|h| \geq |y| + 2$, then consider y' s.t.

$$y' = \operatorname{minarg} |y|, \quad \text{where } y \text{ ranges in } \{y \in \Gamma(h) \mid |h| \geq |y| + 2\}$$

   let exactly one node migrate from h to y and stop. Otherwise,

5. if there is a neighbor $y \in \Gamma(h)$ s.t. $|h| = |y| + 1$, then let g in $[G]_l$ be any of the nearest gnodes to h such that $|h| = |g| + 2$. If such a g does not exist, stop. Otherwise, let $h = p_1, p_2, \ldots, p_m = g$ be a shortest path connecting h to g. Notice that by definition of g, we have

$$p_1 - 1 = p_2 = \cdots = p_{m-1} = p_m + 1$$

   Finally, for each $i = 1, 2, \ldots, m-1$, let exactly one node from p_i migrate to p_{i+1}. After the migrations, the new configuration will be such that

$$|p_1| = |p_2| = \cdots = |p_m|$$

Remark 3.3.1. When the PB-rule has selected a migration path $p_1, \ldots, p_m$, the migrations have to happen in the order $p_1 \to p_2$, $p_2 \to p_3, \ldots, .$ Otherwise, suppose $p_{i+1} \to p_{i+2}$ happens before $p_i \to p_{i+1}$, then the link $p_i \to p_{i+1}$ could become broken if the unique border node connecting p_{i+1} with p_i has migrated to p_{i+2}.

Remark 3.3.2. Steps 3., 4. and 5. can be replaced by a single step: substitute in step 5. the condition $|h| = |y| + 1$, $|h| = |g| + 2$ with $|h| \geq |y| + 1$, $|h| \geq |g| + 2$. In

this way the migration path $p_1, \ldots, p_m$ can be of length $m = 1$, $m = 2$ or $m \geq 3$. From a global point of view, we can restate the PB-rule as follow: for any level l, find a shortest migration path with a smallest gnode at its end.

Remark 3.3.3. The non-deterministic steps of the PB-rule (4. and 5.) can be made deterministic by applying different heuristics:

1. suppose that the group g has more than one border node that can migrate. In this case, the border node whose removal does not induce a split of the gnode g is preferred.
2. Suppose that x can join to more than one gnode $g_1, \ldots, g_m$. When it decides to join to g_i, it becomes one of its border nodes and some routes may use x as a gateway to reach nodes in g_i. Thus, in order to reduce the latency stretch, the node x will prefer to join/migrate to the gnode g_i such that $\max_{y \in g_i} d(x, y)$ is minimized.

The main reason for using the Path Balancing rule is that it constantly keeps the network saturation-free by satisfying the following property:

Proposition 3.3.4. *When the PB-rule terminates at level l,*

$$\forall g, h \in [G]_l \quad 0 \leq \mathrm{abs}(|g| - |h|) \leq 1$$

Or in other words, the gnodes in $[G]_l$ have almost the same size.

Proof: Consider the set of all the shortest migration paths in $[G]_l$:

$\mathrm{NonIncrPaths}([G]_l) = \{(p_1, \ldots, p_m) \mid p_i \text{ is linked to } p_{i+1} \text{ in } [G]_l,\ |p_i| \geq |p_{i+1}|\ \forall i\}$

$\mathrm{MigrPaths}([G]_l) =$

$= \{p \in \mathrm{NonIncrPaths}([G]_l) \ \mid \ |p_1| \geq |p_{\mathrm{length}(p)}| + 2,$
$\mathrm{length}(p) = \min \mathrm{length}(\mathrm{NonIncrPaths}([G]_l))\}$

If $\mathrm{MigrPaths}([G]_l)$ is empty, then

$$\forall g, h \in [G]_l \quad 0 \leq \mathrm{abs}(|g| - |h|) \leq 1 \qquad (1)$$

is true and the PB-rule terminates.

Suppose that $\mathrm{MigrPaths}([G]_l)$ is not empty. Define the *imbalance* of $[G]_l$ as:

$$I = \sum_{p \in \mathrm{MigrPaths}([G]_l)} I(p)$$
$$\text{where } I(p) = \max\{p_1 - p_{\mathrm{length}(p)} - 1, 0\}$$

Notice that

$$G \text{ finite } \Rightarrow |\mathrm{MigrPaths}([G]_l)| < \infty \Rightarrow I < \infty$$
$$I(p) > 0 \Leftrightarrow |p_1| \geq |p_{\mathrm{length}(p)}| + 2$$
$$I = 0 \Leftrightarrow \mathrm{MigrPaths}([G]_l) = \emptyset$$

The PB-rule selects one path $p \in \text{MigrPaths}([G]_l)$ and, thanks to Prop. 3.2.11, it makes exactly one node migrate from p_i to $p_{i+1} \quad \forall i = 1, \ldots, \text{length}(p) - 1$. Thus, $I(p)$ and I decrease. Since I is finite, eventually, it becomes 0. □

Corollary 3.3.5. *If in the network there is a non full gnode in $[G]_l$, with $1 \leq l \leq L-1$, then any new single node will be able to join.*

Proof: Suppose that the new node x is linked to a node $y \in h$ of level $l' < l$. We have two cases. In the first, $h \in [G]_l$ is not full. Then, x can directly join to h by taking an address of the form $*.x_{l-1}.h_l \ldots h_{L-1}.h_L$. The connectivity constraint is satisfied because x is linked to y and thus also to $h_i \ \forall i \geq l'$.
In the second case, h_i is full $\forall i \geq l$, but by hypothesis, there is a non full gnode $g \in [G]_l$. In this case, let x assume a temporary address of the form $*.x_{l-1}.h_l \ldots h_{L-1}$, with $x_{l-1} \geq S$ (note[5]). This will change the size of h_l to $|h'_l| = |h_l| + 1 = S + 1$. Since the PB-rule applies at all levels, when it terminates we have that

$$0 \leq \text{abs}(|f| - |g|) \leq 1 \ \ \forall f \in [G]_l \underbrace{\Rightarrow}_{|g|<S} |f| \leq S \ \ \forall f \in [G]_l \quad (2)$$

After the migrations x belongs to some f and by (2) it can can now assume a proper address, with $0 \leq x_{l-1} \leq S - 1$. □

We now proceed to describe the Last Minute rule (LM-rule). We will later see that it is more efficient that the PB-rule.
Consider the procedure of the PB-rule, then the LM-rule's procedure is obtained by substituting the size $|\cdot|$ operation with the following:

$$\|X\| = \begin{cases} -1 & |X| < |S| \\ |X| - |S| & \text{else} \end{cases}$$

(note[6]) In this way, we have

$$|h| < S, \ |g| = S, \ |b| = S + h \ \Rightarrow \ \|b\| > \|g\| > \|h\|$$

and a gnode will apply the balancing rule only when it becomes full.

The Corollary 3.3.5 still holds with the due changes, thus also the LM-rule avoids saturation.
Remark 3.3.6. The LM-rule is a generalization of the PB-rule. In fact, if S in the

[5] we are temporary violating the constraint of using an address in $\{0, 1, \ldots, S-1\}^L$

[6] we are improperly stating that the size of a gnode can become larger than S. What we really mean is that if a gnode X is full and h nodes want to join in X, then $|X| = S + h$.

definition of $\|\cdot\|$ is replaced by a parameter S_0, then with $S_0 = 0$ we obtain the PB-rule.

We end this chapter with the description of the rules that allow a new node to join the network.

Definition 3.3.7. Suppose that a node x is turned on. If x is now connecting multiple disconnected networks, then x will prefer to join to the largest one. x has two strategies for joining to a network:

1. (Dispersive-rule) if there is a free gnode of level $L-1$, then x will create it. Otherwise, let λ_g be the length of the migration path that is created by the balancing rules when x joins to g. Then, x joins to the neighbor $g \in \Gamma_l(x)$ that minimizes λ_g, where l is the maximum level s.t. $[G]_l$ contains a non-full gnode.
2. (Aggregative-rule) even if there is a free gnode that x can create, x prefers to join to a neighboring gnode, as described above.

Intuitively, the Aggregative rule is more costly because it moves the configuration toward saturation. We will later see that simulations confirm this intuition.

Chapter 4

Distributed Balancing Rules

The balancing rules can be naturally translated to a distributed version: the only information required for constructing a migration path is the size of the groups, which can be known from the routing tables of their nodes. Moreover, the migration paths starting from a given group can be locally discovered using a BFS-like exploration. The BFS search does not flood all the single nodes because a group can be visited by selecting only one of its nodes. The discovery of migration paths is started by border nodes: when $b \in \text{bnode}(g, h)$ receives a routing update and notices that $\|g\|$ has become too large (or too small) with respect to $\|h\|$, it will try to create a migration path and migrate.

In the section below, we will see that the actions of multiple border nodes can be coordinated with atomic distributed locks. Next, we will describe how the nodes can repair a gnode split and how two separated networks can be merged once they become connected.

4.1 The Memory of a Group

There are various situations where events need to be serialized:

1. when two or more nodes want to simultaneously join to the same group they cannot act independently, otherwise there is the risk that they will choose the same address;
2. the migration of a group node of level $l \geq 1$ is not instantaneous: its single nodes change address one by one. Suppose that g decided to migrate because a condition Q was true. While the migration process runs, the condition Q

might become false and the remaining nodes in g will not have any reason for continuing the migration. For example, a condition Q can be $Q =$"the group G has become split".

Also, if g is full, it cannot accept new nodes and this is true also when it is migrating. However, since during the migration its size gradually decreases, a new node might believe that g is not full and may join to it.

A solution to the above problems is to take one step further in viewing group nodes of level l as nodes belonging to groups of level $l+1$. We define the *memory of a gnode* as the atomic distributed memory formed by its single nodes.

There are different ways for forming an atomic distributed memory. A simple one is the following: let $\mu_0(g) = \min g$ be the single node in g with the lowest address. Then the memory of g is identified with the memory of $\mu_0(g)$ and atomicity is achieved using simple locking mechanisms. Nodes in g are able to contact $\mu_0(g)$ with the same greedy routing adopted by the HDHT, i.e. they will contact in order $\mu_{L-1}(g),\ \mu_{L-2}(g),\ \ldots,\ \mu_0(g)$. However, this solution relies on a single node: when the address $\min g$ changes, the memory of g remains in an inconsistent state until all its nodes discover the new $\min g$ propagated by the routing updates. We suggest that at the cost of an increased communication cost, it should be possible to realize a fault tolerant atomic distributed memory by applying the Paxos consensus protocol[24] and the ideas presented in Etna[25]. We now briefly describe what its main components would be: $\mu_0(g)$ becomes the *primary memory* of g and it serves and serialize the read/write requests. k nodes of g are elected as *replicas*, each of them fetches and maintains a copy of the primary memory. In order to ensure a uniform spatial distribution, the replicas are selected across the hierarchy with the following function:

$\mu_l(g,k)$:

If $\mathrm{lvl}(g) = 0$, **return** $\{g\}$

else g is a gnode with elements $g = \{h_1, \ldots, h_{|g|}\}$, with $h_i \leq h_{i+1}\ \forall i$

If $k < |g|$, return $\{\mu_0(h_1), \ldots, \mu_0(h_k)\}$, where $\mu_0(h_i) = \min h_i$

else let $r = k \bmod |g|$, $\quad d = k/|g|$

return $\mu(h_1, d+1) \cup \ \ldots\ \cup \mu(h_r, d+1) \cup \mu(h_{r+1}, d) \cup \ldots \cup \mu(h_{|g|}, d)$

Each time a read/write request is issued to the memory of g, the primary node verifies the memory consistency by querying the replicas with the Paxos protocol. If it receives more than $(k+1)/2$ ACKs from the replicas it will accept the request. Using a counter mechanism, in the case of concurrent writes only the most up to date write commits are accepted by the replicas. Every time the primary node

changes, i.e. $\mu_0(g)$ points to another node x, the Paxos protocol populates the primary memory by collecting from the majority of replicas the most up to date version.

Finally, we describe how we can solve our serialization problems:

1. a node x that wants to join to $g = *.g_l \dots g_{L-1}$ by taking an address $g = *.x_{l-1}.g_l \dots g_{L-1}$ will try to set to 1 the x_{l-1}-th bit of the memory of g_l. Since its memory is atomic, no other node will be able to simultaneously set to 1 the x_{l-1}-th bit, and thus join to g_l.
2. similarly as above, a bit is set to 1 until the condition Q is true. In this way, the nodes can atomically check if Q is true or not.

4.2 Gnode Split

Suppose a gnode h splits into connected components $H_1, \dots, H_m$. Then in order to satisfy the connectivity constraint of group nodes, all but one $H_1, \dots, H_m$ will have to change address.
A node $x \in H_i$ can recognize the splitting of h after the routing updates occur: there will be a missing path to reach a node of another component. Using the routing table the node x can deduce the number of nodes of its component. Also, x can know $\min H_i$, the smallest address of the nodes in H_i. If x is a border node of h, it acts as follow: if $\min H_i \neq \min h$, x leaves H_i and migrate. After completing the migration, it also tells its old neighbors in H_i to do the same. In this way, the only component that does not migrate is the one that contains $\min h$.

Notice that if H_i does not contain any border node, then H_i has been completely disconnected from the network. In this case, nodes in H_i do not have to change addresses.

We can optimize the above rule by minimizing the number of migrations, as follow: for each component $H_1, \dots, H_m$, the node $\min H_i$ sends to a rendezvous node not in h the size $|H_i|$ along with its address $\min H_i$. The rendezvous node acts as a hub and forwards each message to all the nodes $\min H_i,\ i = 1, \dots, m$. The node $\min H_i$ will in turn forward the received message to the border nodes of H_i. In this way, the border nodes know the size of all components and the new condition for migration becomes $\min H_i \neq \min H_j$, where H_j is the component such that

$$|H_j| = \max_{1 \leq i \leq m} |H_i|$$
$$\min H_j \geq \min H_k \ \ \forall k \text{ s.t. } |H_k| = |H_j|$$

In other words, H_j is the component with the maximum size and with the maximum $\min H_j$ among the components of maximum size.
It follows that the only component that will not migrate is one of those with the largest size.

4.3 Network ID and Network Merging

Up to now we have considered the whole network G as a connected network. However, for a complete distributed implementation we have to consider the general case of a disconnected network. That is, G will be formed by the union of connected networks $G_1, \ldots, G_m$. Lets consider the case where $m = 2$. Since G_1, G_2 are disconnected, the nodes in each network have no way for coordinating the choice of their addresses, thus a node in G_1 can choose the same address of a node in G_2. If later on G_1, G_2 become connected, then the resulting network $G_1 \cup G_2$ will have an address collision.
There are two ways for solving this problem:

1. Use an out of band communication between G_1 and G_2 for coordinating the address assignment.

   With this solution we are effectively creating virtual links between G_1 and G_2 and we can always assume that the original network is connected.

2. Assign a unique ID to each distinct network.

   This solution can be viewed as extending the address $y_1 \ldots y_{L-1}$ of each node to $y_1 \ldots y_{L-1}.\eta$, where η is the network ID (netid).

   All the nodes with the same netid will form a connected network. In other words, the netid can be effectively viewed as a group node of level L.

We now analyze in more details how to assign a network ID to each node.

1. Each node v chooses a random number $\eta_0(v)$, with sufficiently many bits such the probability that two nodes have the same $\eta_0(v)$ is negligible. Notice that the netid will not be used for routing purposes and thus the number of bits can be chosen more freely.

2. The network ID of a connected network G_1 is then

$$\eta(G_1) = \min_{v \in G_1} \eta_0(v)$$

The nodes of G_1 can know $\eta(G_1)$ in a single flooding round: at first, each node v sets $\eta(G_1) := \eta_0(v)$, secondly it broadcasts its current known $\eta(G_1)$ to its neighbors. After a node w receives a broadcast η, it sets $\eta(G_1) := \min\{\eta(G_1), \eta\}$. If $\eta(G_1)$ has changed, w retransmits it to its other neighbors.

3. With the above procedure, if G_1 is a connected network, then all its nodes will agree on a unique $\eta(G_1)$. Suppose now the two networks G_1 and G_2 become connected. Suppose also that the nodes know an estimate of the size of their original network. Then instead of choosing $\min\{\eta(G_1),\ \eta(G_2)\}$, the nodes will prefer the netid of the smaller network. In this way, the number of flooded nodes will be minimized. In particular, this is a necessary optimization when G_1 is formed by a single node x, i.e. when x joins the network G_2.

4. Suppose that the node $x \in G$ with the minimum netid $\eta_0(x)$, i.e. $\eta_0(x) = \eta(G)$, leaves the network. A node will participate in a new round of netid-discovery, only after it receives the routing update regarding the departure of $\eta_0(x)$.

   This rule handles the case when the network G becomes disconnected into components $G_1, \ldots, G_m$, due to a link or node failure. The nodes in the components where x is missing will change their netid, so that at the end of the netid-discovery each network G_i will have a distinct netid.

The above solution presents a drawback: when the node x with $\eta_0(x) = \eta(G)$ dies, then a new netid-discovery round will occur and the entire network will be flooded. By assuming that nodes can synchronize their clocks, we can damp this problem with the following heuristic: the node with the highest uptime will be the least likely to leave the network. To apply the heuristic, it suffices to change the definition of $\eta_0(x)$ as follow:

$$\eta_0(x) = (\text{Time when } x \text{ has been turned on},\ \text{Random Number})$$

two pairs (t, r), (t', r') will be compared using the lexicographic order.

Finally, suppose that two separated networks G_1, G_2 become linked and that G_1 is the one that will change netid. The two networks have also to resolve all the address collisions. This is possible because the nodes bridging the two networks will exchange their routing table. If they notice that two gnodes of level $L-1$ have the same address, then the one in G_1 will be alerted and its nodes will start to migrate.

Chapter 5

The Cost of Balance

The main cost associated to the balancing rules is the number of migrations that occur while the network evolves.
Each time a single node migrates, it has to advertise its new address by updating the *name→address* mapping stored in the HDHT. Moreover, if the node is storing some mappings of the HDHT, it has to transfer them to the most appropriate node. Assuming that the HDHT equally distributes the mappings and that at most a constant number of them are registered by each node, a location update requires the transfer of $O(\log N)$ stored mappings[1]. Thus if M is the total number of migrations, $O(M \log N)$ mappings will be transferred.
Also, the change of the routing address of a node affects higher layer transmission protocols: a TCP connection between a node and a migrating node will break. Thus additional countermeasures such as virtual circuits are required.
Finally, when a node x migrates from a group g to a group h, new routing updates are necessary for discovering the paths connecting x to the other nodes of h. Depending on the routing protocol implementation, it may be necessary to update the routing tables of the higher levels, for example when x affects the route stretch introduced by the groups g and h.

For all the above reasons, our ultimate aim is to understand what is the behavior of the balancing rules when the network evolves and estimate the number of

[1] the $\log N$ term derives from the extended HDHT where a same mapping is stored in $L = \log_S N$ nodes

migrations.

Remark 5.0.1. It is possible to trade the handoff overhead caused by migrations of high level gnodes with an increased communication cost for the HDHT. The read/write procedure of the HDHT becomes the following: a node $x = x_0 \dots x_{L-1}$ stores its $name \mapsto address$ mapping only to the nodes with address $h_{x,i}(\text{name}) = h_0 \dots h_i . x_{i+1} \dots x_{L-1} \quad \forall i \leq l$, where $l < L - 1$. The mapping stored at node $h_{x,i}(\text{name})$ points to the partial address of the form $x_0 \dots x_i$. In this way, x does not need to update the mapping when one of its higher gnodes $*.x_i \dots x_{L-1}$, with $i > l$, migrates. Thus, the S^{l+h} mapping updates required by the migration of a full gnode of level $l + h$ are saved. However, in order to retrieve the address of x, a node y that does not belong to $*.x_l \dots x_{L-1}$ will have to query all the possible nodes with an address of the form $h_0 \dots h_{l-1}.*$

We now give some results on the performance of the balancing rules. Then we will proceed in an experimental evaluation through simulation.

5.1 Number of Migrations

Remark 5.1.1. In the case of node removal the PB-rule induces new migrations when a gnode becomes too small in comparison with other gnodes, i.e. when the condition of Proposition 3.3.4 does not hold anymore. Instead, the LM-rule does not force any migration when the size of a gnode decreases.

We now examine what happens when new nodes are added in the network.
Proposition 5.1.2. *Suppose that all the gnodes in $[G]_l$ have the same size a, with $S - 1 \geq a \geq 1$. Suppose s nodes join to a gnode $g \in [G]_l$, where $s \leq (S - a)N$, $N = |[G]_l|$.*

The number of migrations caused by a balancing rule (PB or LM rule) is:

$$M(s) = \left[\frac{s}{D}\right] \sum_{j=1}^{m} (j-1)D_j + \sum_{j=1}^{m'} (j-1)D_j + m'r'$$

where

$$m' = \max\left\{ m' \mid (s \bmod D) - \sum_{k=1}^{m'} D_i \geq 0 \right\}, \quad r' = (s \bmod D) - \sum_{k=1}^{m'} D_k,$$

$$D = \sum_{i=1}^{m} D_i = S_0 N, \quad N = |T(g)| = |[G]_l|$$

$$m = \max_{y \in [G]_l} d(g, y), \qquad \textit{where } d(x, y) \textit{ is the hop distance}$$

$$D_i = S_0 E_i, \quad E_i = |\{h \in [G]_l \mid d(g, h) = i\}|$$

$$S_0 = \begin{cases} 1 & \textit{PB-rule} \\ S - a & \textit{LM-rule} \end{cases}$$

For the LM-rule the formula becomes simpler:

$$M_{LM}(s) = \sum_{j=1}^{m'} (j-1) D_j + m' r'$$

$$m' = \max\left\{ m' \mid s - \sum_{k=1}^{m'} D_i \geq 0 \right\}, \quad r' = s - \sum_{k=1}^{m'} D_k$$

$$s \leq (S - a) N$$

Notice that the LM formula does not depend on m, while the PB formula does.

Proof: Let's fix $S_0 = 1$ and consider the PB-rule first.
By Rem. [3.3.2,pg.24], each time one node is added in g, a shortest migration path is selected, i.e. for each addition $1 \leq i \leq s$ we have a path

$$P^i = (p_1^i, \ \ldots, \ p_{m_i}^i)$$
$$\text{length}(P^i) = m_i$$
$$p_1^i = g \ \ \forall i$$

We prove by induction on i that if $i \leq D$, then

1. $i > 1 \Rightarrow m_{i-1} \leq m_i \leq m_{i-1} + 1$
2.
   - 2.0 $p_1^i = g$
   - 2.1 $|p_j^i| = a + 1 \;\; \forall 2 \leq j \leq m_i - 1$
   - 2.2 after the migrations of P^i, we have $|g| = a + 1$
3. *Before* the i-th node is added, we have
   - 3.0 $|p_{m_i}^i| = a$

   after, we have:
   - 3.1 $|p_{m_i}^i| = a + 1$
   - 3.2 $f_i : \left\{P^k \mid k \leq i\right\} \longrightarrow \{h \in [G]_l \mid |h| = a + 1\}$ is a bijection: $P^k = (p_1^k, \ldots, p_{m_k}^k) \mapsto p_{m_k}^k$
   - 3.3 $i = |\left\{P^k \mid k \leq i\right\}| = |\{h \in [G]_l \mid |h| = a + 1\}|$
4. $M^i = m_i - 1$ is the number of migrations caused by the addition of the i-th node

If $i = 1$, the new size of g is $a+1$. The size of all the other gnodes is still a. Thus, $P^1 = (g,)$ and $m_1 = 1,\ M^1 = 0$. Also, 3. is true.
Consider the case $i + 1$. Let P^{i+1} be any shortest migration path. By induction, before adding the new $i + 1$-th node, the size of a gnode is either a or $a + 1$ and $|g| = a + 1$.
By definition of migration path, we have:

$$\begin{aligned} & p_1^{i+1} = g, \;\; |g| = a + 2 \\ & |p_j^{i+1}| = a + 1 \;\; \forall j \leq m_{i+1} - 1 \\ & |p_{m_{i+1}}^{i+1}| = a \qquad\qquad (0) \end{aligned}$$

Such a P^{i+1} exists: if, by contradiction, all the nodes have size $a+1$, then each of them has been already reached by a migration path, i.e. by 3. we have $i = |[G]_l|$, but

$$D = \sum_{i=1}^{m} D_i = |[G]_l|$$

$$i + 1 \underbrace{\leq}_{\text{by hypothesis}} D \Rightarrow i \leq |[G]_l| - 1$$

which contradicts $i = |[G]_l|$.
By $2., 3.0, 3.1$ and (0), the gnode $p_{m_{i+1}}^{i+1}$ cannot be any of the gnodes reached by previous migration paths, so

$$|\left\{P^k \mid k \leq i + 1\right\}| = |\left\{P^k \mid k \leq i\right\}| + 1 = i + 1$$

After adding the new node, the migrations occur and we have:

$$p_1^{i+1} = g, \;\; |g| = a+1$$
$$|p_j^{i+1}| = a+1 \;\; \forall j \leq m_{i+1} - 1$$
$$|p_{m_{i+1}}^{i+1}| = a+1$$

At least, $m_{i+1} - 1$ migrations have occurred to let one node migrate from p_j^{i+1} to $p_{j+1}^{i+1} \;\; \forall j \leq m_{i+1} - 1$. Finally, by Prop. 3.2.11, these are the only migrations that have occurred, thus no other gnode apart from $p_{m_{i+1}}^{i+1}$ has increased its size. Thus, $2., 3., 4.$ are true. We now prove that also 1. is true,

$|p_{m_{i+1}-1}^{i+1}| = a+1 \underbrace{\Rightarrow}_{3.2} \exists!\;\; P^j : \; p_{m_j}^j = p_{m_{i+1}-1}^{i+1}, \;\; \text{with } j \leq i$

P^{i+1} is a shortest migr. path $\Rightarrow m_j \geq m_{i+1} - 1$

$m_i \underbrace{\geq}_{j \leq i, \text{ inductive Hp}} m_j \geq m_{i+1} - 1 \Rightarrow m_i + 1 \geq m_{i+1}$

It is impossible that $m_{i+1} < m_i$, otherwise P^i is not a shortest migr. path $\Rightarrow$

$\Rightarrow m_i \leq m_{i+1}$

If $D \geq i \geq \sum_{j=1}^{q} D_j$ the number of migration paths of length $j \leq q$ is equal to D_j (1):

let f be the restriction of f_i to $\left\{P^k \mid k \leq i, \;\; m_k = j\right\}$

$f : \left\{P^k \mid k \leq i, \;\; m_k = j\right\} \longrightarrow D_j$ is a bijection:

$|g| = a+1, \;\; \forall h \in [G]_l \;\; a \leq |h| \leq a+1$

P^k is a *shortest* migration path

$\Rightarrow P^k$ is a *shortest* path $\Rightarrow \; f(P^k) = p_{m_k}^k \in D_{m_k} = D_j$

f is injective $\Rightarrow$ f injective

from $i \geq \sum_{j=1}^{q} D_j$ and $1., 3.3,$ it follows that f is surjective

By 4., the number of migrations after the s-th node has been added is:

$$M(s) = \sum_{i=1}^{s} (m_i - 1) \;\; (5)$$

If $s \leq D$, with (1) we can collect the terms in (5) by length:

$$M(s) = \sum_{j=1}^{m'} (j-1) D_j + ((m'+1) - 1) r'$$

$$\text{where } m' = \max \left\{ m' \leq m \mid s - \sum_{k=1}^{m'} D_i \geq 0 \right\}, \quad r' = s - \sum_{k=1}^{m'} D_k$$

$\sum_{j=1}^{m'} (j-1) D_j$ are the migrations required to increase the size of all the gnodes

reachable in at most m' hops. Each time we add a node in one of these gnodes, we are using a path of length j and thus the required migrations are $j-1$. r' is the number of gnodes reachable in $m'+1$ hops that we can still enlarge.
Finally, notice that if $s = D = |[G]_l|$,

$$s = D = \sum_{k=1}^{m} D_k$$
$$m' = m = \max \text{length}(\text{Path}([G]_l)), \quad r' = 0$$
$$M(s) = \sum_{j=1}^{m} (j-1) D_j$$

and, after the migrations, all the gnodes have size $a+1$. Thus, the same initial situation has been reached again. In the case of any new addition, we can reapply the same reasoning. So, if $s = qD$ is a multiple of D:

$$M(s) = q \sum_{j=1}^{m} (j-1) D_j$$

In general, considering the remainder r of s/D, we have:

$$M(s) = \left[\frac{s}{D}\right] \sum_{j=1}^{m} (j-1) D_j + \sum_{j=1}^{m'} (j-1) D_j + ((m'+1)-1) r'$$
$$\text{where } m' = \max \left\{ m' \mid r - \sum_{k=1}^{m'} D_i \geq 0 \right\}$$
$$r' = r - \sum_{k=1}^{m'} D_k$$

The formula for the LM-rule, can be derived with a similar reasoning. The main difference is that we can think of adding $S-a$ nodes each time. Another way to derive it is to notice that the LM-rule is equivalent to an application of the PB-rule on the graph obtained from $[G]_l$ by substituting a gnode $h \neq g$ with $S-a$ identical copies (i.e. nodes with the same edges of h). In the new graph, E_i^{new} is equal to $(S-a)E_i^{\text{old}} \ \ \forall i = 1, \ldots, m$.
Finally, the simplification of the LM formula is derived as follow: we cannot add more than $(S-a)|[G]_l| = (S-a)N = D$ nodes, thus we have $s \leq D$, and

$$[s/D] = 0, \quad r = s$$
$$M(s) = \sum_{j=1}^{m'} (j-1) D_j + m' r'$$
$$m' = \max \left\{ m' \mid s - \sum_{k=1}^{m'} D_i \geq 0 \right\}, \quad r' = s - \sum_{k=1}^{m'} D_k$$

□

Corollary 5.1.3. *Consider two different address assignments such that* $[G]_l$ *and* $[\overline{G}]_l$ *are two different graphs. Suppose also that all the gnodes, both in* $[G]_l$ *and in* $[\overline{G}]_l$*, have the same number of nodes* a*, with* $S - 1 \geq a \geq 1$*. As in Prop. 5.1.2, add* s *nodes in a gnode* $g \in [G]_l$ *and in a gnode* $\overline{g} \in [\overline{G}]_l$*. Let* $M(s)$ *be the number of migrations induced in* $[G]_l$ *and* $\overline{M}(s)$ *those in* $[\overline{G}]_l$*.*
If in $[\overline{G}]_l$ *there are more shortest paths than in* $[G]_l$*, we have* $\overline{M}(s) \leq M(s)$*. More precisely, using the same notations of Prop. 5.1.2, suppose*

$$\begin{aligned} &\overline{E}_i = E_i + h_i \ \ \forall i = 1, \ldots, \max\{m, \overline{m}\} \\ &h_i \in \mathbb{Z}, \ \ \sum_{i=1}^{j} h_i \geq 0 \quad \forall j \leq \max\{m, \overline{m}\} \\ &N = \overline{N} \end{aligned}$$

then

$$\overline{M}(s) \leq M(s)$$

In other words, if $[\overline{G}]_l$ *is more connected than* $[G]_l$*, it will induce less migrations. The intuitive explanation is that the migration paths in* $[\overline{G}]_l$ *are shorter and cause less migrations than the equivalent paths in* $[G]_l$*.*

Proof: Recall from the previous proof that if $s \leq N$,

$$M(s) = \sum_{i=1}^{s} (m_i - 1)$$

$$\text{length}(P^i) = m_i \leq m_{i+1} \leq m_i + 1 \ \ \forall i = 1, \ldots, s-1$$

Since $s \leq N = \overline{N}$, the same is true for $\overline{M}$ and $\overline{m}_i$.
By induction on i, we prove that $m_i \geq \overline{m_i} \ \ \forall i = 1, \ldots, N$. This suffices to show that $\overline{M}(s) \leq M(s)$, also when $s > N$.
Base case:

$$P^1 = (g,), \ \overline{P}^1 = (\overline{g},) \ \Rightarrow \ m_1 = 1 = \overline{m_1}$$

Suppose $m_i \geq \overline{m_i}$ is true. Since $\overline{m_i} \leq \overline{m_{i+1}} \leq \overline{m_i} + 1$ we have two cases
CASE: $\overline{m_{i+1}} = \overline{m_i}$

$$\overline{m_{i+1}} = \overline{m_i} \leq m_i \leq m_{i+1}$$

CASE: $\overline{m_{i+1}} = \overline{m_i} + 1$

Since $1 = \overline{m_1} \leq \overline{m_2} \leq \cdots \leq \overline{m_{\overline{N}}}$ and $\overline{m_{i+1}} \leq \overline{m}_i + 1$, we have

$$i = \sum_{j=1}^{\overline{m_i}-1} \overline{\nu}_j + \overline{\rho}(\overline{m_i}, i)$$

$$\text{where } \overline{\nu}_j = \Big| \big\{k \leq \overline{N} \mid \overline{m_k} = j\big\} \Big|, \quad \overline{\rho}(q,i) = \Big| \big\{k \leq i \mid \overline{m_k} = q\big\} \Big|$$

and analogously,

$$i = \sum_{j=1}^{m_i-1} \nu_j + \rho(m_i, i)$$

$$\text{where } \nu_j = \Big| \big\{k \leq N \mid m_k = j\big\} \Big|, \quad \rho(q,i) = \Big| \big\{k \leq i \mid m_k = q\big\} \Big|$$

Notice that

$$\nu_{m_i} \geq \rho(m_i, i)$$

Recall that the number of all migration paths of length j is equal to the number of gnodes of $[G]_l$ reachable in j hops:

$$\nu_j = E_j$$

Analogously for $\overline{G}$:

$$\overline{\nu}_j = \overline{E}_j = E_j + h_j = \nu_j + h_j$$

We are under the hypothesis $\overline{m_{i+1}} = \overline{m_i} + 1$, thus

$$i+1 = \sum_{j=1}^{\overline{m_{i+1}}-1} \overline{\nu}_j + \overline{\rho}(\overline{m_{i+1}}, i+1) = \sum_{j=1}^{\overline{m_i}} \overline{\nu}_j + \overline{\rho}(\overline{m_{i+1}}, i+1) \underbrace{>}_{\overline{\rho} \geq 1} \sum_{j=1}^{\overline{m_i}} \overline{\nu}_j =$$

$$= \sum_{j=1}^{\overline{m_i}} \nu_j + \underbrace{\sum_{j=1}^{\overline{m_i}} h_j}_{\geq 0} \geq \sum_{j=1}^{\overline{m_i}} \nu_j \quad (*)$$

$m_{i+1} \geq m_i \geq \overline{m_i}$, and it's impossible that $m_{i+1} = \overline{m_i}$ in fact:

$$i+1 \underbrace{>}_{(*)} \sum_{j=1}^{\overline{m_i}} \nu_j \underbrace{=}_{m_{i+1} = \overline{m_i}} \sum_{j=1}^{m_{i+1}} \nu_j \geq \sum_{j=1}^{m_{i+1}-1} \nu_j + \rho(m_{i+1}, i+1) = i+1$$

$i+1 > i+1$ contradiction

thus $m_{i+1} \geq \overline{m_i} + 1 = \overline{m_{i+1}}$

□

Corollary 5.1.4. *When adding s nodes in $[G]_l$, the maximum number of migrations is reached when $[G]_l$ is a linear path and all the nodes are added at one of the two extremes. This is true both for the LM-rule and for the PB-rule.*

Proof: Corollary 5.1.3 □

Proposition 5.1.5. *Under the hypotheses of Prop. 5.1.2, suppose that $[G]_l$ is a linear path and g is one of the two extremes. Then, the number of migrations*

caused by the PB rule is:

$$M(s) = \left[\frac{s}{N}\right] \frac{N(N-1)}{2} + \frac{r(r-1)}{2}$$
$$r = s \bmod N$$

and by the LM rule is:

$$M_{LM}(s) = S_0 \frac{m'(m'-1)}{2} + m'r'$$
$$m' = \left[\frac{s}{S_0}\right], \quad r' = s \bmod S_0$$
$$S_0 = S - a$$

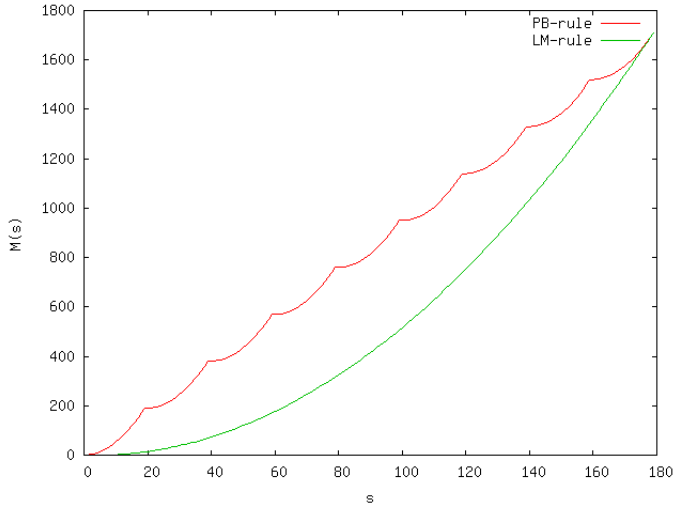

Figure 5.1: Number of migrations of the LM-rule and the PB-rule when $[G]_l$ is a linear path (Proposition 5.1.5). The parameters are $N = 20, S = 10, a = 1$.

Proof: This follows directly from Prop. 5.1.2, because $[G]_l$ is a linear path of N nodes and we have

$$m = N$$
$$D_i = S_0 \cdot 1 \quad \forall i$$
$$D = S_0 N$$

In the case of the PB-rule we have $S_0 = 1$ and

$$m' = \max\{m' \mid r - m' \geq 0\} = r, \quad r' = r - r = 0$$

Instead, for the LM-rule,

$$m' = \max\{m' \mid s - m'S_0 \geq 0\} = \left[\frac{s}{S_0}\right], \quad r' = s - m'S_0 = (s \bmod S_0)$$

$$S_0 = S - a$$

□

5.2 Simulation

The scenario of the simulation is based on a city wireless mesh network where each node is located in a house and the density allows the coverage of the whole city, i.e. the resulting network graph is connected.

Random Geometric Graphs (RGG) are the simplest model of wireless mesh networks. In a RGG the nodes are uniformly distributed in the unit square $[0,1]^2$ and a link is established between two nodes $\overline{x}, \overline{y}$ if $d(\overline{x}, \overline{y}) \leq r_0$.
However, the geometric topology of a city wireless network is very different from the unit square. If a node places its antenna behind a window, then at least half of its coverage is shadowed by the building. Also, the height of buildings may vary and antennas placed on the roofs may be shadowed by taller buildings.
In order to simplify the matter, while still taking into account the obstacles of a city wireless network, we will assume that the links of a RGG are removed with a fixed probability $1-p_c$. We will call the resulting graph a City Random Geometric Graph (CRGG).
If $p_c = 1$, a CRGG is still a RGG, instead if $p_c < 1$, we can approximate it with a classic random graph where an edge is established between two nodes with a fixed probability p. For this reason, in our simulations we will resort to random graphs only.

Proposition 5.2.1. *Let G be a CRGG with radius r_0 and link probability p_c, then*

1. *If $p_c = 1$, G is a RGG*
2. *If $p_c < 1$ and we ignore nodes near the border of the unit square, G is a random graph with link probability $p_c \pi r_0^2$*

Proof:
$P(xy \text{ are linked }) = P(xy \mid d(x,y) \leq r_0) P(d(x,y) \leq r_0) = p_c \operatorname{Area}(I_{r_0}(x)) = p_c \pi r_0^2$
□

Note: a further generalization of CRGGs are the Random Distance Graphs studied in [28].

Notice that the assumption of having a constant probability of link establishment

is justified by the fact that wireless links between fixed nodes can become broken only due to new obstacles and in a city we can consider this an infrequent event. As a consequence, once the network has been formed, nodes addition and removal will be the only network events occurring in the network. In other words, at a given time a node can be either *On* or *Off*:

$$\lambda_x(t) = \begin{cases} 1 & \text{node } x \text{ is On at time } t \\ 0 & \text{else} \end{cases}$$

Under the above assumptions, we performed two different experiments.

5.2.1 First Experiment

For comparing various properties of the balancing rules we considered a simple network evolution that induce all kinds of migrations, namely migrations caused by splits, by network merging and by new joining nodes.

In details, the first experiment runs as follow:

1. A connected random graph G is generated with link probability p and $N_{\max}$ nodes. We ensure that G is connected by repeatedly generating p-random graphs with the NetworkX Python library[30] until a connected graph is found. A connected random graph will be found quickly if p is greater than the critical threshold $1/N$ of random graphs. In fact, with high probability the size of the giant component of a random graph increases exponentially as $p > 1/N$ increases and it becomes the full graph when $p \geq \ln(N)/N$ [27], [29].
2. Only the first level of the hierarchy is considered and the maximum number of level 1 gnodes is set to ceil$(N_{\max}/S)$, where S is the gnode size.
3. At the start of the simulations, all the nodes are turned Off.
4. Sequentially, each node is turned On, the balancing rules are executed and the number of migrations is counted.
5. When the number of On-nodes is $N_{\max}$, i.e. all the nodes joined the network, we start the turn-Off phase: at each step we turn Off a node and count the number of migrations.

In figure 5.2, we can see how a simulation of the first experiment evolves.
The network starts with 0 nodes, at each step a new node is added. On the Y

axis is reported the total number of migrations. When at step 1000 the network becomes full, the simulation proceeds by removing a node at each step.

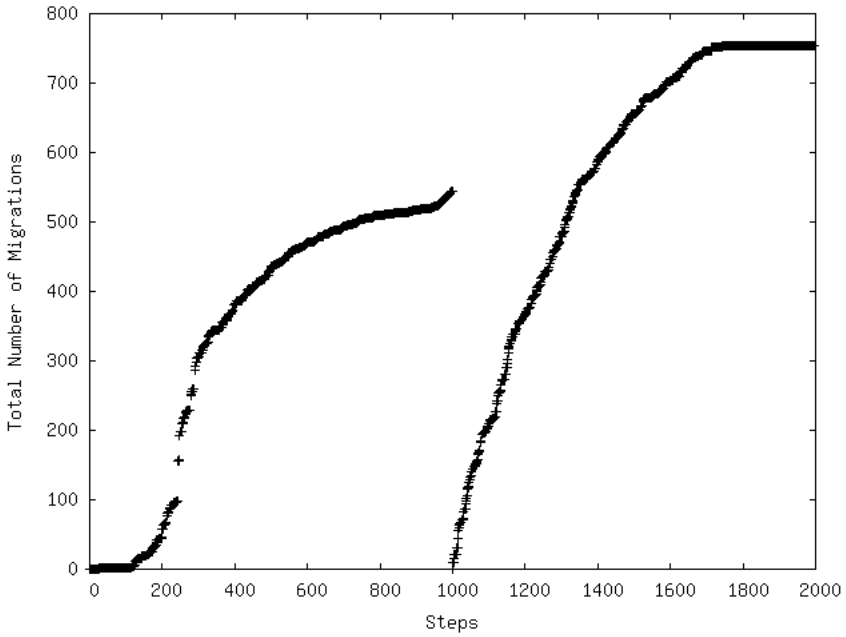

Figure 5.2: Evolution of the first experiment, with parameters $N_{\max} = 1000$, $N_{\min} = 1$, $p = 6/N_{\max}$, $S = 20$.

Migrations and Link Probability

In another simulation we varied the probability of link formation $p = d/N_{\max}$, where d is the average degree of the graph. As shown in the figure 5.3, as p increases, the number of migrations decreases. This is not surprising: as p increases, nodes are more connected. A joining node is then linked to a greater number of groups and it has a greater chance of finding a non-full group as a neighbor. Thus, less migrations occur because the LM-rule enforces a migration only when a node as no other choice than to join into a full group.
Secondly, since it becomes more difficult to disconnect a group with a node removal, the number of migrations due to a gnode split decreases too.

For these reasons, in all successive simulations we preferred to study networks with low connectivity, by choosing $2/N \leq p \leq \ln(N)/N$.

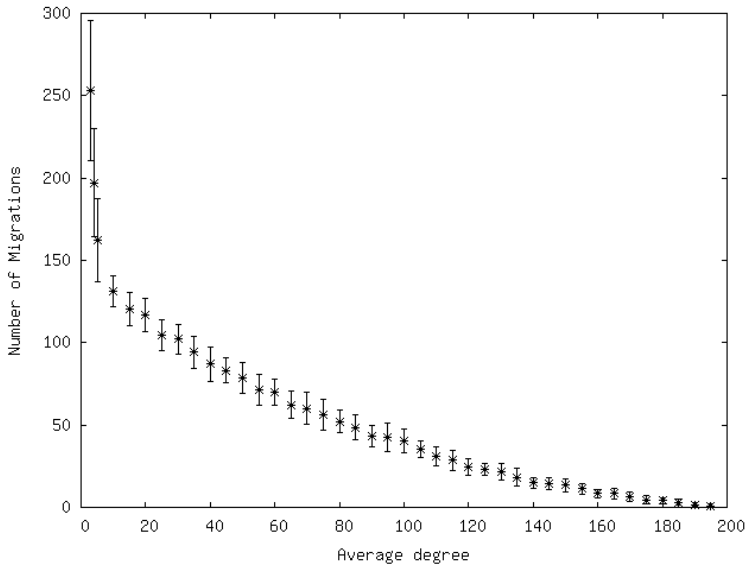

Figure 5.3: The number of migrations as a function of the probability of link formation of the generated random graphs. The parameters for this simulation are $N_{\max} = 100,\ S = 6,\ N_{\min} = 1$.

PB and LM Rules Comparison

For comparing the PB-rule against the LM-rule, we repeatedly run the first experiment with parameters in the ranges $80 \leq N_{\max} \leq 100,\ 3/N_{\max} \leq p \leq 4/N_{\max}$, $6 \leq S \leq 20$.

For each generated graph, we used both the PB-rule and the LM-rule. In all simulations, the number of migrations caused by the LM-rule (M_{LM}) was never greater or equal than those caused by the PB-rule (M_{PB}) and on average, M_{LM} was $45 \pm 11\%$ less than M_{PB}.

These results show that the LM-rule is more efficient than the PB-rule. Note: in all subsequents simulations we used the LM-rule only .

Aggregative and Dispersive Rules Comparison

Similarly as before, we compared the Aggregative-rule (A-rule) with the Dispersive-rule (D-rule) (see section 3.3.7). As we expected, the A-rule is more costly. On average, the D-rule generates $37 \pm 24\%$ less migrations of the A-rule and only in 8% of all simulations, the D-rule caused more migrations than the A-rule.

Group Size

The maximum size S of a group node is a tradeoff parameter between the size of the routing table and the latency stretch, but it also influences the number of migrations M.
From the left figure 5.4, we can observe that when the size of a group becomes larger than half of the number of nodes (100), the migrations begin to decrease. In fact, it becomes increasingly difficult to disconnect a group due to a node removal (we will explain why in the next section). Also, the migration paths decrease in length.

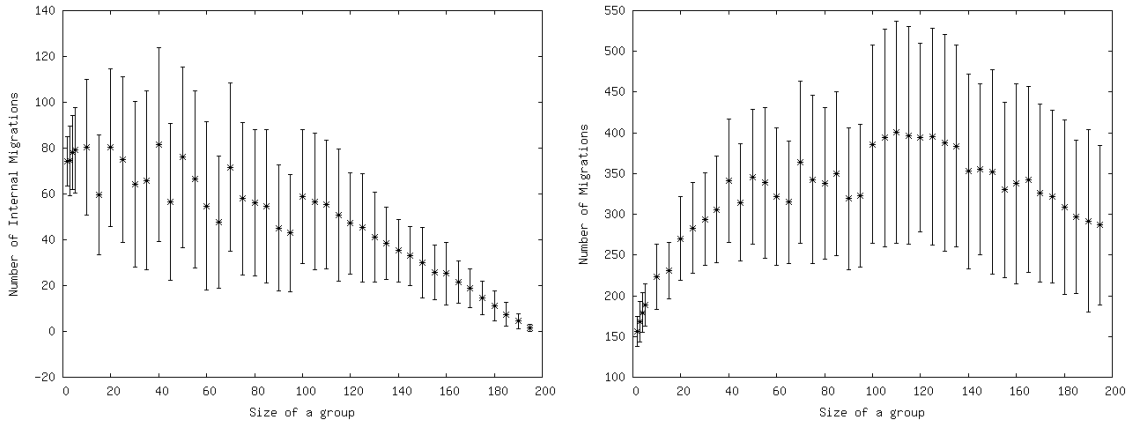

Figure 5.4: On the left, the plot represents the number of migrations as a function of the group size. On the right, the same data is reported, but the migrations due to network merging are also counted. The parameters for this simulation are: $N_{\max} = 200,\ N_{\min} = 1,\ p = 4/N_{\max}$.

However, if we also take into account the migrations due to network merging (Chapter 4.3), M reaches a minimum when the group size is 2 and then increases until a point where it becomes almost constant (right figure 5.4).

Small values of S have a drawback: the number of established virtual addresses becomes higher when S is small. This is shown in figure 5.5.

Groups are Random Graphs

We repeated the simulation for 5000 times by varying the group size S and counted how many edges E were contained in a full group node at the end of the network formation. In figure 5.6, we can see the edge ratio $p' = \frac{E}{\binom{S}{2}}$ plotted as a function

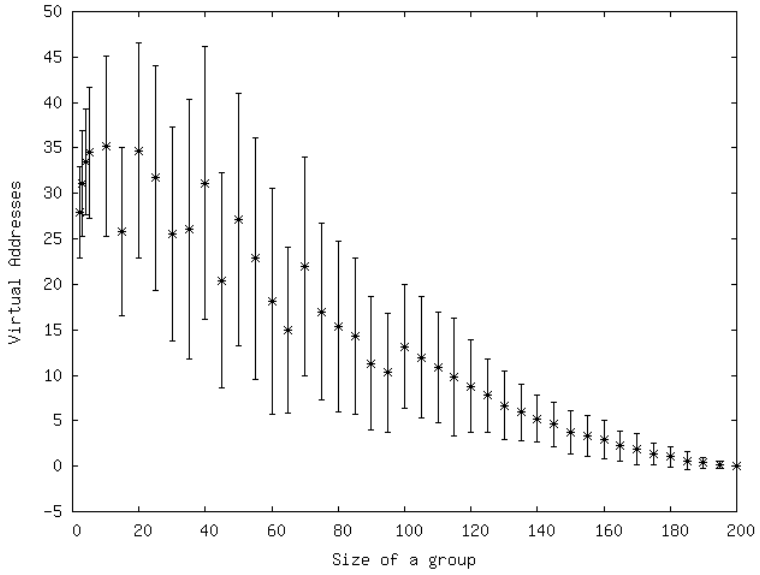

Figure 5.5: The number of established virtual addresses depends on the size of the group nodes. Smaller sizes force the creation of more virtual addresses.

of S. As S becomes larger, p' tends to the probability p of link formation of the network graph. For $S > 20$, the relative error on all points is $\leq 1.5\%$, while on 70% of points it is $\leq 0.25\%$.

Since the error is small, p' depends mainly on the size S and not on the underlying process that generates the groups. For this reason, we can consider large groups of size S as random graphs $G_{S,p'(S)}$.

We also notice that for all points $p' \geq p$, $p'S \leq pN$ (except 5 points) and $p'S$ has an increasing behavior as S grows. This can explain why larger groups are harder to split: when h nodes are removed with a fixed probability from a random graph $G_{S,p'}$, the remaining nodes form a $G_{S-h,p'}$ graph and the expected size of its giant connected component monotonically depends on its average degree $p'(S-h)$.

Virtual Addresses and Average Path Length

By considering all the data acquired from simulations, we counted how many migrations force the creation of virtual addresses: of $157 \cdot 10^6$ simulated migrations only 0.87% established new virtual addresses. We also observed that the average migration path length is short: on average it is 2.00 ± 0.03 hops long. This is not surprising: the average path length in random graphs is $\sim \ln N$, thus only

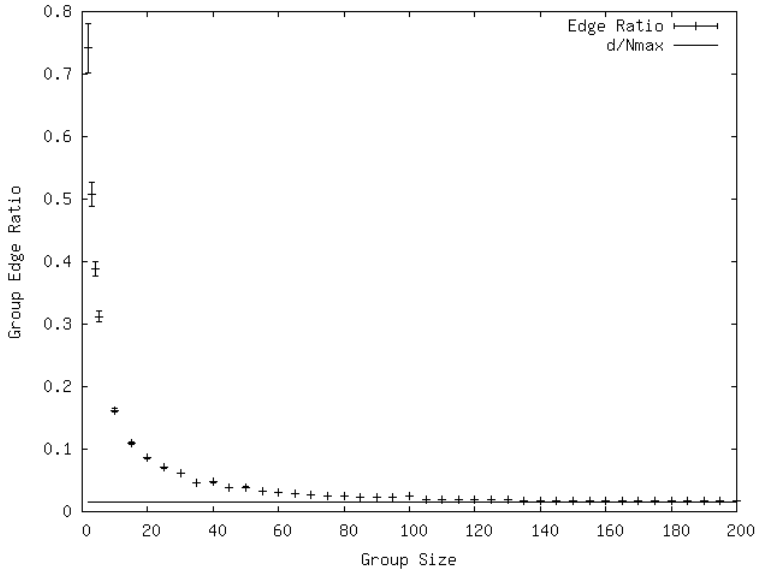

Figure 5.6: Each point indicates the edge ratio of a full group with size S. The parameters of the network graph are $N_{\max} = 200$, $p = 3/N_{\max}$. The bottom constant line is $y = p$.

with large scale simulations it should be possible to notice higher migration path lengths.

5.2.2 Second Experiment

In the second experiment, we studied the behavior of the balancing rules as the network becomes more and more chaotic. $N_{\min}$ nodes are considered *stable*, while the other $N_{\max} - N_{\min}$ nodes are marked as *churn* nodes. During the simulation the churn nodes are turned on and off with a fixed probability and the migrations are counted. In details,

1. A connected random graph G is generated as in the first experiment.
2. $N_{\min}$ nodes are turned on and the balancing rules are applied until the network reaches a stable configuration. The migrations of this step are not counted.
3. For *MaxSteps* times the following procedure is repeated: each churn node is turned off with probability $1 - p_c$ or on with probability p_c. The LM-rule is applied and the number of migrations are counted.

$N_{\min}$ and p_c affect directly the number of migrations. As shown in figure 5.7,

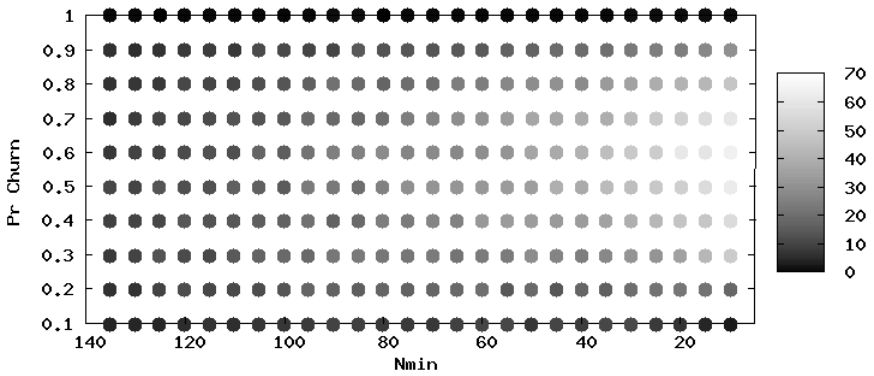

Figure 5.7: At each simulation step, each of the $N_{\max} - N_{\min}$ churn nodes is turned on and off with probability *Pr Churn*. The gray intensity indicates the average number of migrations per step (Migrations / MaxSteps) obtained at the end of the simulation. The parameters used for this simulation are: $N_{\max} = 200, S = 20, p = 4/N_{\max}$.

the migrations increase when p_c gets near $1/2$ and $N_{\min}$ decreases. In fact, $N_{\min}$ determines the percentage of churn nodes in the network. Also, the probability of having k alive churn nodes at each step follows a binomial distribution and the expected number of churn nodes that will change their status from one step to the next is:

$$\mathbb{E}(\Delta_t) = \mathbb{E}\left(\sum_{x=1}^{N_{\max}-N_{\min}} I_{xt}\right) = 2(N_{\max} - N_{\min})p_c(1-p_c)$$

where $I_{xt} = |\lambda_x^{t+1} - \lambda_x^t|$,

$\lambda_x^t = 1 \Leftrightarrow$ node x is On at time t

$$P(I_{xt} = 1) = P(\lambda_x^t = 1, \lambda_x^{t+1} = 0) + P(\lambda_x^t = 0, \lambda_x^{t+1} = 1) = 2p_c(1-p_c)$$

Thus at $p_c = 1/2$ and $N_{\min} = 0$, there is the maximum expected number of On/Off switches.

M as a function of $N_{\max}$

We repeated the experiment by varying the number of nodes $N_{\max}$ and by setting $N_{\min} = N_{\max}/k$, for a fixed $k > 0$. Notice that in this way the ratio of churn nodes is equal to $(N_{\max} - N_{\min})/N_{\max} = 1 - 1/k$. As shown in figure 5.8, M

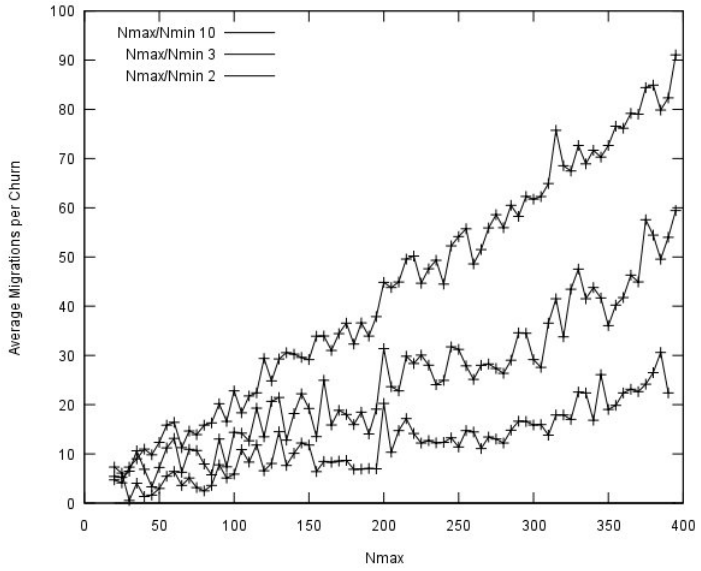

Figure 5.8: Each point represents the average number of migrations M per step of a network with $N_{\max}$ nodes. Each line is obtained by fixing a different $N_{\max}/N_{\min}$ ratio. The parameters used for the simulation are: $p = (\ln(N_{\max}) - 1)/N_{\max}$, $S = 20$, MaxSteps $= 100$,

increases almost linearly as $N_{\max}$ grows. This is also evident from their correlation coefficient (see table below), which is almost 1.

$k = N_{\max}/N_{\min}$	2	3	4	5	6	7	8	9	10
Correlation Coefficient	0.90	0.94	0.97	0.97	0.98	0.99	0.99	0.99	0.99
Linear Best Fit Error	3.01	4.68	4.68	4.45	4.00	3.23	3.16	3.06	2.93

M as a function of $\mathbb{E}(\Delta_t)$

In a next simulation, we calculated the average number of migrations M as a function of the expected number of network changes $\mathbb{E}(\Delta_t)$. By choosing $p_c = 1/2$, $\mathbb{E}(\Delta_t)$ becomes $(N_{\max} - N_{\min})/2$ and we can vary it linearly by changing $N_{\min}$.

In the figure 5.9, we can observe that M increases sublinearly as the number of stable nodes decreases. In other words, the addition of a new churn node influences little the network.

When $N_{\min} > 240$ ($< 20\%$ of churn nodes), the standard deviation of each data point is ≤ 2. In this case, M depends lightly on the underlying graph and on the

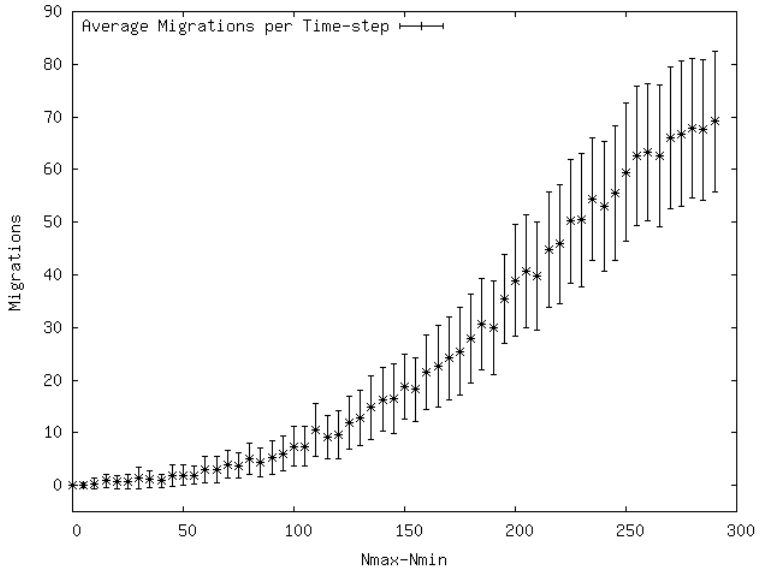

Figure 5.9: Average number of migrations as a function of the expected number of network changes $\mathbb{E}(\Delta_t)$. The parameters used for the simulation are: $N_{\max} = 300,\ S = 20,\ p_c = 1/2,\ \text{MaxSteps} = 100$

particular nodes that switch their On/Off status. Instead, when the percentage of churn nodes grows, the variance of M grows too. In other words, with high churn the peculiar features of the underlying graph begin to matter more.

Gnode Split Migrations

There are two types of migrations: those caused by new nodes joining the network and those caused by gnode splits induced by old nodes that leave the network. Intuitively, the latter type is more costly: the number of migrations caused by a new node is equal to the length of a migration path, which on average is –as we have seen– rather short. Instead, a gnode split can force the migration of a large part of a group.
In a next simulation, we counted the number of migrations occurring due to a gnode split (M_S). In figure 5.10, we can see that when $N_{\min} \leq 250$ ($\geq 17\%$ of churn nodes), more than 60% of migrations are due to gnode splits only. When $N_{\min} \geq 200$, even though the average percentage M_S/M decreases, the variance increases notably.

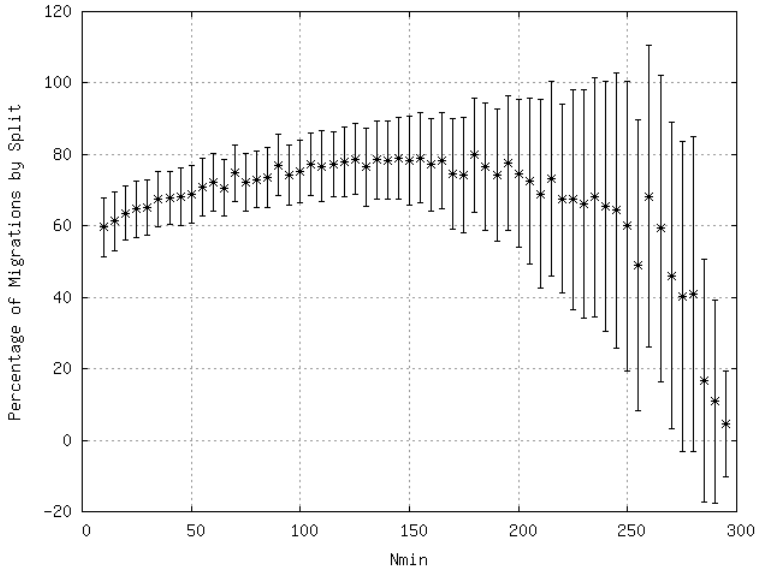

Figure 5.10: The Y axis represents the percentage of migrations due to gnode splits: $\frac{M_S}{M}100$, where M is the number of migrations per step. The parameters used for the simulation are: $N_{\max} = 300,\ S = 20,\ p_c = 1/2,\ \text{MaxSteps} = 100$

5.3 Bounds on the Number of Migrations

In the previous simulations, we have considered a network subdivided in ceil(N/S) groups of single nodes, ignoring the borders formed by levels higher than 1. This gives an under-estimate of the actual number of migrations:

Proposition 5.3.1. *Let M be the total number of migrations occurring in a network after a sequence of events, using either the PB-rule or the LM-rule. Under the same events, consider an application of the chosen balancing rule to $[G]_1$ only, or in other words let only single nodes migrate. Let M' be the relative number of migrations.*
We have:

$$M' \leq M$$

Proof: We omit the proof for space constraints. The main idea is to notice that, without the higher levels, nodes have more freedom for creating group nodes and there are no higher gnodes can become split. □

Proposition 5.3.2. *Consider a hierarchical network G with L levels, groups of size S and $N_{\max} = S^L$. Let M be the number of migrations caused by some network events occurred at the same instant.*

M is upper bounded as follow:

$$M \leq 2\delta L N_{\max}$$

where δ is the diameter of the network after the events.

If G is a random graph $G_{N_{\max},p}$ and $p > 1/N_{\max}$, on average we have

$$M \leq 2\frac{\ln^2(N_{\max})}{\ln(pN_{\max})\ln(S)}N_{\max}$$

Additionally, if $pN_{\max} \geq e$ and $S \geq e$, we have:

$$M = O(N_{\max}\ln^2(N_{\max}))$$

Proof: The balancing rules are applied at all levels, thus

$$M = \sum_{i=1}^{L-1} M_i$$

$$M_i = M_{J,i} + M_{S,i}$$

where M_i is the number of single nodes that migrate for balancing $[G]_i$, $M_{J,i}$ are the migrations caused by new nodes joining the network, while $M_{S,i}$ are those caused by gnode splits.
Recall that for each migration of a node a migration-path P is established and the number of migrations is $\text{length}(P) - 1$.
If P is a path in $[G]_i$, with $i \geq 1$, its length is upper bounded by the diameter of $[G]_i$. Also, $[G]_i$ can be viewed as a contraction of the original graph G, so

$$\text{length}(P) \leq \delta([G]_i) \leq \delta(G) \qquad (1)$$

If at level i there is still a non-full gnode g ($\text{lvl}(g) = i$, $|g| < S$), a new joining single node will cause a migration path in $[G]_i$. When $[G]_i$ becomes full, i.e. $|[G]_i| = S^{L-i}$, any further addition of nodes will cause migration paths in lower levels only. Thus, at each level i at most S^{L-i} migration paths can be established. Each migrating group has at most S^i single nodes, thus by (1) we have

$$M_{J,i} \leq \delta S^{L-i} S^i = \delta N_{\max} \qquad \forall 1 \leq i \leq L-1$$

$$M_J = \sum_{i=1}^{L-1} M_{J,i} \leq (L-1)\delta N_{\max}$$

When a gnode $g \in [G]_i$ is split all the single nodes not belonging to the largest component will migrate. The maximum number of single nodes of g is S^i. In the

worst case, $[G]_i$ is full and all its groups are split, thus

$$M_{S,i} \leq S^i |[G]_i| \delta \leq S^i S^{L-i} \delta = \delta N_{\max} \qquad \forall 1 \leq i \leq L-1$$

$$M_S = \sum_{i=1}^{L-1} M_{S,i} \leq (L-1)\delta N_{\max}$$

In sum we have:

$$M = M_J + M_S \leq 2\delta L N_{\max}$$

Finally, the expected diameter of a random graph $G_{N_{\max},p}$ is[29]:

$$\delta_{N_{\max},p} = \frac{\ln(N_{\max})}{\ln(pN_{\max})}$$

and since $L = \ln(N_{\max})/\ln(S)$, we have

$$M \leq 2\frac{\ln^2(N_{\max})}{\ln(pN_{\max})\ln(S)} N_{\max}$$

□

Remark 5.3.3. The results of the second experiment (section 5.2.2) suggest that, if S is fixed, in a network with a constant churn the number of level 1 migrations is $\Omega(N_{\max})$. Therefore, using proposition Proposition 5.3.1 we may conjecture that in general $M = \Omega(N_{\max})$. Finally, by the previous proposition we can conclude that for connected random graphs:

$$\Omega(N_{\max}) = M = \tilde{O}(N_{\max})$$

Chapter 6

Conclusion and Future Research

A hierarchical topology simplifies routing and name management: the routing tables are small and distributed hash tables can be overlaid in a natural way. However, when the network is dynamic, the task of constantly maintaining a coherent hierarchy becomes complicated. Not only nodes have to change addresses when a group becomes disconnected, but also additional measures are required when new nodes enter the network. In this dissertation, we have studied what are the necessary rules that have to be enforced in order to incrementally update the hierarchy and what protocols can realize their distributed implementation. We have analyzed the behavior of the rules under different network conditions. As expected, well connected and not very dynamic networks are easy to manage. In the worst case, the number of address changes is upper-bounded by $\tilde{O}(N)$, where N is the size of the network. However, with a constant churn, the number of migrations increases at least linearly as N grows.

There are two directions for continuing the research on scalable mesh networks. In the first one, we can try to optimize the hierarchical architecture by exploring various tradeoffs. For example, as explained in Remark 5.0.1, the handoff overhead caused by migrations can be reduced at the cost of increasing the communication cost of the HDHT. Another tradeoff arises while trying to minimize the latency stretch and the number of migrations: groups may be formed in such a way to minimize the introduced latency stretch. However, the formation of group nodes becomes dependent not only on the connectivity properties of the nodes, but also

on the weight of their links. Thus, when the links' weight changes new migrations may be required.

The second direction is to relax the connectivity constraint: instead of requiring a group node to be internally connected through physical paths, we allow the use of virtual circuits, or in another words, group nodes are created on a routing overlay imposed on the physical network. Relaxing the connectivity constraint simplifies many dynamics of the topology maintenance. However, if the overlay is not carefully constructed, the latency stretch may increase rapidly.
The approach of creating unconnected group nodes that minimize the latency stretch is strictly related to a current research field called *Name-independent Compact Routing*[31][32]. The Compact Routing problem is to construct a routing scheme that minimize both the latency stretch and the routing tables size.
One scheme that is similar to our relaxed hierarchical architecture is "Generalized routing scheme for $\tilde{O}(N^{1/2})$ space", presented in [34].
We note, however, that the current compact routing algorithms do not solve the original objective of constructing self-configuring scalable mesh networks. In fact, they assume that the network is static. The design of an efficient dynamic scheme that is updated incrementally as the network evolves is currently an open problem, even though some lower bounds for the involved communication cost are already known[36],[37].

Bibliography

[1] I.F. Akyildiz, X. Wang and W. Wang, *Wireless mesh networks: a survey*, Computer Networks Journal 47 (2005) (4), pp. 445-487.

[2] Fotios A. Elianos, Georgia Plakia, Pantelis A. Frangoudis and George C. Polyzos (2009) *Structure and Evolution of a Large-Scale Wireless Community Network*
http://mm.aueb.gr/publications/2009-WOWMOM-WCN.pdf

[3] B. Milic and M. Malek. *Analyzing large scale real-world wireless multihop network*. IEEE Communication Letters, 2007

[4] G. Bernardi, P. Buneman, and M. K. Marina. *Tegola Tiered Mesh Network Testbed in Rural Scotland*. In WiNS-DR'08, San Francisco, CA, USA, September 2008.

[5] S. Srivathsan, N. Balakrishnan, and S.S. Iyengar *Scalability in Wireless Mesh Networks*. Guide to Wireless Mesh Networks, pp. 325-437, Springer

[6] L. Kleinrock and F. Kamoun, *Hierarchical routing for large networks*. Computer Networks, Vol. 1, No. 3, pp. 155-174.

[7] D. Krioukov and kc claffy. *Toward compact interdomain routing*. arXiv:cs.NI/0508021.

[8] L. Kleinrock and J. Sylvester, *Optimum transmission radii for packet radio networks or why six is a magic number*, Proc. IEEE National Telecommunications Conference, Birmingham, AL, December 1978, pp. 4.3.1-4.3.5.

[9] Jakob Eriksson, Michalis Faloutsos, Srikanth Krishnamurthy. *Routing Scalability in MANETs*, University of California, Riverside.

[10] G. Pei, M. Gerla, and X. Hong. *Lanmar: Landmark routing for large scale wireless ad hoc networks with group mobility*. In ACM MobiHOC'00, 2000.

[11] S. Du, A. Khan, S. PalChaudhuri, A. Post, A.K. Saha, P. Druschel, D.B. Johnson, and R. Riedi, *Safari: A Self-Organizing Hierarchical Architecture for Scalable Ad Hoc Networking*, Ad Hoc Networks J., 2007

[12] R. Ramanathan and M. Steenstrup, *Hierarchically-organized, multihop mobile wireless networks for quality-of-service support*, ACM/Baltzer Mobile Networks and Applications, Vol. 3 (1998), pp. 101-119.

[13] A.D. Amis, R. Prakash, T.H.P. Vuong, and D.T. Huynh, *Max-Min D-Cluster Formation in Wireless Ad Hoc Networks*, Proc. IEEE Infocom, pp. 32-41, 2000

[14] J. Eriksson, M. Faloutsos, and S. Krishnamurthy. *Scalable ad hoc routing: The case of dynamic addressing*. In Proc. IEEE INFOCOM, Hong Kong, China, Mar 2004.

[15] J. Eriksson, M. Faloutsos, and S. Krishnamurthy, *Dart: dynamic address routing for scalable ad hoc and mesh networks*, IEEE/ACM Transactions on Networking, vol. 15, no. 1, pp. 119132, 2007.

[16] Andrea Lo Pumo, *Netsukuku topology*, http://netsukuku.freaknet.org/doc/main_doc/topology.pdf

[17] I. Stoica, R. Morris, D. Lieben-Nowell, D. Karger, M. Kaashoek, F. Dabek, H. Balakrishnan *Chord: a scalable peer-to-peer lookup protocol for Internet applications*, IEEE Transactions on Networks, 11(1) 17-32, 2003

[18] Michael Garey and David Johnson, *Computers and Intractability - A Guide to the Theory of NP-completeness*; Freeman, 1979.

[19] John Sucec, Ivan Marsic. *Hierarchical routing overhead in mobile ad hoc networks.* Mobile Computing, IEEE Transactions on, 3, Jan 2004

[20] F. G. Nocetti, J. S. Gonzalez, I. Stojmenovic, *Connectivity based k-hop clustering in wireless networks*, Telecommunication Systems 22 (2003) 1-4, 205-220, 2003.

[21] S. Banerjee and S. Khuller, *A Clustering Scheme for Hierarchical Control in Multihop Wireless Networks*, in Proceedings of IEEE INFOCOM, April 2001.

[22] Bradford L. Chamberlain. *Graph Partitioning Algorithms for Distributing Workloads of Parallel Computations.* Technical Report UW-CSE-98-10-03, University of Washington, October 1998.

[23] W. Kernighan and S. Lin. *An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal*, 49:291-307, February 1970.

[24] Leslie Lamport, *Paxos made simple.* ACM SIGACT News (Distributed Computing Column) 32 (2001) 18-25

[25] Muthitacharoen Athicha, Gilbert Seth, Morris Robert. *Etna: a Fault-tolerant Algorithm for Atomic Mutable DHT Data*, CSAIL Technical Report

[26] J. Sucec and I. Marsic, *Location management handoff overhead in hierarchically organized mobile ad hoc networks*, Proc. Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, 19 April 2002.

[27] P. Erdös and A. Rényi, Publ. Math. Debrecen 6, 290 1959.

[28] Chen Avin, *Distance graphs: from random geometric graphs to Bernoulli graphs and between*, Proceedings of the fifth international workshop on Foundations of mobile computing, August 18-21, 2008, Toronto, Canada

[29] Albert R. and Barabási A.-L., *Statistical mechanics of complex networks*, Rev. Mod. Phys. 74, 47-97 (2002).

[30] NetworkX Python library http://networkx.lanl.gov/

[31] Dima Krioukov, kc claffy, Kevin Fall, and Arthur Brady. *On Compact Routing for the Internet*. ACM SIGCOMM Computer Communication Review (CCR), Vol. 37, No. 3, 2007.

[32] Papadimitriou, D. and NV, A.L.B. *Compact Routing: Challenges, Perspectives, and Beyond*.
http://streaming.info.ucl.ac.be/data/grascomp/pdf/TFISS09-Papadimitriou.pdf

[33] I.Abraham, C.Gavoille, D.Malkhi, N.Nisan, and M.Thorup. *Compact name-independent routing with minimum stretch*. ACM SPAA, 2004.

[34] Arias M., Cowen L. J., Laing K. A., Rajaraman R., and Taka O. *Compact routing with name independence*. In Proceedings of the 15th annual ACM Symposium on Parallel Algorithms and Architectures. 2003. ACM, New York, 184192

[35] M. Thorup and U. Zwick. *Approximate distance oracles*. In Proc. 33rd ACM Symp. on Theory of Computing, pp. 183-192, May 2001.

[36] Y.Afek, E.Gafni, and M.Ricklin, *Upper and lower bounds for routing schemes in dynamic networks*, In Proc. of FOCS, 1989.

[37] A.Korman and D.Peleg, *Dynamic routing schemes for general graphs*, In Proc. of ICALP 2006, Part I, LNCS 4051, pp. 619-630, Springer-Verlag Berlin Heidelberg 2006.