Adaptive Geometric Routing for the Internet Backbone

Rein Houthooft

Promotoren: prof. dr. ir. Didier Colle, prof. dr. ir. Mario Pickavet
Begeleiders: dr. Wouter Tavernier, Seyedeh Sahel Sahhaf

Masterproef ingediend tot het behalen van de academische graad van
Master in de ingenieurswetenschappen: computerwetenschappen

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniël De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2013-2014

UNIVERSITEIT
GENT

Adaptive Geometric Routing for the Internet Backbone

Rein Houthooft

Promotoren: prof. dr. ir. Didier Colle, prof. dr. ir. Mario Pickavet
Begeleiders: dr. Wouter Tavernier, Seyedeh Sahel Sahhaf

Masterproef ingediend tot het behalen van de academische graad van
Master in de ingenieurswetenschappen: computerwetenschappen

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniël De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2013-2014

UNIVERSITEIT
GENT

Acknowledgement

I would like to thank my promoters prof. dr. ir. Didier Colle and prof. dr. ir. Mario Pickavet for the opportunity they gave me to work in this field of study for nearly a year. Furthermore many thanks to my tutors Seyedeh Sahel Sahhaf and dr. Wouter Tavernier for guiding me throughout the course of this work. I greatly appreciate the support and freedom they gave me in following my own path of interest.

Ghent, January 2014

Rein Houthooft

Adaptive Geometric Routing for the Internet Backbone

Rein Houthooft

Promotoren: prof. dr. ir. Didier Colle, dr. ir. Mario Pickavet
Begeleiders: dr. Wouter Tavernier, Seyedeh Sahel Sahhaf

Masterproef ingediend tot het behalen van de academische graad van Master in de ingenieurswetenschappen: computerwetenschappen

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniël De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2013-2014

Keywords: geometric routing, load balancing, scale-free network, BGP, fault-tolerance

Adaptive Geometric Routing for the Internet Backbone

Rein Houthooft
Supervisors: Seyedeh Sahel Sahhaf, dr. Wouter Tavernier, prof. dr. ir. Didier Colle, prof. dr. ir. Mario Pickavet

***Abstract*—The application of geometric routing to scale-free inter-AS topologies is investigated. The main contribution forms a family of routing schemes, called Forest Routing, based on the principles of geometric routing focusing on memory scalability and natural load balancing. This is achieved by using an aggregation of greedy embeddings along with a novel distance function. Incorporating link load information in the forwarding layer ensures load balancing behaviour, while still attaining a short average path length. Furthermore the routing mechanisms are highly resilient towards network failures and are as such able to deal even with a severely deteriorated network.**

***Index Terms*—geometric routing, load balancing, scale-free network, fault-tolerance**

I. Introduction

Currently traffic routing within the Internet backbone is done by the Border Gateway Protocol (BGP). Though being a fundamental Internet protocol, it copes with several issues. Due to the vast growth of the Internet, the number of Autonomous Systems (ASs) is increasing steadily. This poses great memory scalability problems for routers as the number of BGP forwarding entries severely increases. This is caused by every node being required to store a gigantic amount of route information [1] [2]. Clearly BGP was simply never built to withstand such a growth. Therefore a need exists for alternative routing mechanisms that are scalable by nature. Furthermore it would be beneficial to have desirable properties such as load balancing behaviour and fault-tolerance.

In the last decade, geometric routing systems started emerging. Though they were originally designed for ad-hoc wireless networks and wireless sensor networks (WSNs), researchers have found that they are also applicable to wired networks that. This entails a different approach as wired networks are generally modelled as scale-free graphs rather than unit-disk graphs, which act as a model for WSNs. Geometric routing makes use of a graph embedding in which a graph is embedded into a mathematical space. Herein every node of the network is assigned a set of coordinates. Based on these coordinates and a corresponding distance function, geometric routing systems route packets closer to its destination by following a distance-decreasing path.

The application of geometric routing to wired networks is still relatively unexplored. An open question is achieving traffic load balancing while still guaranteeing a short average path length and a 100% packet delivery rate. This paper researches how load balancing can be achieved using geometric routing. The main contribution is a family of routing mechanisms called Forest Routing (FR) which are able to achieve load balancing, a low stretch and 100% delivery success rate while being inherently resilient towards node and link failures. They are applicable to a wide range of network types, but designed with scale-free networks in mind.

II. Related work

Many geometric routing algorithms target UDGs rather than scale-free networks. Hence these are difficult to apply to the Internet AS-level topology. However, it is useful to investigate what techniques are used to achieve load balancing in geometric routing. In general load balancing mechanisms can be divided into two classes: active and passive techniques. Passive load balancing draws from the inherent structure of the routing algorithm to spread traffic equally over the network. Active load balancing on the other hand makes use of live traffic information steer traffic away from hotspots. As a result active load balancing techniques are able to adjust routing to a changing traffic matrix.

Many load balancing techniques used in geometric routing applied to WSNs aim at lowering the number of overloaded nodes to avoid quick energy depletion due to their limited battery sizes. As such most load balancing research is focussed on node load balancing rather than link load balancing. One of such energy-driven systems is Geographical and Energy Aware Routing [3]. This mechanism makes use of a 2D Euclidean embedding and utilizes a forwarding decision heuristic based on the current neighbouring nodes' battery levels. As traffic is guided towards nodes with more energy, this also leads to node load balancing because the main source of energy depletion is packet processing. [4] proposes a similar approach. By only allowing nodes to use local neighbourhood information, the routing scheme remains scalable. A more advanced energy-based routing mechanism is Curve-Based Greedy Routing [5]. Herein traffic is guided by a B-spline which is calculated by the source node and stored in the packet headers. By using a selection mechanism based on the distance to this B-spline and the node energy level, nodes have more freedom in making forwarding decisions than basic greedy routing mechanisms. A feedback system is used such that nodes may inquire the source node to recompute the used B-spline to actively reroute traffic around hotspots.

LBLSP uses a non-Euclidean routing system based on curves around obstacles targeting wireless networks [6]. Traffic hotspots are modelled as virtual objects that are avoided by the routing algorithm. How this can be implemented is not specified. Circular Sailing Routing [7] focuses on passive load balancing. This routing scheme makes use of a stereographic projection of 2D Euclidean coordinates onto a sphere. The

authors report that a common problem in UDG based networks is traffic congestion at the center of the network under a uniform traffic matrix. Due to the lack of a sphere center, the system naturally avoids the center hotspot of the network. A similar system is Curveball Routing [8] which also makes use of a 2D Euclidean plane that is projected onto a sphere to avoid congestion.

Another way of balancing traffic passively is using multiple spanning tree based embeddings [9]. Root nodes are chosen randomly to avoid the congestion of the root node, which is reported as a general problem in tree-based geometric routing systems [10]. However as the authors target UDGs, they do not incorporate link load balancing. The tests are also limited to small networks. Virtual Polar Coordinate Routing [10] makes use of smart routing to attain passive load balancing. Smart routing requires the network to be a UDG with a 2D Euclidean embedding. Furthermore the tree should be ordered in a specific way for smart routing to work which makes it hard to apply to scale-free networks.

III. Theoretical foundation

In this section a theoretical foundation for the Forest Routing (FR) scheme is built based on the work of [11] and [12]. Geometric routing systems make use of the graph embedding concept. Such an embedding is a mapping between vertices of a graph and a certain mathematical space, formally defined in the following definition.

Definition 1. *Let S be a set and δ a metric function over S. Let $G = (V, E)$ be a graph, then an embedding of G into S is a mapping $f : V \to S$ such that $\forall u, v \in V : u \neq v \Leftrightarrow f(u) \neq f(v)$.* [13]

To guarantee 100% routing delivery success rate a greedy graph embedding is required, defined in by the following definition.

Definition 2. *A greedy embedding of a graph $G = (V, E)$ in a metric space (S, δ) is a mapping $f : V \to S$ with the following property: for every pair of distinct vertices $u, w \in V$ there exists a vertex v adjacent to u such that $\delta(f(v), f(w)) < \delta(f(u), f(w))$.* [14]

Herein a metric space is the double formed by a space and complementary distance function more formally defined by the following definition.

Definition 3. *For a set S and a function $\delta : S \times S \to \mathbb{R}$ such that the following conditions hold $\forall u, v, w \in S$:*

1) $\delta(u, v) \geq 0 \wedge \delta(u, v) = 0 \Leftrightarrow u = v$
2) $\delta(u, v) = \delta(v, u)$
3) $\delta(u, w) \leq \delta(u, v) + \delta(v, w)$

Then δ is called a metric and the double (S, δ) is called a metric space.

In this work a spanning tree $T = (V, E')$ of the underlying network $G = (V, E)$ is used to generate an embedding. This embedding makes use of a vertex labelling procedure as described by [12] and a distance function representing the shortest path length in T based on [11]. In this work the embedding target space S is denoted as the *tree space* $\mathbb{T}$. This tree space can be defined as

$$\mathbb{T} = \bigcup_{n \in \mathbb{N}} \Big((0)^\frown \mathbb{N}^n \Big) \tag{1}$$

in which the function $(^\frown) : \mathbb{N}^m \times \mathbb{N}^n \to \mathbb{N}^{m+n}$ represents the concatenation of two tuples. The labels assigned by the labelling procedure are now interpreted as coordinates in $\mathbb{T}$. These coordinates form a greedy embedding [11] which is denoted as $\mathcal{T}$. Therefore each vertex $v \in V$ corresponds to a point in $\mathbb{T}$ identified by the coordinates $\mathcal{T}(v)$.[1]

The distance function δ is defined as

$$\delta(u, v) = |u| + |v| - 2|\phi(u, v)| \tag{2}$$

with $\phi : \mathbb{N}^n \times \mathbb{N}^m \to \mathbb{N}^p$ ($p \leq \min\{m, n\}$) the function which generates the largest common prefix of two tuples; $|u|$ represents the size of the coordinates of vertex u. This results in a distance function $\delta : \mathbb{T} \times \mathbb{T} \to \mathbb{R}^+$ that, together with the space $\mathbb{T}$, forms the metric space $(\mathbb{T}, \delta)$, which can easily be proven. Now this metric space can be used according to the principles of geometric routing, which means that every vertex is aware of the coordinates of its neighbourhood and that each node tries to forward a packet to a neighbouring node closer to the destination based on a distance function.

IV. Multi-embedding

A first step towards load balancing is the use of k spanning trees $T_i = (V, E'_i)$ with $\forall i \in \{0, 1, \ldots, k-1\} : E'_i \subseteq E$ to form k greedy embeddings of G into $\mathbb{T}$ instead of only one. These different greedy embeddings are denoted as $\mathcal{T}_i$. The notation δ is now used to denote the k-tuple of distances in all of the k embeddings. Therefore δ_i represents the tree space distance for the i-th embedding $\mathcal{T}_i$. By using a multi-embedding it is possible to avoid traffic congestion at nodes residing at lower depths in the tree, which is huge problem for spanning tree based geometric routing algorithms [10]. The reason is two-fold:

1) Because of the existence of k different root nodes, traffic congestion is now divided amongst them.
2) The chances of having shortcuts available increases, which leads to a less crowded root and an overall lower stretch. Shortcuts for a graph $G = (V, E)$ and a spanning tree $T = (V, E')$ are defined as links $e \in E$ for which holds $e \notin E'$.

The sacrifice made here is the increased storage requirements of the packet headers and the increased computational complexity of the forwarding layer. This second increase in complexity can however be mitigated by high parallelizability of the forwarding decision making procedure.

To have access to a maximum number of shortcuts, the spanning tree creation mechanism should minimize the tree redundancy. With tree redundancy the overlap of different spanning trees in the same network is meant. For example when two trees $T_i = (V, E')$ and $T_j = (V, E'')$ have a maximal redundancy, they completely overlap, meaning that

[1] Instead of writing $\mathcal{T}(u)$ we will make no distinction between the point a vertex represents in its embedding space and the node itself. From the context the meaning should be clear.

$E' = E''$. Because of this there is no advantage in using them both for routing purposes as they will lead to the same set of coordinates, as well as the same set of shortcut links. When these two trees are said to have low redundancy, the set $E' \cap E''$ is small. Having a high tree redundancy should be avoided for two reasons:

1) It leads to low fault-tolerance: a failing link which happens to be a parent-child link for a large fraction of the total trees will likely cause routing voids.
2) Little shortcuts exist: this causes traffic to be routed in the direction of the root node, increasing the root traffic hotspot.

A tree building procedure leading to k spanning trees of minimal depth, minimizing tree redundancy, is used to build the different greedy embeddings $\mathcal{T}_i$ used to support the geometric routing mechanisms presented in the next section.

V. Routing variants

A. Greedy Forest Routing (GFR)

A straightforward way of routing using multiple embeddings would be to allow a vertex to freely alternate between them, due to their individual greediness. This naive forwarding mechanism fails due to the lack of a cycle avoidance mechanism. Routing along a distance-decreasing path in $\mathcal{T}_i$ may increase the distance in $\mathcal{T}_j$. At a certain vertex along the routing path the packet may be send back to its origin, routing the packet along a cycle. A cycle avoidance solution is enforcing each vertex along the routing path to decrease its minimum distance to the destination. This way of working is similar to the TCGR mechanism [9]. For this reason a new distance function $\epsilon : \mathbb{T}^k \times \mathbb{T}^k \to \mathbb{R}^+$ function is defined as

$$\epsilon(u, v) = \min_{0 \leq i < k} \{\delta_i(u, v)\} \ \ \forall u, v \in V \tag{3}$$

The k embeddings into $\mathbb{T}$ can now be treated as a single k-dimensional embedding into $\mathbb{T}^k$. As such the principles of geometric routing can be followed by having a relaxed metric space $(\mathbb{T}^k, \epsilon)$. This double cannot be regarded as a true metric space as the triangle-inequality no longer holds (Def. 3 property 3), hence we call it relaxed. When forwarding, multiple neighbours may have an equal ϵ-distance. Therefore a random choice will be made among them.

This routing mechanism is called the greedy variant of the final Forest Routing (FR) mechanism and is a special case of Hybrid Forest Routing (HFR). Though Greedy Forest Routing (GFR) solely focuses on attaining a low stretch, the embedding into $\mathbb{T}^k$ leads to increased passive load balancing behaviour over $\mathbb{T}$.

B. Load Balanced Forest Routing (LBFR)

To supplement the passive load balancing behaviour emerging from GFR, an active load balancing approach was developed called Load Balanced Forest Routing (LBFR). This system can be seen once more as a special case of the final HFR routing scheme. Now vertices $u \in V$ make use of load information of their incident edges $e \in I(u)$. Solely using local link information is advantageous as it is scalable by nature and therefore fitting for a large-scale distributed setting like the Internet backbone. LBFR relaxes the greedy requirements of GFR. Alternately routing on different embeddings $\mathcal{T}_i$ independently is now allowed. Because naive switching between embeddings may introduce cycles, routing is guided by an auxiliary function κ.

The function κ makes use of a function $\delta^* : \mathcal{P} \times \mathbb{T}^k \to \mathbb{N}^k$ which outputs a k-tuple storing the minimal distance to a destination d attained by a packet so far along its routing path P_u, before arriving at the current node u, for each of the k embeddings. The i-th element of δ^* is denoted as δ_i^*. The set $\mathcal{P}$ represents the union of all possible paths of the network. Assume a packet has been routed along the path $P = \langle p_0, p_1, \ldots, p_n \rangle$ towards a destination vertex d, the function κ is then of the type $\mathcal{P} \times \mathbb{T}^k \to \mathbb{N}$ and is defined as

$$\kappa(P_{p_n}, d) = \sum_{i=0}^{k-1} \delta_i^* \left(P_{p_n}, d\right) \tag{4}$$

with $\delta^*(P_{p_n}, d)$ a function of the type $\mathcal{P} \times \mathbb{T}^k \to \mathbb{N}^k$ that is defined recursively as

$$\delta_i^*(P_{p_0}, d) = \delta_i(p_0, d) \tag{5}$$

$$\delta_i^*(P_{p_n}, d) = \min\{\delta_i^*(P_{p_{n-1}}, d), \delta_i(p_n, d)\} \quad \forall i \in \{0, \ldots, k-1\} \wedge \forall n > 0 \tag{6}$$

Herein P_u represents the path P until u has been reached, it consists of the vertices that a packet arriving at u has reached. p_0 is the source vertex of the path P. Each of the minimum distance of the k embeddings is thus represented by an element $\delta_i^*(P_u, d)$. The LBFR system will route a packet along a distance-decreasing path for each $\mathcal{T}_i$ individually but with a restriction that κ has to decrease strictly monotonically along the routing path: $\kappa(P_{p_n}, d) < \kappa(P_{p_{n-1}}, d) < \ldots < \kappa(p_0, d)$. When forwarding, a node u will select those neighbouring nodes which have a strictly decreasing κ-value and add them to a set $S(u)$. Next u will choose a vertex $v \in S(u)$ as the next node for which the current traffic load of the link (u, v) is minimal compared to its other incident links $I(u)$.

In order to guarantee packet delivery for every source-destination combination the following theorems are introduced.

Theorem 1. *Let $G = (V, E)$ be a graph with k embeddings $\mathcal{T}_i$ for $0 \leq i < k$ in the metric space $(\mathbb{T}, \delta)$. Let d be the destination node, then for every path $P \in \mathcal{P}$ of a graph with $v \in V$ as last element and where d has not been reached yet, thus $d \notin P$, the set of neighbours $S(v)$ for which the value of the κ-function strictly decreases is not empty.*

Proof: Assume a packet arriving at a vertex v by following a path $P = \langle \ldots, u, v \rangle$. Assume this packet has to be forwarded to a destination vertex d and that $d \notin P$. This means that $S(u) \neq \varnothing$. Therefore $\kappa(v) < \kappa(u)$. Because of the definition of κ as the sum defined by Eq. (4): $\exists i \in \{0, 1, \ldots, k-1\} : \delta_i^*(P_v, d) < \delta_i^*(P_u, d)$. Combining the definition of δ^* in Eq. (6) with the definition of the $\min$-function gives $\delta_i(v, d) < \delta_i^*(P_u, d)$ and $\delta_i^*(P_u, d) \leq \delta_i(u, d)$. Therefore, again because of Eq. (6), $\delta_i^*(P_v, d) = \delta_i(v, d)$. Also, $\delta_i(v, d) < \delta_i(u, d)$ which means that the distance towards d in embedding $\mathcal{T}_i$ has decreased. Because $\mathcal{T}_i$ is a

greedy embedding and Definition 2: $\exists w \in N(v) : \delta_i(w,d) < \delta_i(v,d)$. Thus because of Eq. (6) $\delta_i^*(P_v^\frown w, d) = \delta(w,d)$ and therefore $\delta_i^*(P_v^\frown w, d) < \delta_i^*(P_v, d)$. Combining this with Eq. (4) leads to $\kappa(P_v^\frown w, d) < \kappa(P_v, d)$ based on the fact that δ_i^* never increases along a path. From this follows: $S(v) \neq \varnothing$. As such, any element from $S(v)$ is a suitable next vertex to which the packet can be forwarded without violating the LBFR restrictions.

This theorem also holds for a source vertex s. Because of Eq. (5), every value δ_i^* is equal to the distance δ_i. Therefore any vertex for which the distance towards the destination decreases (and such a vertex exists due to each $\mathcal{T}_i$ being a greedy embedding) leads to a lower κ-value. Thus at the source vertex $S(s) \neq \varnothing$. ∎

Theorem 2. *The path followed by a packet routed on a graph $G = (V, E)$ by LBFR is never a cycle.*

Proof: Assume a destination vertex d and a packet traveled along $P = \langle \ldots, u, v, \ldots, w \rangle$ arriving at $w \in N(v)$. When arriving at v for the first time, $\delta_i^*(P_v, d) \leq \delta_i(v,d)$ because of Eq. (6). Since the values of δ^* can never increase due to the definition of the min-function, upon calculating the κ-function value for the second time for vertex v (this time from w): $\nexists i \in \{0, 1, \ldots, k-1\} : \delta_i(v,d) < \delta_i^*(P_w, d)$ because if there would exists such an i then δ^* would already have been updated to this value the first time the packet has arrived at v. Thus the κ-value cannot decrease the second time v is encountered. Therefore no vertex appears twice along the path followed by a packet routed according to LBFR which enforces κ to be strictly monotonically decreasing along a routing path. ∎

Theorem 3. *A packet routed according to the principles of LBFR on a graph $G = (V, E)$ will arrive at its destination.*

Proof: Because κ is strictly monotonically decreasing and the initial κ-value at the source vertex is finite (assuming $|V|$ is finite), it will become 0 after a finite number of vertices have been traversed unless, it would have been routed along a cycle or unless it would have encountered a void. These two last cases are impossible due to Theorem 1 and Theorem 2. When for a vertex $v \in V$ and a destination d holds $\kappa(P_v, d) = 0$ means that $\forall i \in \{0, 1, \ldots, k-1\} : \delta_i(v,d) = 0$. Because of property 1 in Definition 3 which defines a metric space, $v = d$. Therefore the destination has been reached at vertex v. ∎

Because of the previous theorems, the LBFR scheme is always able to route a packet to its destination without encountering cycles or voids. This is important because routing on a network such as the Internet backbone has to be guaranteed for all source-destination pairs.

C. Hybrid Forest Routing (HFR)

In terms of stretch and load balancing behaviours, GFR and LBFR are two opposites: GFR attains a low stretch but has no load balancing technique while LBFR achieves load balancing combined with a very large stretch (see Section VI). To combine the best of both worlds, Hybrid Forest Routing (HFR) was developed. It makes a trade-off between stretch and load balancing by employing a cost function that combines link load information with the ϵ-distance to the destination. This cost function $C : V^3 \to \mathbb{R}$ is defined as

$$C(u,n,d) = \gamma \cdot \hat{L}(u,n)^\alpha + (1-\gamma) \cdot \epsilon(n,d)^\alpha \qquad (7)$$

for $n \in N(u)$, with the ϵ-function defined by Eq. (3). The function $\hat{L}(u,v)$ represents the normalized traffic load of the edge between u and v.[2] This is the traffic load of the link (u,v) divided by the average load of all the node's incident links $I(u)$. This normalized load is defined by

$$\hat{L}(u,n) = \frac{d_G(u) \cdot L(u,n)}{\sum\limits_{v \in N(u)} L(u,v)} \qquad (8)$$

The factor $\gamma \in [0,1]$ is a weight factor to scale between greedy and load balanced routing. The power $\alpha \in]0, +\infty[$ allows for non-linear tuning. As can be seen, HFR also uses the relaxed-metric space $(\mathbb{T}^k, \epsilon)$, but because the cost function is an extension of the ϵ-function it does not employ greedy routing. It is not even necessarily distance decreasing in ϵ. To guarantee packet delivery, the κ-function from LBFR is used to steer packets towards their destination. Hence, it is called a hybrid mechanism. It is able to attain strong load balancing while keeping the stretch down, which is shown in Section VI. GFR and LBFR can now be seen as two special instances of HFR on the opposite side of the spectrum. When $\gamma = 1$ the LBFR mechanism is recreated. When $\gamma = 0$ the HFR reverts to GFR.

VI. Results and Discussion

The different routing algorithms GFR, LBFR and HFR have been analysed on their average stretch $\bar{\rho}$ and link load balancing behaviour. We define the stretch of a path generated by a routing algorithm as the path length divided by the shortest path length between the same source and destination nodes. The load balancing behaviour was measured by calculating the β_E-metric [15] defined as

$$\beta_E = \frac{\left(\sum_{e=1}^{|E|} w_e\right)^2}{|E| \sum_{e=1}^{|E|} w_e^2} \qquad (9)$$

with w_e the weight of edge $e \in E$. Note that $\rho \geq 1$ (1: shortest path routing) and $0 \leq \beta_E \leq 1$ (0: no load balancing, 1: traffic equal on all links). The routing behaviour has been simulated by a programmatic routing framework. In this framework every edge is assigned a weight w_e, initially set to 0. This weight w_e is increased whenever traffic is simulated along the edge e. All traffic is Pareto distributed in size and a uniform traffic matrix was employed at all times. The algorithms were tested on the CAIDA graph [17] as well as scale-free networks generated according to [16] (the number in the legend of each plot indicates the number of nodes of the network, e.g. 8k means 8000 nodes).

In Fig. 1 (top) the effect of changing the number of embeddings k (or the dimension of the space $\mathbb{T}^k$) on the average stretch $\bar{\rho}$ and the β_E-ratio is plotted for GFR. Asymptotic behaviour $\bar{\rho} \to 1$ can be observed as $k \to +\infty$. This can

[2]The representation of the load may be arbitrarily chosen but should be consistent for all network links.

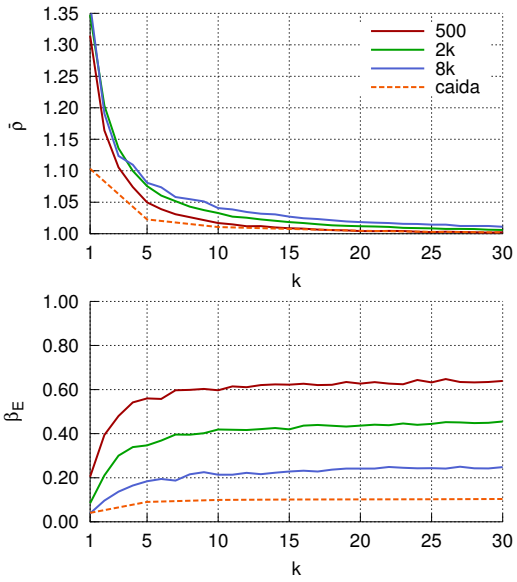


Figure 1. GFR: average stretch $\bar{\rho}$ (top) and link load balancing metric β_E (bottom) in function of the dimension k of the embedding space.

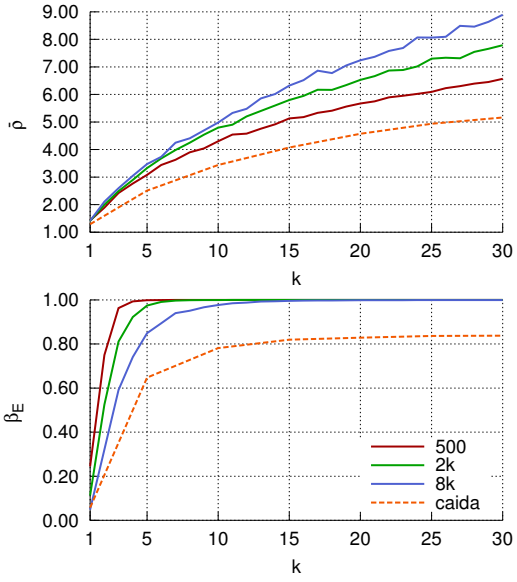


Figure 2. LBFR: average stretch $\bar{\rho}$ (top) and link load balancing metric β_E (bottom) in function of the dimension k of the embedding space.

be explained by the availability of more embeddings, which allows more routing forwarding freedom. As a result, there is an increased chance that a combination of embeddings will lead to a short path between two nodes. As the number of vertices increases, k also needs to increase to maintain the same stretch. However, no scalability issues can be noticed. For the CAIDA graph, even at low k values, low $\bar{\rho}$-values can be observed. A possible explanation is that there exist little alternative paths in the CAIDA graph, compared to the scale-free graphs. In Fig. 1 (bottom) the link load balancing behaviour is plotted. Though there is no active mechanism steering for traffic load balancing in GFR, better load balancing is obtained as k increases. This may be due to the existence of multiple root nodes, splitting the root hotspot among k different roots. Also the existence of additional short paths between different source and destination nodes reduces the reliance on the root node to act as a transit hub for traffic between different parts of the network.

Fig. 2 shows the previous experiment applied to LBFR. In Fig. 2 (top) a steep increase in stretch can be observed as k increases. Because more embeddings are available, there exist more forwarding candidates that respect the κ-restriction (κ should be strictly monotonically decreasing along the routed path). Every node focuses entirely on load balancing the load of its outgoing links, sacrificing the stretch while doing so. In Fig. 2 (bottom) the β_E-ratio is depicted. As k goes up, β_E shoots up towards 1, which indicates near-perfect load balancing behaviour. Although the β_E-ratio indicates that the traffic is spread equally over all links, this does not mean that the sum of the traffic over all network links is at its lowest point. Because the stretch increases by a large factor, the total network traffic goes up too. For this reason, the stretch should always be prioritized in order not to overload the network.

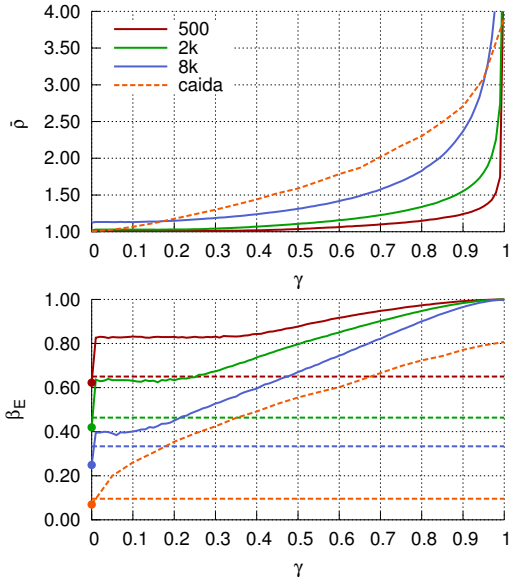


Figure 3. HFR: average stretch $\bar{\rho}$ (top) and link load balancing metric β_E (bottom) in function of a varying γ-value.

Now HFR, which unites both GFR and LBFR, is evaluated. When the parameter γ in Eq. (7) is shifted towards 0, the GFR system is recreated, while shifting γ to 1 results in LBFR. Therefore in Fig. 3 a sensitivity analysis for the γ-parameter has been conducted. Fig. 3 (top) shows that at low γ-values $\bar{\rho}$ becomes equally low as with GFR. This stretch remains steady for $0 \leq \gamma \leq 0.5$. Afterwards, $\bar{\rho}$ starts to incline quickly as $\gamma \to 1$. This is consistent with HFR approximating LBFR. The β_E-values in Fig. 3 (bottom) indicates that that shifting γ between its two extremes gives the expected results. Something interesting happens when looking at the right side of $\gamma = 0$. A step can be noticed such that β_E suddenly rises. When inspecting Fig. 3 (top) there is no such step or $\bar{\rho}$. This can be explained by the fact that when a node has to make a forwarding decision, lots of potential candidates will have

an equal distance to the destination. Therefore it does not matter which node to take as the next node in terms of stretch. However when taking into account load balancing, a huge improvement can be made by prioritizing those links with a low current load.

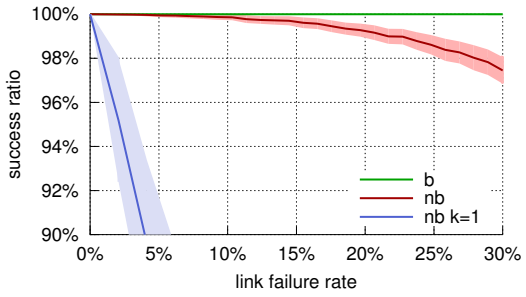


Figure 4. Fault-tolerance: the x-axis depicts the fraction of links removed from the total number of links that can be removed ($|E| - |V| + 1$) without disconnecting the graph $G = (V, E)$. HFR ($\gamma = 0.1$, $\alpha = 1$, $k = 15$) with (b) and without (nb) backup mechanism was tested along with GFR with only a single embedding.

In Fig. 4 the routing success rate of the HFR system with $\gamma = 0.1$, $\alpha = 1$ and $k = 15$ is shown, alongside HFR combined with a backup routing system [18] and GFR with $k = 1$, exercised on a scale-free graph with 500 nodes. Links were removed probabilistically such that the link failures are spread out evenly over the network rather than being randomly concentrated in a certain area. In the figure it can be noticed that HFR is easily augmented with a backup mechanism that is able to guarantee 100% success rate even in severe network failure scenarios. However, even without a backup routing mechanism, HFR is able to attain a success rate over 97% when 30% of the possible links are removed. This is a huge improvement over GFR with $k = 1$. This can be explained by the fact that more embeddings create more possible pathways from source to destination. Combined with its load balancing behaviour, this makes HFR naturally very fault-tolerant.

VII. Conclusion

In this paper a theoretical framework was built which serves as a foundation for the developed family of geometric routing systems, called Forest Routing (FR). Combining a strictly greedy approach (GFR) with a load balanced routing scheme (LBFR) resulted in HFR. In HFR routing paths approximate shortest paths very closely, while achieving link traffic load balancing, two features until now not perceived to be compatible. Due to its distributed nature and local routing decision making it is highly scalable regarding router memory requirements, making it robust towards the vast Internet AS growth that can be currently witnessed.

Furthermore the HFR system has favourable characteristics such as inherent fault-tolerance and the ability to cope with a highly deteriorated network topology. Even at link failures rates of 30% it is able to guarantee success rates as high as 97%. When augmenting the system with a tailored backup routing mechanism, the success rate becomes 100% even at high failure rates.

Acknowledgment

This work was carried out using the Stevin Supercomputer Infrastructure at Ghent University, funded by Ghent University, the Hercules Foundation and the Flemish Government – department EWI. This work is partly funded by the European Commission through the EULER project (grant 258307), part of the Future Internet Research and Experimentation (FIRE) objective of the Seventh Framework Programme (FP7). To visualize the different routing algorithms the open-source graph visualization framework Gephi [19] was used.

References

[1] A. Narayanan, "A survey on bgp issues and solutions," *CoRR*, vol. abs/0907.4815, 2009.

[2] F. Papadopoulos, D. Krioukov, M. Bogua, and A. Vahdat, "Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces," in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–9.

[3] Y. Yu, R. Govindan, and D. Estrin, "Geographical and Energy Aware Routing: a recursive data dissemination protocol for wireless sensor networks," *Energy*, vol. 463, 2001.

[4] K. Zeng, K. Ren, W. Lou, and P. J. Moran, "Energy aware efficient geographic routing in lossy wireless sensor networks with environmental energy supply," *Wirel. Netw.*, vol. 15, no. 1, pp. 39–51, Jan. 2009.

[5] J. Zhang, Y.-p. Lin, M. Lin, P. Li, and S.-w. Zhou, "Curve-based greedy routing algorithm for sensor networks," in *Proceedings of the Third international conference on Networking and Mobile Computing*, ser. ICCNMC'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 1125–1133.

[6] N. Carlsson and D. L. Eager, "Non-euclidian geographic routing in wireless networks," *Ad Hoc Netw.*, vol. 5, no. 7, pp. 1173–1193, Sep. 2007.

[7] F. Li, S. Chen, and Y. Wang, "Load balancing routing with bounded stretch," *EURASIP J. Wirel. Commun. Netw.*, vol. 2010, pp. 10:1–10:16, Apr. 2010.

[8] L. Popa, A. Rostamizadeh, R. Karp, C. Papadimitriou, and I. Stoica, "Balancing traffic load in wireless networks with curveball routing," in *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, ser. MobiHoc '07. New York, NY, USA: ACM, 2007, pp. 170–179.

[9] M. Tang, H. Chen, G. Zhang, and J. Yang, "Tree cover based geographic routing with guaranteed delivery," in *Communications (ICC), 2010 IEEE International Conference on*, 2010, pp. 1–5.

[10] J. Newsome and D. Song, "Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information." ACM Press, 2003, pp. 76–88.

[11] E. Chávez, N. Mitton, and H. Tejeda, "Routing in wireless networks with position trees," in *Ad-Hoc, Mobile, and Wireless Networks*, ser. Lecture Notes in Computer Science, E. Kranakis and J. Opatrny, Eds. Springer Berlin Heidelberg, 2007, vol. 4686, pp. 32–45.

[12] A. Korman, D. Peleg, and Y. Rodeh, "Labeling schemes for dynamic tree networks," in *STACS 2002*, ser. Lecture Notes in Computer Science, H. Alt and A. Ferreira, Eds. Springer Berlin Heidelberg, 2002, vol. 2285, pp. 76–87.

[13] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *Proceedings of the 9th annual international conference on Mobile computing and networking*, ser. MobiCom '03. New York, NY, USA: ACM, 2003, pp. 96–108.

[14] R. Kleinberg, "Geographic routing using hyperbolic space," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 1902–1909.

[15] H. Velayos, V. Aleo, and G. Karlsson, "Load balancing in overlapping wireless lan cells," in *Communications, 2004 IEEE International Conference on*, vol. 7, 2004, pp. 3833–3836 Vol.7.

[16] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[17] Y. Hyun, A. Broido, and k. claffy, "Traceroute and BGP AS Path Incongruities," Cooperative Association for Internet Data Analysis (CAIDA), Tech. Rep., Mar 2003.

[18] A. Cvetkovski and M. Crovella, "Hyperbolic embedding and routing for dynamic graphs," in *INFOCOM 2009, IEEE*, 2009, pp. 1647–1655.

[19] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," 2009.

Adaptieve geometrische routering voor de internet-backbone

Rein Houthooft
Begeleiders: Seyedeh Sahel Sahhaf, dr. Wouter Tavernier
Promotors: prof. dr. ir. Didier Colle, prof. dr. ir. Mario Pickavet

***Samenvatting*—In dit werkstuk werd het toepassen van geometrische routering op het scale-free inter-AS netwerk onderzocht. De belangrijkste bijdrage is een familie van routeringsalgoritmen genaamd Forest Routing (FR), gebaseerd op de principes van geometrische routering. Hierbij wordt er gefocust op de schaalbaarheid van het geheugen van de routers en de natuurlijke loadbalancing van het netwerkverkeer. Dit wordt bereikt door het gebruik van een aggregatie van greedy inbeddingen, in samenwerking met een nieuwe afstandsfunctie. Het aanwenden van verkeersinformatie in de forwardinglaag kan voor loadbalancing zorgen in combinatie met een lage gemiddelde padlengte. Voorts zijn de routeringsalgoritmen zeer bestendig ten opzichte van het falen van links en knopen, en zijn ze dusdanig in staat om zelfs met een grondig verstoord netwerk om te kunnen.**

***Trefwoorden*—geometrische routering, loadbalancing, scale-freenetwerken, fouttolerantie**

I. Introductie

Op dit moment wordt routering binnen de internet backbone voorzien door het Border Gateway Protocol (BGP). Hoewel het een fundamenteel internet protocol is kampt het met ettelijke problemen. Het aantal Autonome Systemen (AS'en) gestaag door de aanzienlijke groei van het internet. Dit veroorzaakt schaalbaarheidsproblemen van het geheugen van routers omdat het aantal rijen in de BGP tabellen eveneens sterk toeneemt. Dit komt doordat elke knoop verplicht is een gigantische hoeveelheid informatie betreffende verschillende routes bij te houden [1] [2]. Hierdoor is het duidelijk dat het BGP nooit gebouwd was om aan deze groei te voldoen. Er is dus nood aan alternatieve routeringsmechanismen die van nature schaalbaar zijn. Voorts zou het goed zijn moesten deze toekomstige algoritmen begeerde eigenschappen bezitten zoals loadbalancing en fouttolerantie.

In het laatste decennium maakten geometrische routeringsalgoritmen hun intrede. Hoewel ze origineel ontworpen waren voor ad-hoc draadloze netwerken en draadloze sensor netwerken (WSN'en) hebben onderzoekers ontdekt dat ze ook voor bekabelde netwerken bruikbaar zijn. Dit vereist echter een andere aanpak aangezien bekabelde netwerken meestal als scale-freenetwerken gemodelleerd worden in plaats van eenheidsschijfgrafen (unit-disk graphs of UDG's), welke een model zijn voor WSN'en. In geometrische routering wordt er gebruik gemaakt van een graaf inbedding waarbij elke vertex wordt ingebed in een wiskundige ruimte. Hierbij wordt elke knoop van een set van coördinaten in deze ruimte voorzien. Op basis van deze coördinaten en een bijhorende afstandsfunctie kunnen pakketten via geometrische routering naar hun bestemming geleid worden langs een afstandsverminderend pad.

Het toepassen van geometrische routering op bekabelde netwerken is een nog relatief onontgonnen onderwerp. Een open vraag is dan ook hoe we loadbalancing van het netwerkverkeer kunnen verkrijgen in combinatie met een korte gemiddelde padlengte en een succesratio van 100%. Dit artikel onderzoekt hoe loadbalancing toegepast kan worden in geometrische routering. De belangrijkste bijdrage is een familie van routeringsalgoritmes genaamd Forest Routing (FR). Deze algoritmen zijn in staat loadbalancing te combineren met een lage stretch en 100% routeringssuccesratio, terwijl ze resistent zijn ten opzichte van het falen van links en knopen. Ze zijn toepasbaar op een breed scala van netwerken maar werden specifiek ontwikkeld voor scale-freenetwerken.

II. Gerelateerd werk

Veel geometrische routeringsalgoritmen doelen op UDG's i.p.v scale-freenetwerken. Hierdoor zijn deze ze moeilijk toe te passen op het inter-AS netwerk. Het is echter nuttig te onderzoeken welke technieken gebruikt worden om loadbalancing in geometrische routering te verkrijgen. Over het algemeen kunnen loadbalancing technieken in twee categorieën opgedeeld worden: passieve en actieve loadbalancing strategieën. Passieve loadbalancing strategieën maken gebruik van de inherente structuur van het routeringsalgoritme om verkeer te spreiden over het netwerk. Daarentegen maakt actieve loadbalancing gebruik van verkeersinformatie om verkeer weg van leiden van hotspots. Een gevolg is dat actieve loadbalancing mechanismen routering kunnen aanpassen naargelang de verkeersmatrix.

Veel loadbalancing strategieën die op WSN'en toegepast worden trachten het aantal overbelaste knopen te verminderen, omwille van hun gelimiteerde batterijcapaciteit. Dusdanig focust loadbalancing onderzoek zich vooral op knoop loadbalancing eerder dan link loadbalancing. Een routeringsalgoritme gestuurd door de energieniveaus van de knopen in het netwerk is Geographical and Energy Aware Routing [3]. Dit mechanisme maakt gebruik van een 2D Euclidische inbedding en een forwardingbeslissingsheuristiek gebaseerd op de energieniveaus van de buurt van de huidige knoop. Doordat verkeer gestuurd wordt naar knopen met meer energie leidt dit tot loadbalancing, aangezien de grootste bron van energieverbruik het verwerken van pakketten is. [4] stelt een gelijkaardige aanpak voor. Door knopen enkel toe te laten hun lokale buurt te benutten blijft het routeringsalgoritme

schaalbaar. Een meer geavanceerde aanpak is Curve-Based Greedy Routing [5]. Hierin wordt verkeer gestuurd d.m.v. een B-spline welke berekend wordt door de bronknoop waarna deze in de pakketheader geplaatst wordt. Door een selectiecriterium te benutten dat gebaseerd is op de afstand tot deze B-spline en het energieniveau van een node hebben knopen meer vrijheid in het maken van hun forwardingbeslissing dan gewone greedy routering. Daarnaast gebruiken de knopen een feedbackmechanisme om ervoor te zorgen dat de bronknoop een nieuwe B-spline berekent om het verkeer rond hotspots te kunnen sturen.

LBLSP gebruikt een niet-Euclidisch routeringssysteem gebaseerd op curves rondom obstakels, specifiek gericht op draadloze netwerken [6]. Verkeershotspots worden gemodelleerd als virtuele objecten die het algoritme tracht te ontwijken. Echter hoe dit geïmplementeerd kan worden wordt niet gespecificeerd. Circular Sailing Routing [7] focust op het bereiken van passieve loadbalancing. Dit routeringsalgoritme maakt gebruik van een stereografische projectie van 2D Euclidische coördinaten op een sfeer. De auteurs rapporteren dat een algemeen probleem van UDG-gebaseerde netwerken is dat het centrum van het netwerk opgestopt raakt onder een uniforme verkeersmatrix. Omdat een sfeer geen centrum heeft, vermijdt het algoritme op natuurlijke wijze dit centraal opstoppingsprobleem. Een gelijkaardige aanpak is Curveball Routing [8] waarbij er ook gebruik gemaakt wordt van een projectie van een 2D Euclidisch vlak op een sfeer om opstopping te vermijden.

Een andere manier om passieve loadbalancing te verkrijgen is het gebruik van meerdere inbeddingen gebaseerd op verschillende opspannende bomen van het netwerk [9]. De wortelknopen worden op willekeurige wijze gekozen om het opstoppingseffect ronde de wortelknoop te vermijden, wat een algemeen probleem is van boomgebaseerde geometrische routeringssystemen [10]. Omdat de auteurs zich richten op UDG's houden ze echter geen rekening met linkloadbalancing. Ook zijn de testen beperkt tot kleine netwerken. Virtual Polar Coordinate Routing [10] maakt gebruik van smart routing om passieve loadbalancing te bereiken. Smart routing vereist dat het netwerk een UDG is dat ingebed is in een 2D Euclidische ruimte. Voorts dienen de boomknopen op een specifieke wijze geordend te zijn opdat smart routing zou slagen, wat moeilijk toe bereiken is in scale-freenetwerken.

III. Theoretisch fundament

In deze sectie wordt een theoretisch fundament voor Forest Routing (FR) gebouwd gebaseerd op het werk van [11] en [12]. Geometrische routeringsalgoritmen maken gebruik van het concept van een graafinbedding. Zo een inbedding is een mapping tussen knopen van een graaf en een wiskundige ruimte, wat als volgt formeel gedefinieerd kan worden.

Definitie 1. *Laat S een verzameling zijn en δ een metriekfunctie over S. Laat $G = (V, E)$ een graaf zijn, dan is een inbedding van G in S een mapping $f : V \to S$ zodat $\forall u, v \in V : u \neq v \Leftrightarrow f(u) \neq f(v)$.* [13]

Om een routeringssuccesratio van 100% te kunnen garanderen is er nood aan een greedy graafinbedding, als volgt gedefinieerd.

Definitie 2. *Een greedy inbedding van een graaf $G = (V, E)$ in een metriekruimte (S, δ) is een mapping $f : V \to S$ met de volgende eigenschap: voor elk paar verschillende knopen $u, w \in V$ bestaat er een knoop v aanliggend aan u zodat $\delta(f(v), f(w)) < \delta(f(u), f(w))$.* [14]

Hierin is een metriekruimte een koppel gevormd door een ruimte en een bijhorende afstandsfunctie, meer formeel gedefinieerd door volgende definitie.

Definitie 3. *Voor een verzameling S en een functie $\delta : S \times S \to \mathbb{R}$ waarvoor de volgende eigenschappen gelden, $\forall u, v, w \in S$:*

1) $\delta(u, v) \geq 0 \wedge \delta(u, v) = 0 \Leftrightarrow u = v$
2) $\delta(u, v) = \delta(v, u)$
3) $\delta(u, w) \leq \delta(u, v) + \delta(v, w)$

Dan is δ een metriek en het koppel (S, δ) is een metriekruimte.

In dit werk wordt een opspannende boom $T = (V, E')$ van het netwerk $G = (V, E)$ gebruikt om een inbedding te genereren. Deze inbedding maakt gebruik van een knooplabellingsprocedure zoals beschreven door [12] en een afstandsfunctie welke de kortste padlengte in T voorstelt, gebaseerd op [11]. In dit werk wordt de doelruimte S voor de inbedding genoteerd als de boomruimte $\mathbb{T}$. Deze boomruimte kan gedefinieerd worden als

$$\mathbb{T} = \bigcup_{n \in \mathbb{N}} \Big((0)^\frown \mathbb{N}^n \Big) \tag{1}$$

waarin de functie $(^\frown) : \mathbb{N}^m \times \mathbb{N}^n \to \mathbb{N}^{m+n}$ de samenvoeging van twee tupels voorstelt. De labels toegewezen door het labellingsproces worden nu geïnterpreteerd als coördinaten in $\mathbb{T}$. Deze coördinaten vormen een greedy inbedding [11] welke genoteerd wordt als $\mathcal{T}$. Hierdoor correspondeert elke knoop $v \in V$ met een punt in $\mathbb{T}$ geïdentificeerd door de coördinaten $\mathcal{T}(v)$.[1]

De afstandsfunctie δ wordt gedefinieerd als

$$\delta(u, v) = |u| + |v| - 2|\phi(u, v)| \tag{2}$$

met $\phi : \mathbb{N}^n \times \mathbb{N}^m \to \mathbb{N}^p$ ($p \leq \min\{m, n\}$) de functie die de langste gemeenschappelijke prefix van twee tupels teruggeeft; $|u|$ stelt de lengte van de coördinaten van de knoop u voor. Dit resulteert in een afstandsfunctie $\delta : \mathbb{T} \times \mathbb{T} \to \mathbb{R}^+$ welke samen met de ruimte $\mathbb{T}$ een metriekruimte $(\mathbb{T}, \delta)$ vormt (dit is triviaal te bewijzen). Nu kan deze metriekruimte gebruikt worden volgens de principes van geometrische routering, die inhouden dat elke knoop zich bewust is van de coördinaten van zijn buurt en dat elke knoop een pakket dichter naar de bestemming tracht te sturen, gebaseerd op een afstandsfunctie.

IV. Multi-inbedding

Een eerste stap tot het verkrijgen van loadbalancing is het gebruik van k opspannende bomen $T_i = (V, E_i')$ met $\forall i \in \{0, 1, \ldots, k-1\} : E_i' \subseteq E$ om op basis hiervan k greedy inbeddingen van G in $\mathbb{T}$ te vormen, in plaats van één enkele inbedding. Deze verschillende greedy inbeddingen

[1] In plaats van $\mathcal{T}(u)$ te schrijven zullen we geen onderscheid maken tussen het punt dat een knoop voorstelt in zijn inbeddingsruimte en de knoop zelf. De exacte betekenis kan afgeleid worden uit de context.

worden genoteerd als $\mathcal{T}_i$. De notatie δ beschrijft nu het k-tupel van afstanden in elk van de k inbeddingen. Hierdoor beschrijft δ_i de afstand in de boomruimte voor de i-de inbedding $\mathcal{T}_i$. Door het aanwenden van een multi-inbedding is het mogelijk verkeersopstoppingen rond knopen op lagere dieptes in de boom te vermijden. Dit is namelijk een groot probleem in geometrische routeringsalgoritmen die werken met opspannende bomen [10]. De reden hiervoor is tweeledig:

1) Omdat er nu k verschillende wortelknopen bestaan wordt de verkeersopstopping gespreid over deze k knopen.
2) De kans dat er shortcuts beschikbaar zijn wordt groter, wat leidt tot een minder opgestopte wortelknoop en een gemiddeld lagere stretch. Shortcuts van een graaf $G = (V, E)$ en een opspannende boom $T = (V, E')$ zijn links $e \in E$ waarvoor geldt $e \notin E'$.

De opoffering die hier gemaakt wordt is verhoogde opslagvereisten van de pakketheaders en een verhoogde computationele complexiteit van de forwardinglaag. Echter, deze tweede verhoogde complexiteit kan beperkt worden door het uitbuiten van de parallelliseerbaarheid van de forwardingbeslissingsprocedure.

Om toegang te hebben tot zoveel mogelijk shortcuts, dient het opspannende boom generatieproces boomredundantie te vermijden. Hiermee wordt de overlap tussen verschillende opspannende bomen bedoeld, binnen hetzelfde netwerk. Bijvoorbeeld, wanneer twee bomen $T_i = (V, E')$ en $T_j = (V, E'')$ een maximale redundantie hebben, overlappen ze volledige, wat betekent dat $E' = E''$. Hierdoor is er geen voordeel bij het gebruik van beide bomen voor routeringsdoeleinden aangezien ze tot de zelfde set van coördinaten leiden, alsook dezelfde set van shortcutlinks. Wanneer deze bomen een lage redundantie hebben is de set $E' \cap E''$ klein. Hoge boomredundantie moet vermeden worden omwille van twee redenen:

1) Het leidt tot een lage fouttolerantie: een falende link die een ouder-kind link is voor een grote fractie van de bomen heeft een grotere waarschijnlijkheid om tot routeringsvoids te leiden.
2) Er bestaan weinig shortcuts: hierdoor wordt verkeer meer in de richting van de wortelknoop gestuurd, wat het wortelopstoppingseffect versterkt.

Een boomgeneratieprocedure die leidt tot k opspannende bomen van minimale diepte dat ook boomredundantie vermijdt dient dus aangewend te worden bij het opbouwen van de verschillende greedy inbeddingen $\mathcal{T}_i$ om de geometrisch routeringsalgoritmen beschreven in volgende secties te ondersteunen.

V. Routeringsvarianten

A. Greedy Forest Routing (GFR)

Een logische manier om te routeren op basis van meerdere inbeddingen is het toelaten van een knoop om vrij te alterneren tussen deze verschillende inbeddingen, op basis van hun afzonderlijke greedyheid. Echter, deze naïeve manier van forwarding faalt omdat cycli niet vermeden worden. Routering volgens een afstandsverminderend pad in $\mathcal{T}_i$ kan de afstand in $\mathcal{T}_j$ namelijk doen toenemen, omdat in een bepaalde knoop langs het gerouteerde pad beslist kan worden om het pakket terug te zenden naar zijn beginpunt, waardoor een cyclus ontstaat. Een manier om cycli te vermijden is het verplichten van elke knoop om zijn minimum afstand tot de bestemming te verminderen bij elke hop. Deze manier van werken is gelijkaardig aan het TCGR mechanisme [9]. Daarom wordt er een nieuwe afstandsfunctie $\epsilon : \mathbb{T}^k \times \mathbb{T}^k \to \mathbb{R}^+$ geïntroduceerd:

$$\epsilon(u, v) = \min_{0 \leq i < k} \{\delta_i(u, v)\} \ \ \forall u, v \in V \tag{3}$$

De k inbedding in $\mathbb{T}$ kunnen nu gezien worden als één enkele k-dimensionale inbedding in $\mathbb{T}^k$. Dusdanig kunnen de principes van geometrische routering gerespecteerd worden door gebruik te maken van de gerelaxeerde metriekruimte $(\mathbb{T}^k, \epsilon)$. Dit koppel vormt echter geen metriekruimte meer in de strikte zin van het woord aangezien de driehoeksongelijkheid niet meer geldt, vandaar dat we het een gerelaxeerde metriekruimte noemen. Bij het forwarden kan het voorkomen dat meerdere buren een gelijke ϵ-afstand hebben. Indien dit het geval is wordt er onder hen een willekeurige keuze gemaakt.

Dit routeringsalgoritme wordt de greedy variant van het finale Forest Routing (FR) mechanisme genoemd en is een speciaal geval van Hybrid Forest Routing (HFR). Hoewel Greedy Forest Routing (GFR) louter focust op het behalen van een lage stretch, leidt de inbedding in $\mathbb{T}^k$ tot een verhoogd passief loadbalancinggedrag ten opzichte van $\mathbb{T}$.

B. Load Balanced Forest Routing (LBFR)

Om het passieve loadbalancinggedrag van GFR te ondersteunen werd een actieve loadbalancingaanpak ontworpen, genaamd Load Balanced Forest Routing (LBFR). Dit systeem kan opnieuw gezien worden als een speciaal geval van het uiteindelijke schema HFR. Nu maken knopen $u \in V$ gebruik van belastingsinformatie betreffende hun aanliggende links $e \in I(u)$. Uitsluitend gebruik maken van lokale belastingsinformatie is voordelig aangezien dit van nature schaalbaar is, wat het zeer passend maakt in een grootschalige gedistribueerde context zoals de internet backbone. LBFR relaxeert de greedy vereisten van GFR. Alternerend onafhankelijk routeren op basis van verschillende inbeddingen $\mathcal{T}_i$ is nu toegestaan. Doordat naïef veranderen van inbedding leidt tot cycli, wordt het routeren gestuurd door een hulpfunctie κ.

De functie κ maakt gebruik van een functie $\delta^* : \mathcal{P} \times \mathbb{T}^k \to \mathbb{N}^k$ die een k-tupel teruggeeft dat de minimale afstand tot de bestemming d, bereikt langs het gerouteerde pad P_u voor elke van de k inbeddingen, voorstelt, voordat in u aankwam. Hier stelt $\mathcal{P}$ de unie van alle mogelijke paden in het netwerk voor. Stel dat een pakket langs het pad $P = \langle p_0, p_1, \ldots, p_n \rangle$ gerouteerd werd naar een bestemmingsknoop d, dan is de functie κ van het type $\mathcal{P} \times \mathbb{T}^k \to \mathbb{N}$ en is deze gedefinieerd als

$$\kappa(P_{p_n}, d) = \sum_{i=0}^{k-1} \delta_i^* (P_{p_n}, d) \tag{4}$$

met $\delta^*(P_{p_n}, d)$ een functie van het type $\mathcal{P} \times \mathbb{T}^k \to \mathbb{N}^k$ die recursief gedefinieerd is als

$$\delta_i^*(P_{p_0}, d) = \delta_i(p_0, d) \tag{5}$$

$$\delta_i^*(P_{p_n}, d) = \min\{\delta_i^*(P_{p_{n-1}}, d), \delta_i(p_n, d)\} \quad \forall i \in \{0, \ldots, k-1\} \wedge \forall n > 0 \tag{6}$$

Hierin stelt P_u het pad P voor tot u bereikt werd. Dit pad bestaat dus uit alle knopen die het pakket bezocht heeft voordat het ontvangen werd in u (u inclusief). p_0 is de bronknoop van het pad P. Elke van de minimale afstanden van de k inbeddingen wordt dus gerepresenteerd door een element $\delta_i^*(P_u, d)$. Het i-de element van δ^* wordt genoteerd als δ_i^*. Het LBFR systeem zal een pakket routeren langs een afstandsverminderend pad voor elke $\mathcal{T}_i$ afzonderlijk maar met de beperking dat κ strikt monotoon moet dalen langs het gevolgde pad: $\kappa(P_{p_n}, d) < \kappa(P_{p_{n-1}}, d) < \ldots < \kappa(p_0, d)$. Bij het forwarden zal een knoop u de buren selecteren die een strikt dalende κ waarde hebben en ze toevoegen aan de verzameling $S(u)$. Hierna zal u een knoop $v \in S(u)$ kiezen waarvoor de belasting van de link (u, v) minimaal is in vergelijking met de andere aanliggende links $I(u)$.

Om het bereiken van de bestemming te garanderen voor elke bron-bestemmingscombinatie worden de volgende theorema's geïntroduceerd:

Theorema 1. *Laat $G = (V, E)$ een graaf zijn met k inbeddingen $\mathcal{T}_i$ voor $0 \leq i < k$ in de metriekruimte $(\mathbb{T}, \delta)$. Laat d een bestemmingsknoop zijn, dan is voor elk pad $P \in \mathcal{P}$ in de graaf met $v \in V$ als laatste element en waarbij d nog niet bereikt is, dus $d \notin P$, de verzameling van buren $S(v)$ waarvoor de κ-functie strikt daalt niet leeg.*

Proof: Veronderstel een pakket dat toekomt in knoop v door een pad $P = \langle \ldots, u, v \rangle$ te volgend. Stel dat dit pakket naar de bestemmingsknoop d gestuurd dient te worden en dat $d \notin P$. Dit betekent dat $S(u) \neq \varnothing$. Daardoor is $\kappa(v) < \kappa(u)$. Omwille van de definitie van κ als de som gedefinieerd door vgl. (4): $\exists i \in \{0, 1, \ldots, k-1\} : \delta_i^*(P_v, d) < \delta_i^*(P_u, d)$. Combineer de definitie van δ^* in vgl. (6) met de definitie van de min-functie, dit geeft $\delta_i(v, d) < \delta_i^*(P_u, d)$ en $\delta_i^*(P_u, d) \leq \delta_i(u, d)$. Hierdoor, en opnieuw vanwege vgl. (6), $\delta_i^*(P_v, d) = \delta_i(v, d)$. Ook is $\delta_i(v, d) < \delta_i(u, d)$ wat betekent dat de afstand tot d in inbedding $\mathcal{T}_i$ verkleind is. Aangezien $\mathcal{T}_i$ een greedy inbedding is en definitie 2: $\exists w \in N(v) : \delta_i(w, d) < \delta_i(v, d)$. Dus omwille van vgl. (6) is $\delta_i^*(P_v^\frown w, d) = \delta(w, d)$, aldus is $\delta_i^*(P_v^\frown w, d) < \delta_i^*(P_v, d)$ waar. Dit combineren met vgl. (4) leidt tot $\kappa(P_v^\frown w, d) < \kappa(P_v, d)$ gebaseerd op het feit dat δ_i^* nooit stijgt langs het gevolgde pad. Hieruit volgt: $S(v) \neq \varnothing$. Dusdanig is eender welk element uit $S(v)$ een geschikte kandidaatsknoop om het pakket naartoe te sturen zonder de LBFR beperkingen te schaden.

Dit theorema is ook waar voor de bronknoop s. Omwille van vgl. (5) is elke waarde δ_i^* gelijk aan de afstand δ_i. Daarom zal elke knoop waarbij de afstand naar de bestemming daalt (en zo een knoop bestaat aangezien elke $\mathcal{T}_i$ een greedy inbedding is) leiden tot een lagere κ waarde. Dus voor de bronknoop geldt $S(s) \neq \varnothing$. ■

Theorema 2. *Het pad gevolgd door een pakket in een graaf $G = (V, E)$ dat gerouteerd wordt door LBFR is nooit een cyclus.*

Proof: Neem een bestemmingsknoop d en een pakket dat langs $P = \langle \ldots, u, v, \ldots, w \rangle$ gerouteerd is en toekomt in $w \in N(v)$. Wanneer er voor het eerst in v aangekomen wordt zal $\delta_i^*(P_v, d) \leq \delta_i(v, d)$ zijn door vgl. (6). Omdat de waarden van δ^* nooit kunnen stijgen omwille van de definitie van de min-functie zal bij de tweede keer dat κ berekend wordt voor knoop v (deze keer vanuit w) gelden: $\nexists i \in \{0, 1, \ldots, k-1\} : \delta_i(v, d) < \delta_i^*(P_w, d)$ omdat indien er zo een i zou bestaan zou δ^* reeds geüpdatet zijn naar deze waarden de eerste keer dat het pakket in v arriveerde. Dus de tweede keer dat v bereikt wordt kan κ niet meer dalen. Aldus kan geen knoop tweemaal bereikt worden door een pakket gerouteerd door LBFR zonder de beperking te schaden dat κ strikt monotoon dalend moet zijn langs het routeringspad. ■

Theorema 3. *Een pakket dat gerouteerd wordt volgens de principes van LBFR in een graaf $G = (V, E)$ zal aankomen in zijn bestemming.*

Proof: Omdat κ strikt monotoon dalend is en de aanvankelijke waarde van κ eindig is (in de veronderstelling dat $|V|$ eindig is), zal deze 0 worden na een eindig aantal stappen, tenzij er in een cyclus zou gerouteerd worden of er een void bereikt zou worden. Deze gevallen zijn uitgesloten door theorema 1 en theorema 2. Wanneer voor knoop $v \in V$ en een bestemming d geldt dat $\kappa(P_v, d) = 0$ betekent dit dat $\forall i \in \{0, 1, \ldots, k-1\} : \delta_i(v, d) = 0$. Omwille van eigenschap 1 in definitie 3, die een metriekruimte definieert, geldt $v = d$. Dus de bestemming is bereikt in v. ■

Omwille van de vorige theorema's zal LBFR altijd in staat zijn een pakket naar zijn bestemming te leiden zonder in cycli te routeren of voids tegen te komen. Dit is belangrijk aangezien routering in een netwerk zoals de internet backbone moet gegarandeerd worden voor alle bron-bestemmingsparen.

C. Hybrid Forest Routing (HFR)

In termen van stretch en loadbalancing zijn GFR en LBFR twee tegengestelden: GFR bereikt een lage stretch maar wendt geen loadbalancingtechniek aan, terwijl LBFR weliswaar loadbalancinggedrag vertoont maar een zeer hoge stretch heeft (zie sectie VI). Om het beste van deze twee werelden te combineren wordt Hybrid Forest Routing (HFR) geïntroduceerd. HFR maakt een trade-off tussen stretch en loadbalancing door een kostfunctie te gebruiken welke linkbelastingsinformatie combineert met de ϵ-afstand tot de bestemming. Deze kostfunctie $C : V^3 \to \mathbb{R}$ wordt gedefinieerd als

$$C(u, n, d) = \gamma \cdot \hat{L}(u, n)^\alpha + (1 - \gamma) \cdot \epsilon(n, d)^\alpha \qquad (7)$$

voor $n \in N(u)$, met de ϵ-functie gedefinieerd door vgl. (3). De functie $\hat{L}(u, v)$ is de genormaliseerde belasting van de link tussen u en v.[2] Dit is de verkeersbelasting van de link (u, v) gedeeld door de gemiddelde belasting van alle aanliggende links $I(u)$ van deze knoop. Deze genormaliseerde belasting is gedefinieerd als

$$\hat{L}(u, n) = \frac{d_G(u) \cdot L(u, n)}{\sum_{v \in N(u)} L(u, v)} \qquad (8)$$

Met $\gamma \in [0, 1]$ een gewichtsfactor om een afweging tussen greedy en routering met loadbalancing mogelijk te maken. De exponent $\alpha \in]0, +\infty[$ staat niet-lineaire tuning toe. Zoals

[2]De representatie van de genormaliseerde belasting kan arbitrair gekozen worden maar dient consistent te zijn voor alle links in het netwerk.

gezien kan worden maakt HFR eveneens gebruik van de gerelaxeerde metriekruimte $(\mathbb{T}^k, \epsilon)$, echter via de kostfunctie welke een extensie is van de ϵ-functie. Hierdoor gebruikt HFR geen greedy routering. Het is zelfs niet zo dat HFR afstandsverminderende paden in ϵ genereert. Om te garanderen dat pakketten hun bestemming bereiken wordt de κ-functie van LBFR gebruikt om pakketten te sturen. Vandaar dat we HFR een hybride mechanisme noemen. HFR is in staat loadbalancing te bereiken in combinatie met een lage stretch, wat wordt aangetoond in sectie VI. GFR en LBFR kunnen nu gezien worden als twee speciale gevallen van HFR aan andere uiteinden van het spectrum. Wanneer $\gamma = 1$ wordt het LBFR mechanisme gecreëerd. Indien $\gamma = 0$ bekomen we het GFR mechanisme.

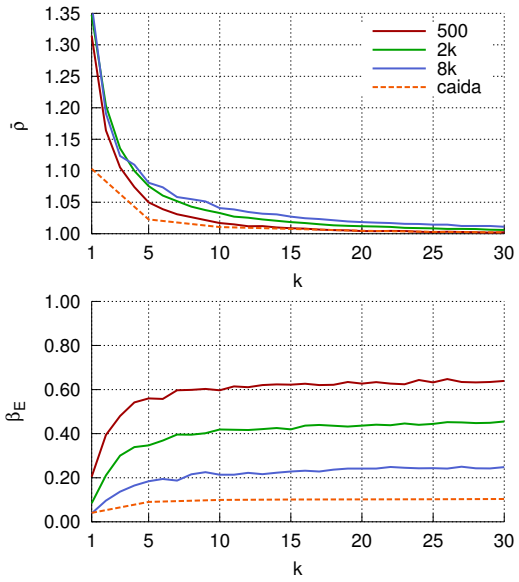
VI. Resultaten en discussie

De verschillende algoritmen GFR, LBFR en HFR zijn geanalyseerd op basis van hun gemiddelde stretch $\bar{\rho}$ en hun link loadbalancinggedrag. De stretch wordt gedefinieerd als de lengte van een pad gegenereerd door een routeringsalgoritme gedeeld door de kortste padlengte tussen dezelfde bron en bestemming. Het loadbalancinggedrag wordt gemeten door het berekenen van de β_E-metriek [15] gedefinieerd als

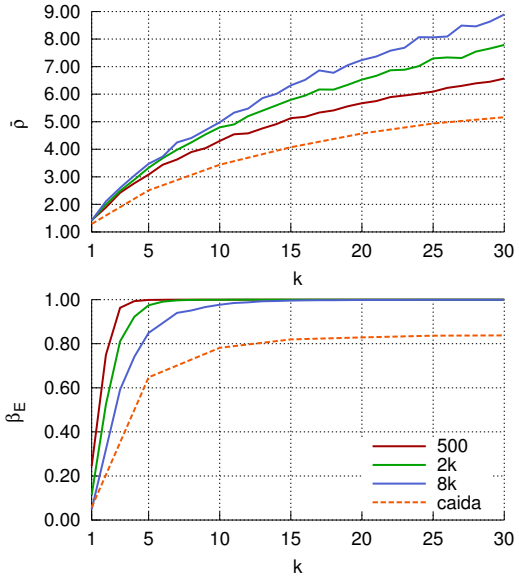
$$\beta_E = \frac{\left(\sum_{e=1}^{|E|} w_e\right)^2}{|E| \sum_{e=1}^{|E|} w_e^2} \tag{9}$$

met w_e het gewicht van de link $e \in E$. Merk op dat $\rho \geq 1$ (1: kortste pad routering) en dat $0 \leq \beta_E \leq 1$ (0: geen loadbalancing, 1: verkeer is evenredig verdeeld over alle links). Het routeringsgedrag werd gesimuleerd door een programmatisch routeringsraamwerk. In dit raamwerk is elke link een gewicht w_e toebedeeld, welke initieel op 0 staat. Dit gewicht w_e wordt verhoogd telkens wanneer er verkeer over de link e gesimuleerd wordt. Alle verkeer is Pareto-gedistribueerd in grootte en een uniforme verkeersmatrix werd aangewend. De algoritmen werden getest op de CAIDA graaf [16] en scale-freenetwerken gegenereerd volgens [17] (het nummer in de legende van elke plot duidt op het aantal knopen in de graaf, vb. 8k betekent 8000 knopen).

In figuur 1 (boven) wordt het effect getoond van het veranderen van het aantal inbeddingen k (wat ook de dimensie van de ruimte $\mathbb{T}^k$ is) op de gemiddelde stretch $\bar{\rho}$ en de β_E-ratio voor GFR. Er kan asymptotisch gedrag $\bar{\rho} \to 1$ waargenomen worden als $k \to +\infty$. Dit kan verklaard worden door de beschikbaarheid van meerdere inbeddingen, wat meer routeringsvrijheid inzake de forwardingbeslissing toelaat. Hieruit volgt dat er een verhoogde kans is dat een combinatie van inbeddingen tot een kort pad tussen twee knopen zal leiden. Indien het aantal knopen omhoog gaat moet k ook meeschalen om dezelfde stretch aan te houden. Toch zijn er geen schaalbaarheidsproblemen merkbaar. Voor de CAIDA graaf kan zelfs bij lage k waarden een lage gemiddelde stretch $\bar{\rho}$ waargenomen worden. Een mogelijke verklaring is dat er weinig alternatieve paden aanwezig zijn in de CAIDA graaf, in vergelijking met de scale-freenetwerken. In figuur 1 (beneden) wordt het link loadbalancinggedrag geplot. Hoewel er geen mechanisme in GFR aanwezig is



Figuur 1. GFR: gemiddelde stretch $\bar{\rho}$ (boven) en linkloadbalancingmetriek β_E (beneden) in functie van de dimensie k van de inbeddingsruimte.

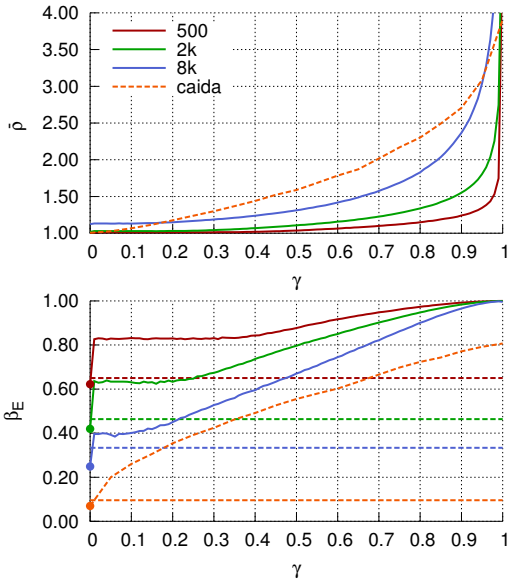


Figuur 2. LBFR: gemiddelde stretch $\bar{\rho}$ (boven) en linkloadbalancingmetriek β_E (beneden) in functie van de dimensie k van de inbeddingsruimte.

dat actief belasting spreidt kan er toch verbeterd loadbalancinggedrag waargenomen worden indien k groter wordt. Dit is mogelijks het resultaat van het bestaan van meerdere wortelknopen welke het wortelopstoppingseffect splitsen over k verschillende wortels. Ook het bestaan van additionele korte paden tussen verschillende bron- en bestemmingsknopen leidt tot een mindere afhankelijkheid van de wortelknoop als doorvoerhaven van netwerkverkeer tussen verschillende delen van het netwerk.

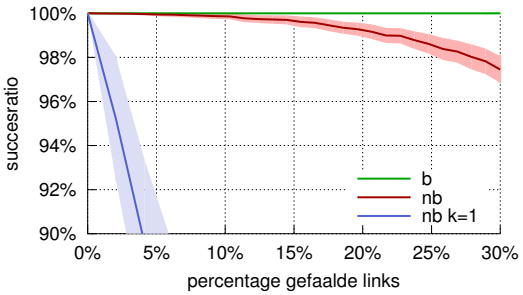
Figuur 2 toont het vorige experiment toegepast op LBFR. In figuur 2 (boven) kan een sterke stijging van de stretch waargenomen worden als k verhoogd wordt. Omdat er meer inbeddingen beschikbaar zijn, zijn er meer forwardingkandi-

daten die de κ-beperking respecteren (κ moet strikt monotoon dalend zijn langs het gevolgde routeringspad). Elke knoop focust nu volledig op het balanceren van verkeer op zijn uitgaande lijnen waardoor de stretch wordt opgeofferd. In figuur 2 (beneden) wordt de β_E-ratio getoond. Als k groter wordt schiet β_E snel naar 1, wat bijna perfect loadbalancing-gedrag indiceert. Hoewel de β_E-ratio toont dat het verkeer evenredig verspreid is over alle links betekent dit niet dat het de som van het verkeer in alle links van het netwerk op zijn laagste punt is. Omdat de stretch met een grote factor omhoog gaat zal het totale netwerkverkeer eveneens stijgen. Omwille van deze reden is het nuttig de stretch een hogere prioriteit toe te kennen dan het loadbalancinggedrag.



Figuur 3. HFR: gemiddelde stretch $\bar{\rho}$ (boven) en linkloadbalancingmetriek β_E (beneden) in functie van een variërende γ waarde.

Nu wordt HFR, het algoritme dat GFR en LBFR verenigt, geëvalueerd. Indien de parameter γ in vgl. (7) naar 0 verschoven wordt, wordt GFR benaderd. γ naar 1 schuiven resulteert in LBFR. Daarom wordt in figuur 3 een sensitiviteitsanalyse uitgevoerd van de γ parameter. Figuur 3 (boven) toont dat bij lage γ waarden $\bar{\rho}$ eveneens laag wordt, zoals bij GFR. Daarna blijft de stretch stabiel voor $0 \leq \gamma \leq 0.5$. Hierna begint $\bar{\rho}$ sterk te stijgen als $\gamma \to 1$. Dit is consistent met het benaderen van LBFR door HFR. De β_E waarden in figuur 3 (beneden) tonen aan de het verschuiven van γ tussen zijn twee extreme waarden het verwachte resultaat teruggeeft. Iets bijzonder gebeurt er aan de rechterzijde van $\gamma = 0$. Er kan een stap waargenomen worden zodat β_E plots stijgt. Indien we kijken naar figuur 3 (boven) kan zo een stap niet waargenomen worden voor $\bar{\rho}$. Dit kan verklaard worden door het feit dat bij de forwardingbeslissing van een knoop vele kandidaten een gelijke afstand tot de bestemming bezitten. Hierdoor maakt het niet uit welke knoop als volgende knoop geselecteerd wordt met betrekking tot de stretch. Indien belastingsinformatie gebruikt wordt kan er een grote verbetering verkregen worden door de links met een lage huidige belasting voor te nemen.



Figuur 4. Fouttolerantie: de x-as stelt de fractie van het aantal links die verwijderd werden voor ten opzichte van het totale aantal links dat verwijderd kan worden ($|E| - |V| + 1$) zonder de graaf te ontbinden. HFR ($\gamma = 0.1$, $\alpha = 1$, $k = 15$) met (b) en zonder (nb) backup-mechanisme werd getest naast GFR met een enkele embedding.

In figuur 4 wordt de routeringssuccesratio van het HFR systeem met $\gamma = 0.1$, $\alpha = 1$ en $k = 15$ getoond. Daarnaast wordt ook HFR gecombineerd met een backup-routeringssysteem [18] weergegeven en GFR met $k = 1$. Alles werd uitgevoerd op een scale-freenetwerk met 500 knopen. Links werden verwijderd op probabilistische wijze zodat het falen van de links evenredig verspreid werd over het gehele netwerk. In de figuur kan opgemerkt worden dat HFR gemakkelijk uit te rusten is met een backup-mechanisme waardoor het een routeringssuccesratio van 100% kan bereiken, zelfs indien er een groot aantal links niet beschikbaar zijn. Maar zelfs zonder backup-mechanisme is HFR in staat succesratio's van meer dan 97% te bereiken indien er tot 30% van de mogelijke links faalt. Dit is een enorme vooruitgang ten opzichte van GFR met $k = 1$. Dit kan verklaard worden door het feit dat meerdere inbeddingen meer mogelijke paden van bron tot bestemming aanbieden. Dit in combinatie met het loadbalancingmechanisme van HFR maakt het algoritme op natuurlijke wijze zeer fouttolerant.

VII. Conclusie

In dit werkstuk werd een theoretisch raamwerk gebouwd dat diende als basis voor de ontwikkelde familie van geometrische routeringssytemen, genaamd Forest Routing (FR). Een strikt greedy aanpak (GFR) werd gecombineerd met een loadbalancingstrategie (LBFR) wat resulteerde in HFR. In HFR worden zeer korte routeringspaden gecombineerd met een hoge loadbalancingfactor. Dit zijn twee eigenschappen die voorheen niet als compatibel gepercipieerd werden. Door de lokaliteit van de forwardingbeslissingsprocedure in HFR is dit algoritme hoogst schaalbaar wat betreft de geheugenvereisten van routers. Hierdoor is het robuust ten opzichte van een steeds groeiend aantal AS'en, wat vandaag waargenomen kan worden.

Voorts heeft HFR voordele karakteristieken zoals inherente fouttolerantie waardoor het beter kan omgaan met een zwaar beschadigd netwerk. Zelfs bij foutratio's van 30% kunnen succesratio's van meer dan 97% gegarandeerd worden. Indien het systeem uitgerust wordt met een backup-routeringsalgoritme wordt deze succesratio zelfs 100% bij een zwaar aangetast netwerk.

DANKWOORD

Dit werk was uitgevoerd gebruik makende van de Stevin Supercomputer Infrastructuur van Universiteit Gent, bekostigd door Universiteit Gent, de Herculesstichting en de Vlaamse Overheid – departement EWI. Dit werk is deels bekostigd door de Europese Commissie via het EULER project (subsidie 258307), deel van het Future Internet Research and Experimentation (FIRE) objectief van het Seventh Framework Programme (FP7). Om de verschillende routeringsalgoritmen te visualiseren werd er gebruik gemaakt van het visualisatieraamwerk Gephi [19].

REFERENTIES

[1] A. Narayanan, "A survey on BGP issues and solutions," *CoRR*, vol. abs/0907.4815, 2009.

[2] F. Papadopoulos, D. Krioukov, M. Bogua, and A. Vahdat, "Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces," in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–9.

[3] Y. Yu, R. Govindan, and D. Estrin, "Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks," *Energy*, vol. 463, 2001.

[4] K. Zeng, K. Ren, W. Lou, and P. J. Moran, "Energy aware efficient geographic routing in lossy wireless sensor networks with environmental energy supply," *Wirel. Netw.*, vol. 15, no. 1, pp. 39–51, Jan. 2009.

[5] J. Zhang, Y.-p. Lin, M. Lin, P. Li, and S.-w. Zhou, "Curve-based greedy routing algorithm for sensor networks," in *Proceedings of the Third international conference on Networking and Mobile Computing*, ser. ICCNMC'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 1125–1133.

[6] N. Carlsson and D. L. Eager, "Non-euclidian geographic routing in wireless networks," *Ad Hoc Netw.*, vol. 5, no. 7, pp. 1173–1193, Sep. 2007.

[7] F. Li, S. Chen, and Y. Wang, "Load balancing routing with bounded stretch," *EURASIP J. Wirel. Commun. Netw.*, vol. 2010, pp. 10:1–10:16, Apr. 2010.

[8] L. Popa, A. Rostamizadeh, R. Karp, C. Papadimitriou, and I. Stoica, "Balancing traffic load in wireless networks with curveball routing," in *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, ser. MobiHoc '07. New York, NY, USA: ACM, 2007, pp. 170–179.

[9] M. Tang, H. Chen, G. Zhang, and J. Yang, "Tree cover based geographic routing with guaranteed delivery," in *Communications (ICC), 2010 IEEE International Conference on*, 2010, pp. 1–5.

[10] J. Newsome and D. Song, "Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information." ACM Press, 2003, pp. 76–88.

[11] E. Chávez, N. Mitton, and H. Tejeda, "Routing in wireless networks with position trees," in *Ad-Hoc, Mobile, and Wireless Networks*, ser. Lecture Notes in Computer Science, E. Kranakis and J. Opatrny, Eds. Springer Berlin Heidelberg, 2007, vol. 4686, pp. 32–45.

[12] A. Korman, D. Peleg, and Y. Rodeh, "Labeling schemes for dynamic tree networks," in *STACS 2002*, ser. Lecture Notes in Computer Science, H. Alt and A. Ferreira, Eds. Springer Berlin Heidelberg, 2002, vol. 2285, pp. 76–87.

[13] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *Proceedings of the 9th annual international conference on Mobile computing and networking*, ser. MobiCom '03. New York, NY, USA: ACM, 2003, pp. 96–108.

[14] R. Kleinberg, "Geographic routing using hyperbolic space," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 1902–1909.

[15] H. Velayos, V. Aleo, and G. Karlsson, "Load balancing in overlapping wireless LAN cells," in *Communications, 2004 IEEE International Conference on*, vol. 7, 2004, pp. 3833–3836 Vol.7.

[16] Y. Hyun, A. Broido, and k. claffy, "Traceroute and BGP AS path incongruities," Cooperative Association for Internet Data Analysis (CAIDA), Tech. Rep., Mar 2003.

[17] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[18] A. Cvetkovski and M. Crovella, "Hyperbolic embedding and routing for dynamic graphs," in *INFOCOM 2009, IEEE*, 2009, pp. 1647–1655.

[19] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," 2009.

Contents

Acronyms

AE anchor node election 54, 67, 101, 102
AS Autonomous System 1–5, 20–23, 29, 113, 116
ASN Autonomous System Number 3, 4

BA Barabási-Albert 22
BFM breadth-first mode 54, 68, 69, 73, 80, 82, 101–105, 111
BFRM breadth-first redundant mode 54, 68, 73, 80, 82, 101–105, 111
BGP Border Gateway Protocol xix, 3–5, 19–21, 29, 113–115
BVR Beacon Vector Routing 17

CALB congestion-aware load balancing 73, 105–108, 126
CCDF complementary cumulative distribution function 22, 23
CPU central processing unit 30, 37
CSL Circular Sail Routing 25

DNS Domain Name System 78

ECMP equal-cost multi-path 4

FM first mode 54, 68, 80, 82
FR Forest Routing 39, 52–54, 58, 67, 73–79, 82, 83, 113–115

GEAR Geographic and Energy Aware Routing 24
GFR Greedy Forest Routing 54, 58, 59, 63, 64, 66, 67, 80, 83–89, 91, 92, 96, 102, 104, 106, 107, 110, 113
GG Gabriel graph 12
GLIDER Gradient Landmark-Based Distributed Routing for Sensor Networks 17
GLP Generalized Linear Preference 22
GP Gravity-Pressure 28
GPS Global Positioning System 9, 13, 15
GPSR Greedy Perimeter Stateless Routing 11, 12, 24

GPU graphics processing unit 30
GUI graphical user interface 30, 31

HDE highest-degree node election 54, 67, 101, 102
HFR Hybrid Forest Routing 54, 63, 64, 66, 67, 76, 81–83, 92–101, 106–113, 126
HKE highest-key node election 54, 67, 85, 88, 96, 97, 100–103, 106, 108, 112
HPC high-performance computer 36–38, 84

IG Interactive Growth 22
IGP interior gateway protocol 2
IP Internet Protocol 3, 22
ISP Internet service provider 1, 2
ITM Interdomain Traffic Matrix 23

JUNG Java Universal Network/Graph Framework 30, 38

LAN Local Area Network 1
LB load balancing 73
LBFR Load Balanced Forest Routing 54, 59–64, 67, 83, 88–92, 96, 104, 105, 113
LBLSP Load Balanced Local Shortest Path 25
LSP Local Shortest Path 25

MAC Media Access Control 40

PFP Positive-Feedback Preference 22
PIC Practical Internet Coordinates 17
PSO particle swarm optimization 16, 43
PSVC Particle Swarm Virtual Coordinates 16, 40

RCC rich-club connectivity 21, 22
RIB Routing Information Base 3
RM redundant mode 54, 68, 69, 71–73, 80, 85, 88, 92, 97, 100–106, 108, 111, 112
RNG relative neighbourhood graph 12
RTP Routing with Position Trees xx, 19, 40, 51–54
RTT round-trip time 17

S4 Small State and Small Stretch Routing Protocol 17
SA simulated annealing 43–45

SALT Simulated Annealing Label Trees xix, 35, 40, 41, 43, 45, 47, 48, 53, 67, 113

TCGR Tree Cover Based Geographic Routing with Guaranteed Delivery 58

UDG unit-disk graph 8, 15, 18, 20, 24–28, 39, 42, 47, 50, 92, 96, 116

VCap Virtual Coordinate assignment protocol 17

VPCR Virtual Polar Coordinate Routing xix, 18, 26, 48, 49, 52, 53

VPCS Virtual Polar Coordinate Space 26

WDG Weighted Distance Gain 25

WSN wireless sensor network 5, 8, 15, 20, 24–26

Chapter 1

Introduction

The Internet is becoming an indispensable communication medium for people, businesses and government institutes.[1] In a world where information is being created and processed at an unseen speed, communication on a global scale is a necessity. As a result the Internet is more and more being thought of as a commodity good, much like electricity. The super-linear trend in growth to be witnessed today imposes high routing scalability requirements. Only the Internet is built on design principles from 40 years ago when the number of network gateways was multiple orders of magnitude lower.[2] Therefore a dire need exist for more scalable and flexible routing mechanisms to prevent the Internet from bursting at the seams.

1.1 The Internet architecture

Much of the Internet communication takes place under the form of data transfer between source-destination host pairs. This data is transferred by physical lines after it has been divided into small packets. However, packets are not sent directly to their destination. They first traverse a set of intermediary devices that are thought to close the distance to the end-point. These devices are termed *routers*. In this fashion packets traverse multiple *hops* before reaching their target. To be able to support the massive data transfer demands, a well-built Internet architecture exists. It can be represented as number of hierarchical layers, each at a different level of granularity, as shown in Figure 1.1. At the most detailed level reside small Local Area Networks (LANs). This type of network interconnects different general purpose computing devices via a wired or wireless transmission medium and spans a small geographical location such as a house or a company.

At a slightly higher level the *access networks* are encountered, connecting Internet users to their Internet service provider (ISP). Next there are the so-called *aggregation networks*. From this layer on, powerful routers are necessary to forward traffic to its destination. At the heart there are the *core networks*, which the Internet *backbone network* is a part of, interconnecting various independent networks, called Autonomous Systems (ASs). They are autonomous in the sense that they are being managed by separate administrative units. Rekhter *et al.* (2006) define an Autonomous System more formally as

[1] Zahariadis *et al.* (2011)
[2] Pan *et al.* (2011), Zahariadis *et al.* (2011)

> *A set of routers under a single technical administration, using an interior gateway protocol (IGP) and common metrics to determine how to route packets within the AS, and using an inter-AS routing protocol to determine how to route packets to other ASs.*

By connecting multiple ASs in a peering relationship we obtain a network of networks, an internetwork, hence coined the "Internet". Each of these ASs is assigned a unique number, its *AS number*. At the time of writing, there are nearly 45.000 unique ASs.[3] It is packet forwarding, or *routing*, in this backbone network that forms the main point of focus of this thesis.

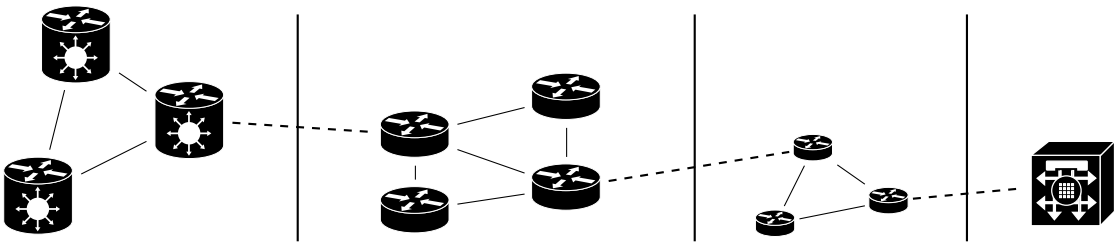


Figure 1.1: A schematic overview of the Internet architecture. From left to right can be seen: backbone network, aggregation network, access network and LAN. The dashed line crossing the vertical solid line represents the peering relationship between the different types of networks.

The previously mentioned core networks can in their turn be further divided into a hierarchical set of tiers. On the lowest level there are the *tier-3 networks* consisting of fixed or wireless networks spanning a specific geographical area such as a neighbourhood. It is said that these networks are engaged in a peering relationship with *tier-2 networks*. In such a peering relationship both networks are typically connected by a router, here called a *border gateway*. These act as transit gateways for traffic flows between both networks, possibly applying traffic conversions. The highest level is formed by the Internet backbone, an aggregation of *tier-1 networks*. Only a few ISPs control these networks that span large geographic areas.[4]

To be able to reason about novel routing procedures, the structure of the Internet backbone has to be captured. Thus a remaining question is what the topology of the inter-AS network looks like. Researchers have investigated this by retrieving routing tables stored by each border gateway.[5] Though it should be noted that some of the extracted data is skewed as ASs might use multiple AS numbers to identify themselves. The topology appears to be scale-free.[6] In such networks the degree distribution follows a power-law $P(k) \sim k^{-\gamma}$ with a parameter γ and k the number of edges incident to a certain vertex.[7] This gives rise to some interesting properties such as the most connected AS being connected to 10% of the total network, which can have a negative effect on routing fault-tolerance, its power to withstand node or link failures. Disconnection of high-degree nodes might partition the network. Other findings are that the most connected nodes seem to cluster and form a nearly complete graph in which the shortest paths are relatively small, called the *small-world* phenomenon.[8] Networks with this emerging property are called small-world

[3] Bates *et al.* (2013)
[4] Tavernier (2012)
[5] Chi *et al.* (2008)
[6] Newman (2003)
[7] Goh *et al.* (2002)
[8] van Steen (2010)

networks and are characterised by the fact that they have a small network diameter.[9] Therefore this work will target scale-free networks of varying size.

1.2 The Border Gateway Protocol

Each AS in the Internet is responsible for a number Internet Protocol (IP) routing prefixes and is uniquely identified by an Autonomous System Number (ASN). Each AS owns one or more border gateways that are connected in a peering relationship with other gateways. These devices route traffic within this interconnected structure based on the Border Gateway Protocol (BGP). Neighbouring ASs discover each other by exchanging update messages containing (AS-number, network ID)-pairs. By doing this, gateways are able to learn new paths to other ASs. To illustrate this: $\texttt{AS}_i$ sends the pair $\texttt{(AS}_i\texttt{, network\_id}_i\texttt{)}$ to a neighbouring $\texttt{AS}_j$, the latter will gain knowledge about the path between $\texttt{network\_id}_i$ and itself. Now the receiving router can add this information to its Routing Information Base (RIB). This is a table containing routes to each destination. Contrary to *routing tables* the RIB may contain multiple routes to other ASs. However in many cases only the shortest route is stored while others are discarded as the tables would become excessively large.[10] Although storing multiple routes is a possibility to enforce some form of load balancing of the inter-AS traffic.[11] Routing tables are used to select the next hop in the forwarding process based on the destination network prefix. The BGP does not broadcast its full routing table to other routers. It is a distance-vector protocol based on update messages where only new or altered path information needs to be send. Thus, still using the same example, the receiving border gateway is free to distribute this newly discovered path to its own neighbours by sending a triple $\texttt{(AS}_j\texttt{, AS}_i\texttt{, network\_id}_i\texttt{)}$.

Though being a fundamental Internet protocol, the BGP copes with several issues.[12] First of all, as the number of ASs is increasing linearly[13], the number of entries in the BGP routing tables increases more than cubically as each node needs to store information about many possible routing paths in the network. This effect can be seen in Figure 1.2 where the number of routing prefixes has been set out in function of the time. A prediction can be made by plotting a best-fitting curve on top of this data. This expansive growth poses a severe scalability problem for the Internet as a whole, because of the increasing router memory storage requirements which form a bottleneck for the forwarding layer.[14] When the Internet was first introduced, it was never intended to be able to deal with such a high number of ASs. Therefore there is a dire need for a more scalable routing approach that includes favourable characteristics such as automatic load-balancing and scalability, a system that truly makes use of the distributed properties of the Internet.

Another problem is that of path instability. Routing tables need to be continuously adjusted to reflect changes in the network. Examples of this are links failing, nodes joining or routers rebooting.

[9] Amaral *et al.* (2000)
[10] van Steen (2010)
[11] Bu *et al.* (2004)
[12] Narayanan (2009)
[13] Another good fit is an exponential trend with a very low exponent value (0.03), which is —as indicated by Dhamdhere & Dovrolis (2011)— very close to linear.
[14] Papadopoulos *et al.* (2010)

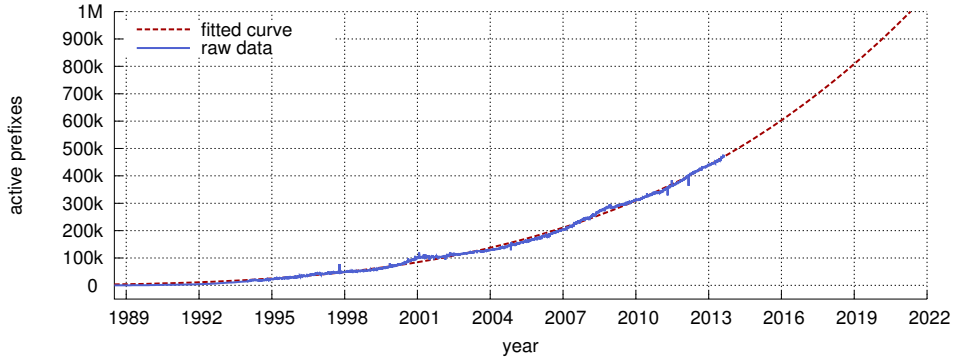


Figure 1.2: The BGP AS router prefix size is plotted in blue. The dotted red line is a fitted curve through this data that shows the trend; the fitted curve is $f(x) = a(x-b)^{\gamma}$ with x the year, $f(x)$ the BGP AS prefix entry count; $a = 0.867391$, $\gamma = 3.73411$, $b = 1979.31$.

This can cause routes to *flap*, which means that they are continuously added and removed from the BGP tables. Route flapping damping mechanisms are implemented to counter this effect. However, these have a negative effect on the BGP convergence time, the time it takes for all gateways to obtain a consistent view of the network through their routing tables. It has been shown that this BGP convergence time is in the order of minutes.[15] Another reason for this high convergence time is its path exploration procedure. When a link fails that causes some destination to be cut off from a certain router, this router will start exploring alternative paths in order to reach this destination. Only when all alternative paths have been explored will it decide which path to store in its routing table, or announce that it is unable to reach the destination, which can take a large amount of time. A novel routing mechanism should therefore be able to converge sufficiently fast.[16]

Automatic load balancing is a third issue. The BGP has only limited support to balance traffic over multiple links in order to avoid link congestion. Manual configuration is possible by the network administrator by means of routing policies, but this is tedious and prone to misconfiguration.[17] For example: assume a gateway connected to two other gateways that are connected to a different part of the network. This first gateway may be set-up to balance load by splitting up the advertised prefix, which can be done by applying a different subnet mask, e.g. by splitting `1.2.3.4/5` into two parts `1.2.3.4/6`, each with their own set of ASNs. Although this leads to load balancing, several issues arise: (i) the number of routes advertised increases; (ii) the number of entries in the forwarding table increases; (iii) load is balanced in a static way, there is no adaptive behaviour; (iv) it is very complex to set-up correct load balancing policies.
When BGP has equal-cost multi-path (ECMP) capabilities, load balancing is not achieved by splitting prefixes, but rather by storing multiple paths in the forwarding table. Traffic can now be dynamically balanced over these multiple routes. A downside however is the increase in forwarding entries, which is already a weakness of BGP.[18]

[15] Oliveira *et al.* (2009)
[16] Bürkle (2003)
[17] Caesar & Rexford (2005)
[18] Halpern & Jakma (2006)

1.3 Geometric routing

As previously mentioned, the current BGP will face great difficulties in supporting the vast growth of the Internet. Hence there is exists a large research interest in alternative routing systems that are scalable by nature. Over the years a new routing paradigm has emerged, termed *geometric routing*. Herein every network, represented by a graph, is embedded into a mathematical space. In this *graph embedding* every vertex is assigned a specific set of coordinates that will serve as a basis for routing decision making. Routers will use the coordinates that uniquely identify each distinct node to choose what node to forward traffic to. To do this they make use of a distance metric present in the space. For example, the space may be the two-dimensional Euclidean space represented by coordinates in $\mathbb{R}^2$ with a distance metric corresponding to the Euclidean distance between two point represented by two sets of coordinates. Every router will then try to send traffic to one of its neighbours that is closer to the destination in terms of distance in the metric space. This behaviour is depicted in Figure 1.3.

The roots of geometric routing trace back to ad-hoc wireless networks and wireless sensor networks (WSNs) in which each routing unit (sender-receiver) has limited battery power. Thus the forwarding decisions should be computationally inexpensive and have low state requirements. Because much of the current research still focuses WSNs, which can be modelled as unit-disk graphs[19], its application to wired networks is still in its early days.[20] Therefore the application to networks such as the inter-connected system of ASs poses an interesting research topic. Especially when concerning the portability of its scalable properties from unit-disk graphs to scale-free networks.

In this thesis the possibility of using geometric routing as an alternative to BGP routing will be investigated. As most of the geometric routing systems come forth from WSNs, algorithms targeting this type of networks will be examined first. In a later stadium, the applicability to other types of graphs that resemble the Internet backbone more accurately will be researched. Introducing geometric routing in the Internet backbone may solve memory scalability, but it may lead to other problems such as the volatility of coordinates which also act as host identifiers. The main contribution of this this is a set of geometric routing algorithms that focus on load balancing, low average path length, fault-tolerance and scalability.

1.4 Reader's guide

This work is structured as follows:

- **Chapter 1** introduces the general concepts that are inside the scope of this work.
- **Chapter 2** reviews the current research status regarding geometric routing and related concepts. Strengths and weaknesses of various methods are weighed up.
- **Chapter 3** explains the preliminary research that led up to the final routing mechanism.

[19] These are networks that generally have a large diameter and a low constant average node degree.
[20] see Chapter 2

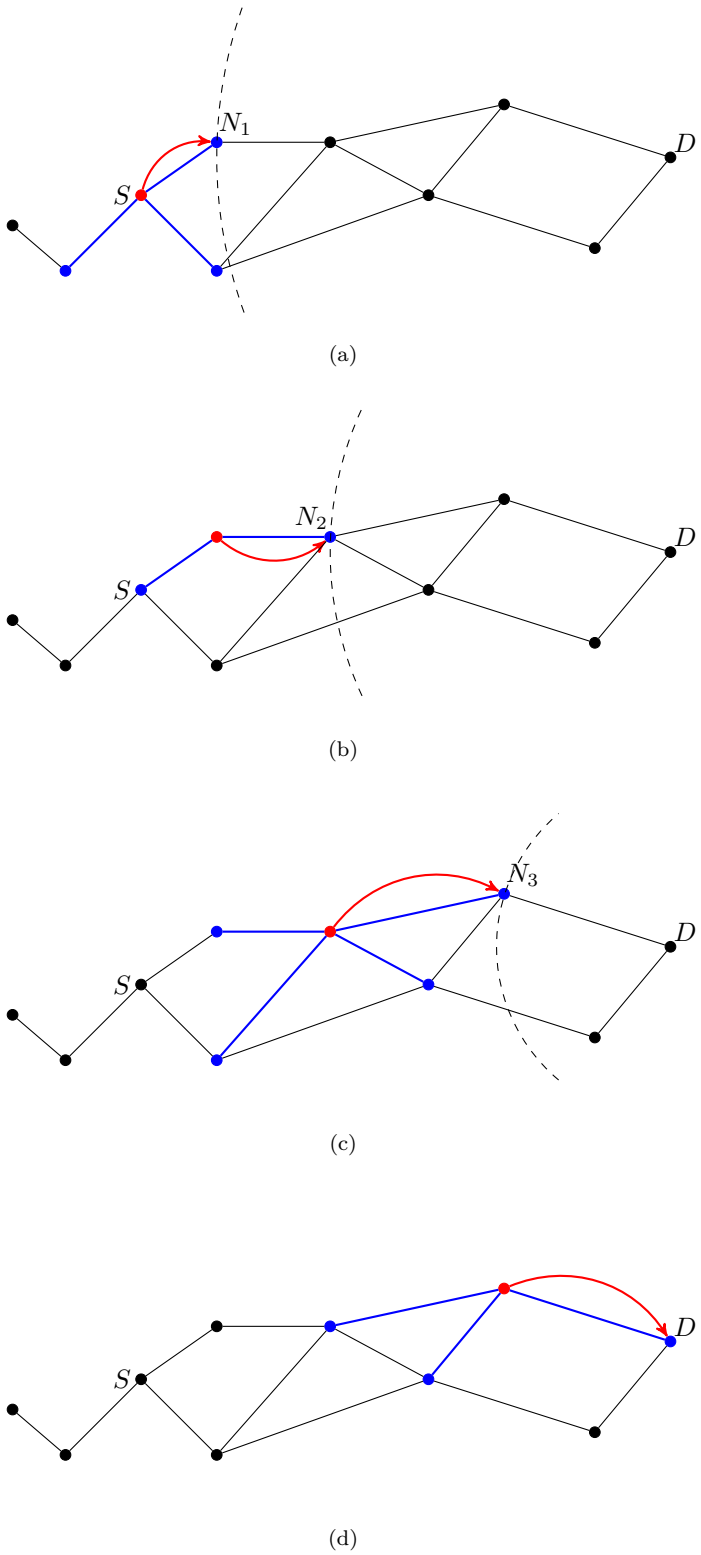


Figure 1.3: Geometric routing in a two-dimensional Euclidean plane based on the Euclidean distance: S is the source node; D is the destination node. The current node is depicted in red while its possible neighbours (and links to them) are painted in blue. From (a) to (d) a message traverses the network as each node tries to send the message as far as possible to D.

- **Chapter 4** explains the main contribution of this thesis, a geometric routing mechanism targeting the Internet backbone.
- **Chapter 5** shows the results of conducted experiments and discusses them.
- **Chapter 6** gives a conclusion of this thesis and proposes future work.

In Table 1.1 the most commonly used symbols in this thesis are summed up along with their meaning. These symbols are used as consistently as possible throughout this thesis to avoid any confusion.

Table 1.1: Commonly used symbols in this thesis

symbol	meaning
$\bar{\star}$	average
$\sigma_\star$ or σ	standard deviation
$\lvert \star \rvert$	set size or absolute value
ρ	stretch
β_V or β_E	node or link β-ratio for load balancing
$\beta_{V,\Pi}$ or $\beta_{E,\Pi}$	node or link β-ratio for load balancing for shortest path Π
λ_V or λ_E	node or link λ-ratio for load balancing
$\lambda_{V,\Pi}$ or $\lambda_{E,\Pi}$	node or link β-ratio for load balancing for shortest path Π
$\delta(u, v)$	distance between u and v in metric space $(\star, \delta)$
τ	τ-ratio for tree redundancy
$\mathbb{E}^d$	d-dimensional Euclidean space
$\mathbb{H}^d$	d-dimensional hyperbolic space
$G = (V, E)$	graph G with vertex set V and edge set E
$T = (V, E)$	tree T with vertex set V and edge set E
$N(u)$	set of adjacent vertices of u
$I(u)$	set of incident edges of u
$\Pi(u, v)$ or Π	shortest path (between u and v)
$P(u, v)$ or P	path $(u, \ldots, v)$
$d_G(V)$	generic degree in graph G of a vertex $v \in V$
$\delta_\Pi(u, v)$	shortest path distance (hops) between u and v
d	graph diameter
h	hop count
$\mathcal{P}$	set of all possible paths in a certain graph
α, β, γ	commonly used as tunable parameter
ξ	link propagation delay bounds
μ	node processing time bounds
$\mathcal{K}(u)$	key of u, a unique identifier of u taken from a set $\mathcal{K}(V)$
t_{proc}	node processing delay
t_{init}	procedure initialisation delay
t_{root}	root election delay
t_{prop}	link propagation delay

Chapter 2

Background

This chapter introduces the concepts of geometric routing. To be able to frame this work in a larger global scientific view, related work concerning geometric routing is investigated. Nevertheless the existence of an ample body of work dealing with such routing mechanisms, much research is yet to be devoted regarding their application to wired networks. Current researcher's main point of focus remains WSNs, interconnected sets of radio sender-receivers with a fixed transmission range. Contrary to wired networks WSNs can be accurately modelled as unit-disk graphs (UDGs). On top of that, due to their wireless nature, an extra mobility factor has to be taken into account as nodes are able to move freely in space, a property that may be neglected for wired networks. The following sections will set forth a solid theoretical base for geometric routing, rooted in these WSNs. Afterwards the application to wired networks, and ultimately the Internet backbone, will be examined. The trade-off between routing scalability, load balancing and average path length will play a central role.

2.1 Physical coordinates

Geometric routing originally emerged from the search for efficient routing mechanisms in large-scale grid networks.[1] However since then it has been generally applied to WSNs.[2] Herein every node is a sender-receiver beacon with a limited radio transmission range. The nodes are scattered over a geographic area. This allows the network to be modelled as UDGs (see Definition 2.1) with a unit corresponding to this transmission range (this is depicted in Figure 2.1).

Definition 2.1. *A graph $G = (V, E)$ is a unit-disk graph (UDG) if every node is assigned a set of two-dimensional Euclidean coordinates and $\forall u, v \in V : v \in N(u) \Leftrightarrow \delta(u, v) \leq 1$ with δ the Euclidean distance between two points.*[3]

[1] Finn (1987)

[2] The earliest modern applications found were those of Karp & Kung (2000) and Karp (2000). Although Finn (1987) applied greedy routing to wired networks, they were not modelled as scale-free networks. From what was described in this article it can be concluded that the author also based his research on grid networks, which have much in common with UDG-like networks, such as a constant node degree, a large diameter and its ability to be embedded into a two-dimensional Euclidean plane.

[3] Kuhn *et al.* (2004)

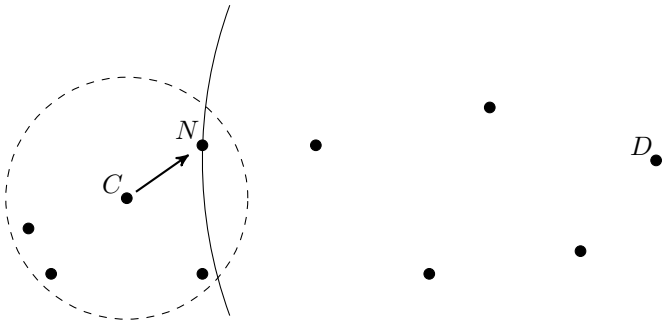


Figure 2.1: A network of sensor nodes: the dashed line represents C's radio range, the solid line is the set of points equidistant to D. Node C may send its traffic to node N when trying to close as much distance as possible to destination node D.

Because of their limited transmission range, the sensor devices rely on each other to forward data packets across the network. A logical approach is that every node tries to send a packet as far as possible in the direction of the destination node. Thus the notion of distance naturally emerges from this type of routing schemes. To be able to send a packet closer to its destination every node needs to know the location of its reachable neighbours. Therefore either Global Positioning Systems (GPSs) were added to the nodes or its coordinates were manually inserted. Nodes could thus gain knowledge of their own position. By means of message passing nodes inform their neighbours about their position. Upon receipt of such coordinate information messages, they generate a small routing table composed of an identifier and the received set of coordinates. As every has node only a limited set of neighbours unrelated to the network size (when considering a constant density), the storage requirements are limited. This is an important observation as most sensor devices only have a limited battery lifetime and are thus restricted by low computational and storage complexity.

To conceptualise, a geometric routing mechanism is simply the assignment of coordinates in a certain space to network nodes. In the previous example these are mostly Cartesian coordinates $\mathbb{R}^2$. To decide which neighbour to forward a packet to, the destination coordinates have to be sent along with the packet itself. And as each node has its own set of coordinates, these function as unique node identifiers. Routers inside the sensor devices will calculate their own distance based on a simple Euclidean distance metric. If node c has coordinates (c_0, c_1) and the destination d has coordinates (d_0, d_1) the distance is defined as the distance between the two points corresponding to these coordinates in the Euclidean two-dimensional space:

$$\delta(c, d) = \sqrt{(c_0 - d_0)^2 + (c_1 - d_1)^2} \tag{2.1}$$

Each router will inspect its routing table, consisting of identifier-coordinates doubles of neighbouring nodes, and select the neighbour that has a distance (to the destination) fulfilling a certain selection criterion, e.g. the lowest distance to the destination. Most of these original routing mechanisms only allow distance-decreasing paths, which makes it easier to guarantee packet delivery.

The method for selecting the next forwarding hop of a packet (routing decision making) leaves much freedom. Lots of the earliest versions of geometric routing were termed as *greedy routing*.

Here every path is distance-decreasing. Traffic is routed to the neighbour with the lowest distance to the destination node, based on Eq. (2.1), called *greedy forwarding*. This is also illustrated in Figure 2.1. The following definition defines this term more formally.

Definition 2.2. *In a graph $G = (V, E)$ with a given distance function $\delta : V \times V \to \mathbb{R}^+$, greedy forwarding entails the following decision: given a destination node d, a node u with neighbours $N(u)$ forwards a message to its neighbour $v \in N(u)$ such that $\delta(v, d) = min_{w \in N(u)} \delta(w, d)$.*

Although greedy forwarding is one of the most-used neighbour selection mechanisms, it is not the only possibility. Other selection variants exist, as shown in Figure 2.2 with each their own specific advantages, according to the situation they are used in.

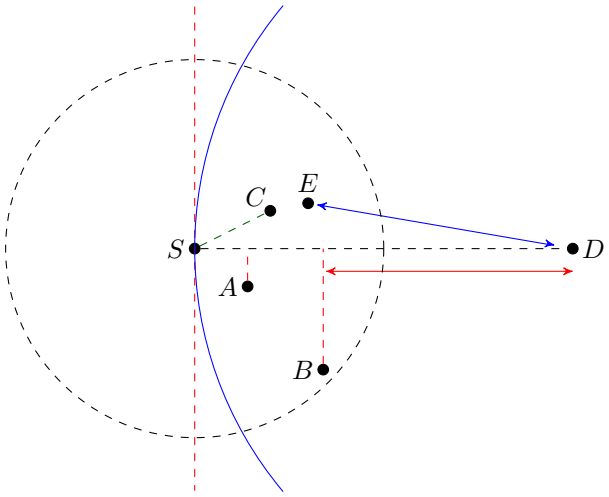


Figure 2.2: Different types of geometric next hop selection mechanisms: a packet arriving at S can be forwarded to multiple neighbours within its transmission range. Node A: *nearest with forwarding progress*, B: *most forwarding progress within radius* by Takagi & Kleinrock (1984), C: *compass routing II* by Kranakis *et al.* (1999) and E: *greedy routing*.

All the routing types in Figure 2.2 have one thing in common: packets always have to travel along a distance-decreasing path to the destination. This reveals a downside of geometric routing: its susceptibility to *dead ends*, also termed *voids*. Ideally while traversing the path from source to destination every node should be able to find a neighbour with a lower distance to the destination than itself. However, it is possible that no such neighbour exists. In this case the routing of a packet gets stuck which causes it to be dropped. Another way of saying this is that a distance *local minimum* has been reached, which can be formally defined by Definition 2.3. Figure 2.3 shows such a local minimum for an embedding into a two-dimensional Euclidean space with the distance function δ being the Euclidean distance. Also note that in case forwarding progress was used as a selection criteria (see Figure 2.2), the packet would not encounter a void. This is indicated by the dashed red lines in Figure 2.3.

Definition 2.3. *For a graph $G = (V, E)$ with a given distance function $\delta : V \times V \to \mathbb{R}^+$ and a destination vertex $d \in V$, a vertex $u \in V$ is a considered to be a local minimum when $\nexists n \in N(u) : \delta(n, d) < \delta(u, d)$, with $N(u) \subseteq V$ being the set of neighbours of u.*

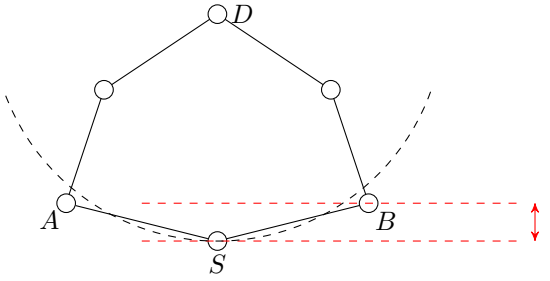


Figure 2.3: A local minimum in the distance landscape for greedy routing: both of S its neighbouring nodes A and B are further away from the destination D than S itself. The red dotted line indicates that when using forwarding progress as a selection criterion there is no local minimum at S.

To elevate the issue of voids many algorithms have been developed to use in conjugation with geometric routing. A pioneering mechanism is called *face routing*, based on the work of Kranakis *et al.* (1999), in which it was called *Compass Routing II*. It was first introduced in a routing mechanism called Greedy Perimeter Stateless Routing (GPSR).[4] GPSR has the ability to route packets in two distinct *routing modes*: *greedy mode* and *face mode*. Packets are forwarded towards their destination, whose coordinates are encoded in the packet header, by following a distance-decreasing path with a greedy selection mechanism. Whenever a void is reached in a node v, the routing system switches to face mode for that particular packet. It also records the current distance to the destination $\delta(v, d)$, at which it got stuck in the local minimum. Face mode remains active until a node w is reached whose distance $\delta(w, d)$ is lower than the distance $\delta(v, d)$ recorded in the packet header. At that exact moment greedy mode is re-entered, because the packet has overcome the void. Figure 2.4 illustrates the mechanics of face routing. While in face mode, packets are send in a counter-clockwise fashion without crossing the direct line between source and destination, which can be calculated solely based on the coordinates of both nodes.

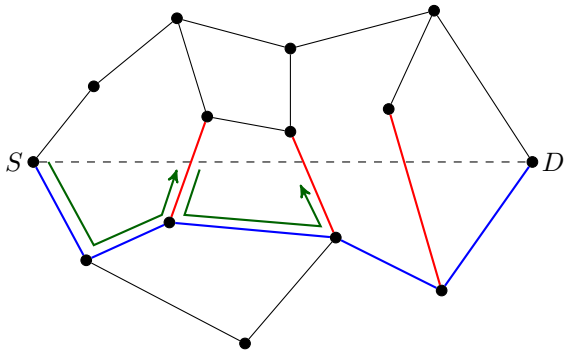


Figure 2.4: Face routing: the blue lines represent the path followed while using face routing, the red lines are edges crossing the straight line between source S and destination D and the green arrows represent the direction of face routing on each face.

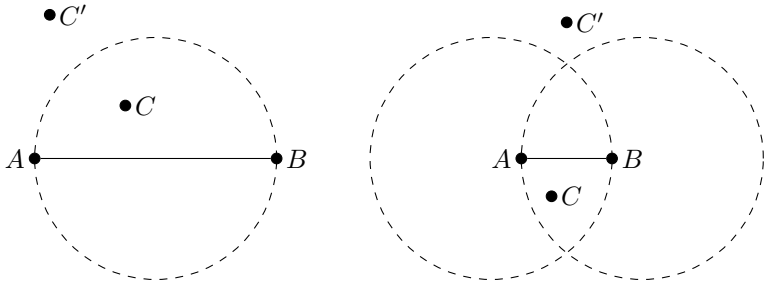
A limiting factor of face routing is that it solely works on planar graphs. Such a graph has

[4] Karp & Kung (2000)

the property that it can be embedded into a plane, while none of its edges cross. As realistic graphs rarely possess the virtue of planarity, the graph needs to be transformed. The GPSR system achieves this planarization by calculating a Gabriel graph (GG) (see Definition 2.4 and Figure 2.5(a)) or a relative neighbourhood graph (RNG) (see Definition 2.5 and Figure 2.5(b)) that is a sub-graph of the original graph, in a distributed fashion. Face routing is then exercised on this sub-graph. Although the authors report high success rates, an issue raised by Bruck *et al.* (2005) and Funke & Milosavljevic (2007) is the load balancing aspect of this mechanism. Face routing concentrates traffic heavily around the void perimeters, creating severe traffic bottlenecks.

Definition 2.4. *A graph $G = (V, E)$ is called a Gabriel graph if for all vertices u and v there exists no vertex w that resides inside the circle going through u and v with diameter $\overline{uv}$.*

Definition 2.5. *A graph $G = (V, E)$ is called a relative neighbourhood graph if an edge (u, v) only exists between vertices u and v if $\delta(u, v)$ is less than or equal to the distance between every other vertex w, and whichever of u and v is farther from w. $\forall w \neq u, v : \delta(u, v) \leq \max\{\delta(u, w), \delta(v, w)\}$.*



(a) Gabriel graph: A and B cannot be connected if there exists a point C within the dashed circle. A point C' outside the circle poses no problem.

(b) relative neighbourhood graph: A and B can only be connected if there exists no point C in the intersection of the two dashed circles. A point C' outside the circle poses no problem.

Figure 2.5: Gabriel graph and relative neighbourhood graph: the dashed circles have an arbitrary but fixed radius.

Another system using greedy forwarding combined with face routing is GOAFR$^+$ by Kuhn *et al.* (2003), which is quite similar to GPSR. Their system differs from GPSR in the way that it switches between greedy and face routing. It explores each face until sufficient[5] face nodes have been discovered. From this set the closest one gets selected as the next hop. The system is proven to be asymptotically optimal in a worst-case setting and efficient on average case graphs.

2.2 Virtual coordinates

2.2.1 General concepts

The most prominent downside of using physical coordinates for geometric routing is that they may be unavailable, or unrelated to the network topology. It is simply not guaranteed that every

[5]Exact criteria can be found in Kuhn *et al.* (2003).

routing device possesses a GPS-like system. Because of this, many researchers have investigated the use of *virtual coordinate systems*. These virtual coordinates can be seen as a generalisation of the physical ones. The following definition explains virtual coordinates and its properties, as defined by Goodrich & Strash (2009).

Definition 2.6. *Let Σ be an alphabet that gives forth a set of finite-length strings σ. A coordinate system f over a space S can be defined by the following characteristics:*

- *f is a mapping $f : \sigma \to S$.*
- *f can be parametrised, assigning string of σ may depend on a fixed set of parameters.*
- *f is oblivious: f may only depend on the parameters of f itself and the string $x \in \sigma$ it assigns. It may not be dependent on other strings of σ or other points in S.*

As such virtual coordinates are a more general concept than physical coordinates. They can be manipulated more freely. It is possible, in some occasions, to eliminate the appearance of routing voids rather than avoiding them by assigning the coordinates in a special way. This is one of the most important aspects of geometric routing, namely achieving 100% packet delivery rate. This special way of assigning coordinates is called a *greedy graph embedding*. First, a graph embedding is the assignment of coordinates to each node of a graph as the following definition explains.

Definition 2.7. *Let S be a set and $G = (V, E)$ a graph, then an embedding of G into S is a mapping $f : V \to S$ such that $\forall u, v \in V : u \neq v \Leftrightarrow f(u) \neq f(v)$.*[6]

In a greedy embedding of a graph $G = (V, E)$ coordinates are assigned to vertices in a coordinate space such that for each pair of vertices s and t there exists a neighbour of s closer to t than s itself. Formally this can be defined by as follows.

Definition 2.8. *A greedy embedding of a undirected graph $G = (V, E)$ into a metric space (S,δ) (see Definition 5.1 for the definition of a metric space) is a mapping $f : V \to S$ with the following property: for every pair of distinct vertices $s, d \in V$ there exists a vertex u adjacent to s such that $\delta(f(u), f(d)) < \delta(f(s), f(d))$.*[7]

This essentially means that coordinates are assigned in such a way that there exists a neighbouring node that is on a distance-decreasing path to the other, for every two nodes. From this follows that there exist no voids, thus a packet will never get stuck in a local distance minimum. This will prove to be an important concept, as packet delivery is of utmost importance for the Internet backbone. The construction of such a greedy embedding is not possible for every space. For example the following $K_{1,7}$ star graph shown in Figure 2.6 has no greedy embedding in $\mathbb{R}^2$.[8]
Furthermore Papadimitriou & Ratajczak postulated the following theorem.

Theorem 2.1. *The bipartite graphs $K_{1,7}, K_{2,13}, \ldots, K_{r,6r+1}$ admit no greedy embedding into the Euclidean plane $\mathbb{E}^2$.*

[6] Rao *et al.* (2003)
[7] Kleinberg (2007)
[8] He & Zhang (2011)

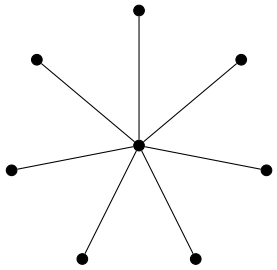


Figure 2.6: Greedy drawing of $K_{1,7}$ not possible in $\mathbb{E}^2$ space

Determining properties about whether graphs are greedily embeddable in a certain space is a much researched topic. For example Leighton & Moitra (2010) recently proved the Papadimitriou & Ratajczak conjecture[9] that every three-connected planar graph can be embedded in $\mathbb{R}^2$. This is important as any k-connected planar graph with $k > 3$ has a 3-connected sub-graph. A limitation however is that this holds only for planar graphs. Kleinberg (2007) compensated this shortcoming by proving the following theorem.

Theorem 2.2. *Every connected finite graph has a greedy embedding in the hyperbolic plane* $\mathbb{H}^2$.

This theorem will later prove to be important when utilising structured embeddings (see Section 2.2.3). The hyperbolic plane $\mathbb{H}^2$ will be explained in more detail later on in Section 4.2.2. Because geometric routing systems forward traffic based on a distance function, the paths generated are not necessarily shortest paths. Therefore it is important to know how much the routing system deviates from the shortest path routing scenario. Therefore an important concept is the *stretch* of a routing system.

Definition 2.9. *The stretch* ρ *of a path* $P(s,d)$ *in a graph* $G = (V,E)$ *with a start vertex* $s \in V$ *and an end vertex* $d \in V$*, is its number of vertices* $|P(s,d)|$ *divided by the number of vertices* $|\Pi(s,d)|$ *with* $\Pi(s,d)$ *the shortest path between* s *and* d.

This means that $\rho \in \mathbb{R} \wedge \rho \geq 1$. A stretch of one occurs when the path $P(u,v)$ is a shortest path between u and v, denoted as $\Pi(u,v)$. The stretch ρ thus expresses how much the observed routing system approximates a shortest path routing mechanism.

Based on the literature research done for this thesis, it became clear that virtual coordinate systems can be divided into two types. The first type is systems based on *iterative embeddings*, which will be explained in the next section. These iterative embeddings have in common that they are constructed by iteratively altering the assigned coordinates to optimise some sort of objective function. The second type of system is based on *structured embeddings*. In these embeddings coordinates are assigned by following a fixed underlying structure, mostly a spanning tree.

2.2.2 Iterative graph embeddings

Earlier generations of virtual coordinate-based geometric routing systems try to assign coordinates in such a way that they approximate the underlying physical ones. These systems came forth

[9] Papadimitriou & Ratajczak (2005)

from wireless ad-hoc networks or WSNs which had no node GPS-system available. Especially in a distributed setting with no node conveying the grand network topology, mimicking the underlying structure can be troublesome. Couto & Morris (2001) began to eliminate the need for nodes to have knowledge of their exact whereabouts by only requiring that certain *proxy nodes* have the ability to pin-point their exact location. Other nodes use the location information of their closest proxy node as their own. A downside however remains the requirement for location-awareness of —although only a limited number— nodes. Routing between local nodes and their proxies happens in a link-state way. Geometric routing is used solely for inter-proxy routing.

The physical coordinate proxy mechanism presented by Couto & Morris (2001) allows for the detection of voids in the network. This is done by requiring nodes to inform the source whenever they drop a packet due to a local distance minimum. Upon receipt of such a message, the sending proxy selects an intermediate node that resides within a certain radius from the middle between source and destination. Until the packet has been correctly received by the destination, the sending proxy keeps on increasing the radius hoping to find an intermediate node outside of the void range. This system however guarantees no 100% delivery rate.

The previous system inspired Rao *et al.* (2003) to develop one of the first geometric routing techniques entirely based on virtual coordinates, later nicknamed *NoGeo*. It is one of the earliest in its kind and is thus used by many as a benchmark for testing the behaviour of newer routing mechanisms.[10] Their system first identifies perimeter nodes, these are nodes that lie on the border of the network.[11] It does this by electing the node that has the largest shortest path to all previously detected perimeter nodes, in an iterative fashion.[12] This shortest hop count distance is based on the triangulation inequality, much used in the distance estimation between nodes. After having identified the perimeter nodes, they are assigned a set of virtual two-dimensional Euclidean coordinates. Hereafter an iterative relaxation algorithm is applied to minimise the error E between the measured shortest path distance and the distance measured in the virtual coordinate space, as shown by the following equation.

$$E = \sum_{u,v \in P} \left(h(u,v) - \delta(u,v)\right)^2 \tag{2.2}$$

with P the set of perimeter nodes, δ the Euclidean distance function and h the shortest path distance function. After the perimeter nodes have their coordinates assigned, the other nodes are assigned theirs based on local neighbourhood information. The perimeter node coordinates set forth a surge of coordinate assignments towards the core of the network in which each node u calculates its coordinates by

$$u = \frac{\sum_{v \in N(u)} v}{d_G(u)} \tag{2.3}$$

with $d_G(u)$ the degree of node u. The authors report routing success rates comparable to routing based on physical coordinates. The only downside is the need for a high number of relaxation

[10] e.g. Sarkar *et al.* (2009), Leong *et al.* (2007), Zhou *et al.* (2010)
[11] It is still implicitly presumed that the underlying graph is a UDG.
[12] Ties are broken in an arbitrary manner, such as node identifiers.

iterations. Although it should be noted that this only needs to be done on at the embedding construction time, as such it will not hinder routing performance. When building this virtual embedding for a network that has physical coordinates available, a mediocre resemblance between the two embeddings (virtual and physical) can be noticed. This results in mediocre routing success rates. Other drawbacks are reported by Zhou *et al.* (2010): (i) the greedy success rate is low for sparse networks while the stretch is high; (ii) the storage requirements are high as every perimeter node has to store the hop count to every other perimeter node, resulting in a $O\left(p^2\right)$ complexity with the number of perimeter nodes p of the order of $O\left(\sqrt{|V|}\right)$ for a graph $G = (V, E)$. Leong *et al.* (2007) report that NoGeo also performs poorly (i) when the network has a high number of voids and (ii) when the number of nodes is high.

Later Leong *et al.* (2007) designed a similar routing system called GSpring. It assumes non-mobile nodes and behaves as a simulation of a set of physical springs. Each edge of the graph represents a unique spring, pulling the nodes it connects towards each other based on Hooke's law. Much like NoGeo, first a set of perimeter nodes is elected. Afterwards these are mapped onto a circle, whereafter nodes lying on a shortest path to these perimeter nodes are assigned interpolated coordinates in a distributed fashion. Then, each non-perimeter node calculates a coordinate set based on the coordinates of its neighbours. It is exactly this calculation method that is based on a force-directed spring layout. After the initial coordinate assignment phase, a so-called *region of ownership* is determined for each node. Based on Theorem 2.3 —of which the proof can be found in Leong *et al.* (2007)— GSpring tries to form a greedy graph embedding.

Theorem 2.3. *A graph embedding into a Euclidean space is greedy if and only if the region of ownership of every vertex does not contain any other vertices of the graph.*

An iterative coordinate update algorithm tries to push each vertex out of its neighbouring vertices' region of ownership, increasing the greedy success rate. To prevent wild oscillation of the updating procedure, a damping mechanism was implemented to assure procedure convergence. The authors report that GSpring increases the face convexity of the network. This results in a decreased greedy failure rate while still maintaining a stretch that is lower than when actual physical coordinates are used. An issue is however the requirement for a *geocast* addressing mechanism which allows the sending of packets to a specific region in the coordinate space. How this mechanism should be implemented is left out from the article.[13]

Another virtual coordinate based routing mechanism is Particle Swarm Virtual Coordinates (PSVC)[14]. Herein an election mechanism is used to determine a constant number of perimeter nodes. Hereafter coordinates are assigned guided by the minimisation of an error function, much like NoGeo. After this initialisation phase, non-perimeter node coordinates are calculated by a particle swarm optimization (PSO)[15] algorithm. This population-based metaheuristic[16] attempts to optimise a criterion by running multiple optimizations independently and then capturing the emerging optimum. Next, coordinates are relaxed in a similar way to GSpring. Simulations show that the

[13] This is also mentioned by Zhou *et al.* (2010).
[14] Zhou *et al.* (2010)
[15] Originally designed by Kennedy & Eberhart (1995)
[16] Contrary to single-solution based mechanisms that iteratively modify a single solution (Talbi, 2009).

system converges faster and attains a lower average stretch than NoGeo. And unlike GSpring, no geocast mechanism is required.

Caruso *et al.* (2005) also break the shackles of requiring physical coordinates. Their system Virtual Coordinate assignment protocol (VCap) selects multiple anchor nodes to which every other node calculates its shortest path hop count. Based on these hop counts, every node computes its own set of three-dimensional coordinates. As multiple nodes may have the same coordinates, *zones* are defined. These zones are used to perform inter-zone geometric routing. Packet delivery within a zone is done in an ad-hoc way (e.g. routing tables). The stretch observed is slightly higher than when using physical coordinates. Also VCap has a major limitation which imposes that every edge in the network should be of the same length, according to sensor nodes with a fixed transmission range. This can however not be guaranteed for wired networks.

A Euclidean graph embedding procedure based on a *big-bang* simulation was proposed by Shavitt & Tankel (2004). Their system simulates an explosion of particles driven by a force field rather than a pile of springs —as in NoGeo-like systems— by positioning all particles at the same position initially. It utilises a potential energy function to assign coordinates. This system has been built to estimate delays between hosts in the Internet, e.g. to be able to select the closest mirror in a multi-server environment, rather than geometric routing. It also makes use of a centralised calculation method, making it unsuited for large-scale networks.

Other graph embedding schemes exist that focus on delay estimation in Internet hosts. One of these is *Vivaldi* by Dabek *et al.* (2004). Its engine is a spring energy minimisation algorithm. Vivaldi relies on inter-host round-trip times (RTTs) and is thus not adequate for providing a graph embedding when no underlying routing mechanism is yet available. The same holds true for Practical Internet Coordinates (PIC) described by Costa *et al.* (2004). It is quite similar to Vivaldi but employs a different energy minimisation algorithm. PIC also incorporates security measures to cope with malicious nodes that disrupt the network by sending false information.

Yang *et al.* (2010) proposes *SIEMAP* which assigns coordinates based on a fixed set of reference coordinates. The difference between the distance in the virtual space and the shortest path hop counts is minimised in an iterative fashion. A downside is the dependence on an all-to-all shortest path calculation. This makes it unsuited for larger networks as it has a computational complexity $O(|V|^2)$ for a graph $G = (V, E)$.

A subsection of geometric routing algorithms based on iterative embeddings is devoted to landmark-based routing. These systems typically use two-dimensional coordinate planes in which some nodes are identified as so-called landmarks which act as reference points for other nodes. Examples of such systems are the routing algorithm developed by Kleinberg *et al.* (2009), Small State and Small Stretch Routing Protocol (S4) by Mao *et al.* (2007), Gradient Landmark-Based Distributed Routing for Sensor Networks (GLIDER) by Fang *et al.* (2005) and Beacon Vector Routing (BVR) by Fonseca *et al.* (2005).

Boguñá *et al.* (2010) propose a method to embed a graph into the two-dimensional hyperbolic plane $\mathbb{H}^2$. This is done by maximising the resemblance between the embedding produced by their

model and shortest path distance between nodes pairs. This basically comes down to maximising a function that is influenced positively by two adjacent vertices having a low hyperbolic distance and two non-adjacent vertices having a high hyperbolic distance. This is combined with a bias towards embedding high-degree nodes closer to the centre of the hyperbolic Poincaré disk. The main downside is that their algorithm requires centralised computation which —as reported by the authors— can take days to finish. Though a high success rate is reported, the embedding itself is not greedy. This article however points out that using a hyperbolic embedding is very fitting for the inter-AS network and other scale-free networks in general.

2.2.3 Structured graph embeddings

Structured embeddings are a second type of graph embeddings. These differ from iterative embeddings in the sense coordinate assignment is not guided by the optimisation of an objective function. Rather the embedding is constructed by making use of an underlying fixed structure. In most cases this is a *spanning tree* of the network (see Definition 2.10) which is constructed before the embedding procedure begins.

Definition 2.10. *A spanning tree $T = (V, E')$ of a connected graph $G = (V, E)$ is a graph made up of all vertices $v \in V$ and a set of edges $E' \subseteq E$ which contains no cycles.*

Structured embedding construction procedures first generate the underlying structure and then assign coordinates to nodes based on their relative position within this structure. Compared to iterative coordinates they are far more restricted as only certain values of the coordinate space are allowed. This type of embedding is generally more abstract than the first kind.

An example of a routing system using a structured embedding is Virtual Polar Coordinate Routing (VPCR)[17]. In this system a spanning tree is built in a distributed fashion. Hereafter polar coordinate ranges are assigned to the network nodes which act as virtual coordinates. Each point has a distinct polar range $\phi_n = [\theta_a, \theta_b]_n \subset \phi_{n-1} = [\theta_a, \theta_b]_{n-1}$ where the n denotes the depth of the node inside the tree. The ranges are assigned in such a way that nodes at the same depth of the underlying tree never overlap (except a single point), e.g. $\{[0, \pi]_n, [\pi, 2\pi]_n\}$. The larger the depth of the node, the smaller its range is. These ranges will act as two-dimensional coordinate sets (x_0, x_1) with $x_0 = \theta_a$ and $x_1 = \theta_b$. The authors do not formulate a clear-cut distance function δ though. Because of this, routing forwarding is not entirely based on the distance between nodes. By using polar coordinates, ever node can be located in the tree because a child's coordinate set is a subset of its parent's. This yields information about whether a node resides inside a specific sub-tree which can be used for forwarding decision making. A promising characteristic of VPCR is that it allows *smart routing*. In smart routing a node can choose to send a packet closer to the destination by forwarding to a node with a polar range closer to the destination range. This however requires the ranges to be ordered according to the underlying topology, which is only possible in UDGs. Much like other basic tree routing algorithms, bottlenecks form at higher levels in the hierarchical tree. Smart routing diminishes, but not eliminates, this effect. This routing mechanism will be discussed in greater detail in Section 4.2.1.

[17] Newsome & Song (2003)

Goodrich & Strash (2009) show that using a so-called Christmas cactus graph for three-connected graphs allows successful greedy forwarding. This kind of graph can be interesting as it allows more than one path up to the node that acts as the root of the graph. This paper is however strictly theoretical and lacks any form of practical distributed algorithm for computing and assigning the coordinates.

A hyperbolic embedding of a graph was proposed by Kleinberg (2007). It is proven that this embedding is greedy given that it is constructed with a specific spanning tree. The presented routing mechanism employs hyperbolic distances, based on the geodesic that connects two points in the Poincaré disk model. The system fixes coordinates upon the spanning tree in a systematic manner. A prerequisite for having a greedy embedding is that the graph has to be three-connected, which is a requirement for being able to build a minimal binary tree used to guide the coordinate assignment. The author also proposes a procedure for distributed coordinate assignment. Section 4.2.2 will further investigate this routing mechanism.

Later Eppstein & Goodrich (2008) have proven that the coordinates assigned by Kleinberg (2007) requires $\Omega(|V| \log |V|)$ bits for a graph $G = (V, E)$. Therefore the space required is of the same order as the storage requirements of BGP routing tables. They postulate that a graph embedding should use succinct coordinates: coordinates that have a number of bits that is poly-logarithmic in $|V|$. Such a succinct coordinate assignment method is proposed which makes use of a transformation of the spanning tree to an *autocratic weight-balanced tree*. Using this transformed tree, they are able to form a binary tree of depth $O(\log |V|)$. Hereafter an auxiliary space, the *dyadic tree metric space*, is used. In this space, with its corresponding distance metric, a greedy embedding is obtained. Finally this dyadic tree is embedded in a hyperbolic space, conserving its succinctness and greedy properties.

Korman *et al.* (2002) show that labels can be used to identify nodes in a tree. The authors describe protocols to assign labels to different types of trees: static, semi-dynamic and dynamic trees. A static tree is one where the tree structure remains intact. Semi-dynamic means that vertices may be added as leafs of the tree. The dynamic protocol allows leaf nodes to be removed on top of being added. The authors thus incorporate network dynamics in their assignment scheme. The message and storage complexity is analysed in each case. The tree construction itself is not specified, it is assumed to already be present. Neither does the protocol take into account the deletion of non-leaf nodes or the deletion of edges. This work does not address the use of labelled trees for geometric routing. It does however establishes a theoretical framework for the labelling process itself. The use of labels for geometric routing purposes was first seen in Routing with Position Trees (RTP) by Chávez *et al.* (2007). Herein every node is assigned a label as described by Korman *et al.* (2002) while the distance between two nodes is the shortest path length along the underlying spanning tree.

2.3 AS-level Internet

The performance of the various routing algorithm may depend on the network type and the traffic model used. As such it is of importance to investigate how the Internet backbone can be modelled

realistically for simulation purposes.

2.3.1 Topology

Ad-hoc wireless networks and WSNs are typically modelled as UDGs.[18] Hence this serves as a basis for most of the work regarding geometric routing. UDGs can be naturally embedded in a two-dimensional Euclidean plane $\mathbb{E}^2$. However, this is not necessarily the case for the AS-level Internet. Faloutsos *et al.* (1999) collected data that accurately describes its topology by querying the BGP routing tables. What was observed can be formulated into three so-called *power-laws*.

The first power-law states that the degree $d_G(u)$ of a vertex u is proportional to its rank (assume that the vertices of a graph have been sorted in decreasing order of degree) r_u to the power of a constant R:

$$d_G(u) \propto r_u^R \tag{2.4}$$

This power-law captures the equilibrium point of the trade-off between the costs and gains of adding new edges to already existing nodes. The second power-law states that the frequency f_d of a node degree d is proportional to this degree raised to the power of a constant O:

$$f_d \propto d^O \tag{2.5}$$

This degree exponent O expresses the existence of a high number of low-degree nodes and a low number of high-degree nodes. Various Internet datasets have the same degree exponent O which is approximately -2.2. The third power-law states that the eigenvalues of the network[19] λ_i are proportional to its number of vertices $|V|$ raised to the power of a constant ϵ:

$$\lambda \propto |V|^\epsilon \tag{2.6}$$

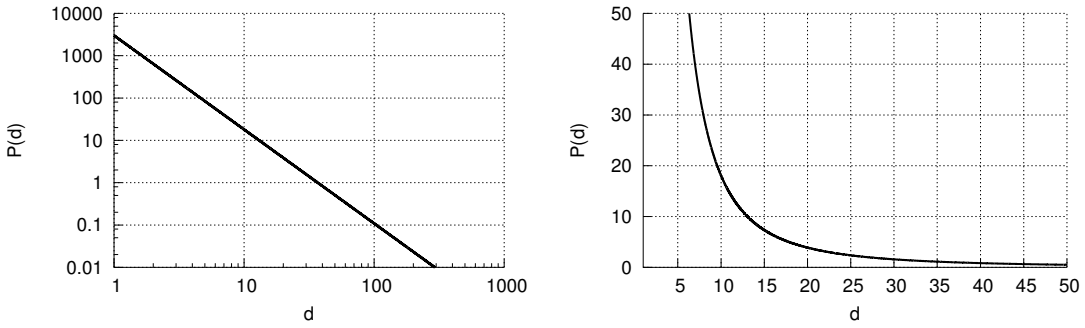
A graph's eigenvalues are useful as they are related to other topological properties such as the number of possible spanning trees or the diameter.[20] Therefore it may be used to classify graphs.

These observations lead to the fact that the Internet AS-network can be modelled as a scale-free network. Such a network follows a statistical degree distribution (defined in Definition 2.11) $P(d) \propto d^{-\gamma}$ with d the vertex degree and $\gamma \approx 2.22$ as $d \to +\infty$. This is shown in Figure 2.7.

Definition 2.11. *Let $n(d)$ be the number of nodes of degree d. The node degree distribution is $P(d) = \dfrac{n(d)}{|V|}$, it is the probability that a randomly selected vertex of a graph $G = (V, E)$ is of degree d.*

Though a network's degree distribution is an important property as it contains information of its global characteristics and can be used to classify different networks[21], it alone does not capture the fundamental hierarchical structure of the Internet backbone. Briefly said, there are three

[18] Bouabene *et al.* (2012)
[19] These are the eigenvalues of the graph's adjacency matrix.
[20] Cvetković *et al.* (1982)
[21] Zhou & Mondragón (2004b)



(a) Log-log scale

(b) Linear-linear scale

Figure 2.7: Scale-free network degree distribution $P(d) = \alpha d^{-\gamma}$; $\alpha = 3000$ and $\gamma = 2.22$

main properties in the Internet backbone: (i) its degree distribution $P(d)$; (ii) the maximum node degree; (iii) its rich-club connectivity (RCC) $\phi(r)$.[22] This RCC can be defined by the following definition.[23]

Definition 2.12. *Nodes of graph $G = (V, E)$ are sorted by decreasing number of links that each node contains. There are instances where groups of nodes contain identical number of links. Where this occurs, they are arbitrarily assigned a position within that group. The node rank r denotes the position of a node in this ordered list. Furthermore r is normalised by $|V|$. The rich-club consists of those nodes with a rank less than r_{max}, which can be arbitrarily defined. The rich-club connectivity (RCC) is defined as the ratio of the total number of links to the maximum possible number of links between members of the rich-club. The maximum possible number of links between n nodes is $n(n-1)/2$.*

This means that the RCC measures the similarity between the rich-club and a clique consisting of the same nodes.[24] Figure 2.8 illustrates this phenomenon.[25] Having a high RCC means that the highest-degree nodes have many links between them, they form a dense core of the network.

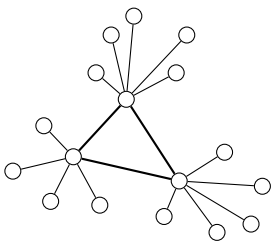


Figure 2.8: Rich-club phenomenon: high-degree nodes are likely to be interconnected.

For obtaining a realistic AS-level connectivity representation, there are two notable datasets: the BGP AS graphs, based on the Internet inter-domain BGP routing tables[26] and the Traceroute AS

[22] Subramanian *et al.* (2002)
[23] Zhou & Mondragón (2004a)
[24] Cliques are complete sub-graphs of the network.
[25] Mahadevan *et al.* (2006)
[26] Hyun *et al.* (2003)

graph, constructed by converting IP traceroute paths. Not all scale-free networks express all properties found in real AS graphs. Most notable is the previously mentioned *rich-club phenomenon* in which a small number of vertices, called rich nodes, have a high degree. Rich nodes are preferentially attached to other rich nodes, forming a tight core. Many other nodes are connected directly to this core which enables the core to act as a traffic hub for the network. A widely used model for generating scale-free networks is the Barabási-Albert (BA) model[27]. Herein a network is grown with a power-law degree, employing a *preferential-growth* mechanism. A small random seed network is expanded by iteratively attaching nodes to nodes already present in the graph with a probability of

$$P(u) = \frac{d_G(u)}{\sum_{v \in V} d_G(v)} \tag{2.7}$$

for the attachment of a new node to node u. A shortcoming indicated by the authors is the lack of a high RCC. Therefore more accurate —though more complex— models have been developed such as the Generalized Linear Preference (GLP) model[28] which also tries to capture the addition of new connections between already existing nodes of the network. Furthermore the GLP model uses a more sophisticated probability distribution for node attachment. Later the Interactive Growth (IG) model[29] has been developed in which the selection of host nodes, nodes to which new nodes are attached, is once again based on Eq. (2.7).

Hereafter Zhou & Mondragón (2004b) made some slight adjustments to the IG model to better approximate the AS-level topology, based on observations of Chang *et al.* (2004). The authors formulated a relationship between the degree of a node and its chance of acquiring a new link when adding a new node. This was used as a base for a new graph generation model based on *nonlinear preferential attachment*:

$$P(u) = \frac{d_G(u)^{1+\psi \log d_G(u)}}{\sum_{v \in V} d_G(v)^{1+\log d_G(v)}} \tag{2.8}$$

with $\psi = 0.048$, obtained by numerical simulations. This equation approximates the original BA equation, Eq. (2.7), for low-degree vertices. The attachment preference increases non-linearly as the degree goes up, according to the most recent observations. This equation is the foundation of the Positive-Feedback Preference (PFP) model, an extension of the IG model. The algorithm generates links between both newly generated nodes and nodes already present in the network. The authors report parameter values for PFP that accurately model the AS-level topology.

Siganos *et al.* (2006) made other striking observations regarding the Internet backbone topology. Approximately 35% to 45% of the nodes are one-degree nodes and these nodes are both connected to high- and low-degree vertices. Trying to capture this property, the authors observed a linear relationship between the number of one-degree nodes of a neighbouring node and the complementary cumulative distribution function (CCDF), denoted as O_r. This relationship forms the basis for a fourth power-law as subsequently defined.

[27] Barabási & Albert (1999)
[28] Bu & Towsley (2002)
[29] Zhou & Mondragón (2003)

Definition 2.13. *Given a graph $G = (V, E)$, the CCDF, denoted as O_r, of the one-degree neighbours of a vertex $v \in V$, is proportional to the number of one-degree neighbours, denoted as r, raised to the power of a constant θ:*

$$O_r \propto r^{\theta} \tag{2.9}$$

Putting this observation to work, a new model was proposed, named the *Jellyfish model*. This model captures the scale-free nature of the AS-level topology and its hierarchical structure, while still adhering to the high frequency of one-degree nodes. Furthermore, the authors evaluated their model over a large period of time and observed that its structure remains valid over multiple years. It thus remains accurate although the Internet increases in size.

An important link between geometric routing and scale-free networks is given by Papadopoulos *et al.* (2010). They postulate that there exist *hidden metric spaces* in scale-free networks. These hidden metric spaces result in two nodes being likely connected when their distance in this space is low. The authors conclude that a hidden metric space for scale-free networks is the hyperbolic plane. This leads to efficient greedy routing when using an embedding into $\mathbb{H}^2$.

2.3.2 Traffic models

In this section the traffic flows of the Internet backbone is investigated. Obtaining a realistic model is difficult as traffic data is rather scarce.[30] A way to represent network traffic is the use of a matrix $(e_{i,j}) \in \mathbb{R}^{|V|\times|V|}$ with $i, j \in \{1, 2, \ldots, |V|\}$ and all elements $e_{i,i} = 0$. Its rows and columns represent the source and destination nodes, while its elements represent traffic between them. Mikians *et al.* (2012) analysed AS-level data and formed an Interdomain Traffic Matrix (ITM). For this they utilised the GÉANT network, a European academic network.[31] Though it is not representative for the entire Internet backbone, it may provide useful insights. The authors' research focuses on the spatial aspects of the traffic generated rather than its temporal evolution.

Some interesting points can be deduced. First of all, the matrix is sparse. This indicates that over a period of time a large part of the nodes are never routing destinations. Traffic flows mostly to popular networks. In a week's time about 45% of all the ITM elements are zero.[32] A remarkable result was that 15% of the destinations were responsible for over 95% of the total traffic. Secondly, the traffic has a *heavy-tailed* distribution.[33] More specifically, the distribution resides somewhere between a Pareto and a log-normal distribution (see the following definitions).

Definition 2.14. *A Pareto distribution has a distribution probability of the form*

$$P(x) = \left(\frac{x}{\sigma}\right)^{-\alpha}, \; x > \sigma \tag{2.10}$$

with the scale parameter σ and α in $\mathbb{R}^+$.

[30] Mikians *et al.* (2012)

[31] GÉANT is a European wide network that resides in the Internet backbone. It spans 34 countries and has an overall throughput of approximately 50 Gb/s. Its customers are however mainly research institutes, leading to an academic bias. Half of its traffic is directed towards commercial networks.

[32] Mikians *et al.* (2012) observed this by collecting data from the Universitat Politècnica de Catalunya, BarcelonaTech access link, as all traffic originating from UPC is required to pass this link no traffic goes unnoticed. The claim is consistent with results reported by Gadkari *et al.* (2011).

[33] This is backed-up by the findings of Downey (2001) which state that Internet HTTP and FTP burst-sizes and lengths are heavy-tailed. Cha *et al.* (2007) report similar results for the distribution of YouTube video traffic.

Definition 2.15. *A log-normal distribution is a continuous probability distribution that has a normal distribution for its logarithm. Its density function can be formulated as*

$$P(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}},\ x > 0 \tag{2.11}$$

with $\mu \in \mathbb{R}$ *and* $\sigma^2 > 0$.

In both distributions parameter values were sought to give a realistic shape. The authors report values for α between 0.37 and 1.20 and a σ^2-value varying between 0.13 and 0.38.

2.4 Load balancing

In this section literature regarding the application of traffic load balancing strategies to geometric routing will be investigated. Generally, load balancing strategies can be categorized into two types: *passive* and *active* load balancing. Passive load balancing strategies attempt to spread traffic out over multiple nodes or links without the use of any live load information. On the other hand, active load balancing strategies do make use of traffic load snapshots. This load information is then used to actively steer traffic in such a way that hotspots are avoided. Because of this active traffic steering, it may dynamically respond to changes in the traffic matrix. Load balancing strategies may also be classified based on whether they try to balance load for network nodes or network links. These different emphasises lead to different results, as will be shown later on in the thesis.

2.4.1 Mechanisms

Because many geometric routing mechanisms target WSNs, many load balancing techniques try to lower the number of overloaded nodes in order to increase battery lifetime. Most node load balancing selection procedures therefore try to avoid nodes with low battery levels. Also the wireless nature of WSNs does not allow link load balancing. As a result, many studies focus on node load balancing rather than link load balancing.

One of the earlier geometric routing systems incorporating load balancing is Geographic and Energy Aware Routing (GEAR)[34]. This mechanism uses a next-hop selection heuristic based on neighbouring node battery levels to guide traffic towards the nodes with highest battery levels. Significantly longer lifetimes have been reported over GPSR. As node energy depletion correlates with the traffic it has to process, this also entails node load balancing. It has to be noted that GEAR is exclusively designed for UDGs. A second drawback is the requirement for every node to store a *learned cost* to every target region. This learned cost is a combination of the energy levels of the nodes traversed when routing towards this region, and the path length. As every node requires $O(|V|)$ entries in its table, scalability problems may arise as the network size increases. Another load balancing mechanism based on node energy consumption is proposed by Zeng *et al.* (2009). Herein the selection criterion for choosing the next hop is based on a cost function. This function is a combination of the distance to the destination and the node energy level. Neighbours are thus not only selected according to their position in the plane, but also according to their battery level.

[34] Yu *et al.* (2001)

Carlsson & Eager (2007) propose a non-Euclidean geometric load balanced routing system called Load Balanced Local Shortest Path (LBLSP). The mechanism combines greedy routing with two additional metrics called Local Shortest Path (LSP) and Weighted Distance Gain (WDG). The former has the property of being loop-free with a low chance of reaching a void while the latter provides load balancing, however it is not guaranteed to be loop-free. LSP makes use of the concept of line-of-sight in radio networks, making it unusable in wired networks. The same holds true for WDG, where an estimated distance to an obstacle —based on the line-of-sight— is used. Virtual obstacles are introduced at the position of traffic hotspot to force traffic deflection. The authors report that the natural detection of the exact size and shape of these virtual objects to attain load balancing is still an open problem.

Li *et al.* (2010) have proposed a novel geometric routing mechanism called Circular Sail Routing (CSL). Passive load balancing is achieved by mapping a two-dimensional coordinate space onto a sphere by means of a stereographic projection. Forwarding decisions are made by employing a geodesic distance on this sphere. As the authors have shown, in a two-dimensional network grid, traffic is concentrated in the centre when using an embedding into $\mathbb{E}^2$.[35] When routing on a sphere this is not the case as no centre exists. As shown by the results, the centre traffic hotspot of the network is indeed avoided by CSL. The downside however is that it is solely designed and tested for grid networks with a size of approximately 400 nodes.

A similar routing mechanism is *Curveball Routing*[36]. The authors first developed a theoretical network load model for UDGs. This is consolidated into an equation that describes the average node load in function of its distance to the centre of the network. They term the traffic hotspot formed under a uniform traffic matrix between random source-destination pairs at the centre of the network as the *crowded centre effect*. The authors then form a bridge between optimal routing paths in terms of load balancing, and curved paths that light follows when traversing a medium with a varying refraction index. Theoretical computation of the minimum load paths seems infeasible as excessive computation overhead is required, or large pre-calculated tables are necessary. This leads to Curveball Routing, a heuristic that approximates this theoretical path construction. Curveball Routing projects the plane $\mathbb{E}^2$ onto a sphere, much like CSL, to diminish the crowded centre effect. The newly obtained coordinates on the sphere are used for greedy routing. The coordinates in $\mathbb{E}^2$ are still used as a fallback method in case voids are encountered. An undesired effect of mapping coordinates onto a sphere is however that the top of the sphere is more crowded due to the projection properties. Also the sphere lacks completely coverage: there is an upper cap with no points mapped to it. The load balancing behaviour was measured by monitoring the maximum load handled by a node. The authors report slightly positive results in terms of load balancing both on their synthetic and real test bed.

Once more driven by the balancing of energy consumption in WSNs, Zhang *et al.* (2005) proposed a *Trajectory-Based Forwarding* technique that forms the basis for their *Curve-Based Greedy Routing Algorithm*. By using B-splines to construct multiple curved paths from source to destination, traffic can be balanced amongst them. The source node selects a fitting B-spline curve and encodes this

[35] This has been formally proven by Popa *et al.* (2007).
[36] Popa *et al.* (2007)

in the packet headers. This information is hereafter decoded at every node to make forwarding decisions. Each node selects its neighbour with the lowest distance to the curve as a next node (which is also closer to the destination). Nodes along the trajectory will send feedback about the node congestion to the previous ones. These can then select a different next hop if needed to avoid node traffic overload. Here the distance to the B-spline is used as a cost function. If routing fails, a feedback message is sent to the source in order to generate a new spline. The algorithm achieves active load balancing but is only tested for 200 nodes, and is based on UDGs. Also the node storage requirements for the splines are not investigated.

In the family of systems based on structured embeddings, Tang *et al.* (2010) have proposed the use of more than one spanning tree simultaneously, to generate multiple embeddings.[37] Root nodes for the various trees are chosen randomly. Every tree in use leads to a different set of coordinates for each node. In their work the packet overhead, the node load and the average stretch are all evaluated for a varying number of trees. A limitation however is that their mechanism is solely focussed on WSNs, and is thus tested only on UDGs. The tests are also limited to around 1000 nodes, so the algorithm's scalability is not extensively examined. As they target wireless networks, the investigation of link load distribution is missing, which plays an important role in the Internet backbone. Their greedy routing mechanism in which a neighbouring node is selected based on its minimal distance over all of the embeddings, results in a passive form of load balancing.

Newsome & Song (2003) propose a passive load balancing scheme for their Virtual Polar Coordinate Space (VPCS)-based routing system. Their tree-based algorithm tries to avoid the natural bottleneck around the root of the underlying tree. Especially when shortcuts within the tree are unavailable, traffic has to be routed over the spanning tree itself. By employing smart routing the algorithm makes use of links that would be left out when using a default greedy routing scheme. Their final scheme is named VPCR and is described in Section 2.2.3 as well as Section 4.2.1. A downside is that the polar ranges have to be ordered in a plane $\mathbb{E}^2$. This comes down to forming rings of nodes at each level of the tree. If these rings are ordered correctly, routing along them will bring a packet closer to its destination. It is hard to see how this can be implemented effectively outside the scope of UDGs. By taking paths that are less likely to traverse the root node, the researchers report better passive load balancing behaviour.

2.4.2 Metrics

To be able to objectively measure the load balancing properties of a routing mechanism, load balancing metrics have to be defined. Metrics found in literature were originally designed to be used for measuring node load balancing, but they can be naturally extended to link load balancing. A metric found in literature is a β-ratio[38]. This ratio calculates the traffic distribution over each node or link as

[37] This system makes use of the theoretical framework established by Korman *et al.* (2002) as its foundation.
[38] Velayos *et al.* (2004)

$$\beta = \frac{\left(\sum_{x \in X} f_x\right)^2}{|X| \sum_{x \in X} f_x^2} \tag{2.12}$$

with X the set of nodes V or links E of the network $G = (V, E)$ and f_x the number of paths going through node or link x. The β-ratio goes from $\frac{1}{|X|} \approx 0$ (unbalanced) to 1 (balanced). An advantage of the β-ratio is that has a standard range $[0, 1]$ which is easily understandable. Another load balancing metric is proposed by Li & Xiao (2010). This metric takes into account the variance of the number of packets passing a certain node or link x:

$$\phi = \frac{\sum_{x \in X} \left(L_x - \hat{L}\right)^2}{|X|} \tag{2.13}$$

with L_x the number of packets passing through node or link x and

$$\hat{L} = \frac{\sum_{x \in X} L_x}{|X|} \tag{2.14}$$

the average number of packets passing through the nodes or links of the network and X the set of nodes V or links E. These metric do not take into account the potentially increased stretch caused by the load balancing effect. This effect cannot be neglected as it is very plausible that high load balancing values will induce more total traffic.

A trade-off between network stretch and load balancing is studied by Gao & Zhang (2004) and Gao & Zhang (2009). These studies focus on UDGs and take into account the node density when embedding the network in the plane $\mathbb{E}^2$. A theoretical framework is constructed, however only unrealistic worst-case traffic scenarios and node layouts are evaluated.[39] Articles such as Li *et al.* (2010) measure the effects of load balancing by observing the average and maximum load (over all network nodes), along with the standard deviation. Carlsson & Eager (2007) particularly measure the average maximum number of packets processed by a node, which is essentially the average maximum load. Popa *et al.* (2007) measure the average and maximum load. Zhang *et al.* (2005) measure the load by the observing the average node energy usage. If the energy usage can be linked to a node's load, this should correspond to the average load. Bruck *et al.* (2005) measure the load balancing behaviour of their mechanism by means of the *normalized standard deviation*, which is the standard deviation divided by the average.

In this thesis the β-ratio will be used for measuring load balancing behaviour as it is easy to interpret, along with a normalized standard deviation which will be called the λ-ratio. Both ratios will be used to measure link and node load balancing behaviour of routing schemes.

[39] This is admitted by the researchers.

2.5 Routing fault-tolerance

This section surveys literature regarding the fault-tolerance of geometric routing mechanisms. With fault-tolerance, a routing system's ability to cope with network dynamics, such as node or link failures, is meant. This however has not yet been extensively studied.[40] Cvetkovski & Crovella (2009) are pioneers in exploring fault-tolerance applied to geometric routing. They propose a generalization of geometric tree routing called Gravity-Pressure (GP) routing. Herein two modes are operational: (i) *gravity* routing mode and (ii) *gravity-pressure* routing mode. The routing algorithm operates by default in gravity mode, which is simply greedy routing. Whenever packets get stuck in a void, gravity-pressure mode is activated for that packet. Now routers are no longer restricted to following a distance-decreasing path for this packet. From the moment gravity mode is activated, each packet will record the nodes it passes in its header, combined with the number of times it passes them (the visitation number). When selecting a next hop this visitation number will be taken into account by only considering those nodes with a minimal visitation number. From this set, the node with the lowest distance to the destination will be selected. The authors have proven that GP routing guarantees 100% routing delivery as long as the network is not disconnected. It is capable of dealing with both node and link failures.

A solution to single and multiple link failures has been proposed by Sahhaf *et al.* (2013). Single link failures are resolved by engaging in a breadth-first search to find an intermediate node on a path towards the destination. If such a node is found, it is queried in order to obtain its coordinates. The node at which packets got stuck will now route packets towards this intermediate node. A second algorithm presented deals with multiple simultaneous link failures. Herein a path protection algorithm is developed to account for link failures in hyperbolic routing as described by Kleinberg (2007). Their mechanism avoids voids with limited overhead and has a comparable stretch to other link protection methods. The link recovery time can be lowered by pre-calculating the alternative paths for fast packet re-routing.

2.6 Conclusion

Little work in scientific literature investigates the application of geometric routing to wired networks such as the Internet backbone. The main difference with wireless networks is that these wired networks cannot be modelled as UDGs. It has been shown that Internet AS-level topologies are scale-free by nature. Also the application of algorithms targeting wireless networks to wired networks has rarely been examined.
The majority of the algorithms investigated are not tested on sufficiently large networks. Therefore no guarantees can be made about their scalability. Also the use of multiple embeddings for routing purposes leaves many open questions. Furthermore not much work deals with fault-tolerance regarding geometric routing.
A general trend that can be witnessed regarding load balancing in geometric routing is that there is little to no work on active load balancing behaviour in scale-free networks. Most work assumes UDGs as an underlying topology. Link load balancing behaviour research could not be found which

[40] Sahhaf *et al.* (2013)

may turn out to be fundamentally different from node load balancing. Once again the different load balancing algorithms have not been tested on large networks. This means that load balancing in scale-free topologies is currently an open research problem, especially in large networks with a focus on scalability.

It can be concluded that load balancing applied to scale-free networks, in particular the Internet AS-level network based on realistic traffic models, with a focus on scalability, link load balancing and fault-tolerance as an alternative to BGP is still an open research problem.

Chapter 3

Routing simulator

An important part of the design, implementation and evaluation of different routing mechanisms is the use a solid simulation. Therefore a routing simulator was required in which it was possible to implement algorithms quickly while avoiding the complexity of real distributed behaviour. To achieve this, a Java software framework was developed in which novel routing mechanisms could easily be implemented, tested and visualised in a multi-threaded environment. This framework was built on top of the open-source graph visualisation tool Gephi[1].

3.1 General

First, a routing simulator was built using the open-source graph manipulation framework Java Universal Network/Graph Framework (JUNG). Afterwards it became clear that this graph framework had only a small code-base regarding extra functions such as generating dynamic graph layouts or graph data analysis. JUNG was also incapable of offloading graphics rendering to the graphics processing unit (GPU). This is an important requirement as the computational power of the central processing unit (CPU) is needed for executing the routing simulation environment itself. The graph visualisation project Gephi had this ability. The downside was that Gephi, unlike JUNG, is not a regular graph manipulation library. Gephi is a piece of standalone software capable of rendering very large graphs by using OpenGL on the GPU. Therefore the Gephi program itself, of which the source code was publicly available, had to be modified to be able to perform the same routing simulations as the previous JUNG-based simulator.

First of all Gephi had to be partially reverse-engineered to be able to understand its different classes and functions. Afterwards unwanted behaviour was changed and additional functionality was added. For example the changing opacity of the links and nodes upon hovering-over by the cursor was removed and the main graphical user interface (GUI) was thinned. The most important modification was the addition/modification of the software interface by which other modules are able to use Gephi to manipulate graphs. This allowed the migration of the code of the previous simulator without any problems. The final result is the existence of two disjoint modules: Gephi with its 2D graph rendering engine and manipulation techniques, and the routing simulator itself.

[1] Bastian *et al.* (2009)

3.2 Architecture

A simplified view on the software architecture of the routing simulator (including the Gephi module) is shown in Figure 3.1. As can be witnessed, the total project can be subdivided into multiple packages, consisting of classes which belong together either hierarchically or logically. First there is the `gephi` package (depicted in blue) which consist of a graph structure `Graph`, capable of linking together `Node` objects by means of `Edge` objects, and the Gephi visualisation engine. This rendering engine will take care of anything related to the visual aspects of the graphs. For example, when the colour field of an `Edge` object is changed, the rendering engine will be notified and repaints it. This `gephi` package actually forms a standalone project which is interfaced with the actual routing simulator, consisting of the other packages.

The `gephi` package interfaces on the one hand with the package `visuals` and on the other hand with the package `bridge`. The package `visuals` consists of a `RoutingPaneGUI` which defines a pane within the main Gephi GUI. From this pane, the routing simulator can be manipulated, e.g. changing parameters, stopping the simulation, generating coordinates etc. Furthermore there is the `ChartGenerator` which is responsible for drawing live plots based on parameters in the `RoutingPaneGUI`. For example, it is capable of monitoring the current stretch or success ratio. The second package interfacing with the `gephi` project is the `bridge`. As the name indicates, this package consists of classes that form a bridge between the Gephi program and the routing simulator. This is a very important module as it separates Gephi completely from the routing simulator itself, thus enforcing code modularity.

The different classes will be explained by describing how a routing simulator instance can be created. First, via the Gephi GUI, a graph structure is loaded into the program from an arbitrary graph file (e.g. a GraphML file). This leads to the creation of a `Graph` object with a set of edges (`Edge`) and vertices (`Node`). Next, the `RouterEtcher` is called. This class is responsible for generating the same graph structure as before, consisting of objects of the type `Vertex` and of the type `Link` in the routing simulator project (the non-blue packages). Afterwards, there exist two graph structures: a graph in the Gephi project, which is used to store visual information for graphics rendering purposes, and the graph structure in the routing simulator project, which is used by the other classes to store general and visual information. This distinction exists to completely separate the routing simulator from Gephi. The routing simulator graph edges and vertices are stored in two hash maps mapping vertex and edge identifiers to objects.

Hereafter, the `RouterEtcher` will attach one `RouterImpl` object to each vertex. The `RouterImpl` class is a specific subclass of the more general `Router` class. For example, one may create routers of different types, e.g. `GeoRouterA` and `GeoRouterB`, which both inherit from `Router`. To keep architecture modular, the `RouterEtcher` will not directly create `RouterImpl` objects, but rather it will use a `RouterFactory` with a specific `RouterFactoryImpl`, based on the factory software pattern. Finally, the `RouterEtcher` will create neighbour tables in each `Router` based on router identifiers, by inspecting the vertex and link hash maps. Now the different routers know their neighbours.

Next a `CoordinateAssigner` is created for a specific routing instance (e.g. `GeoRoutingA`) which will

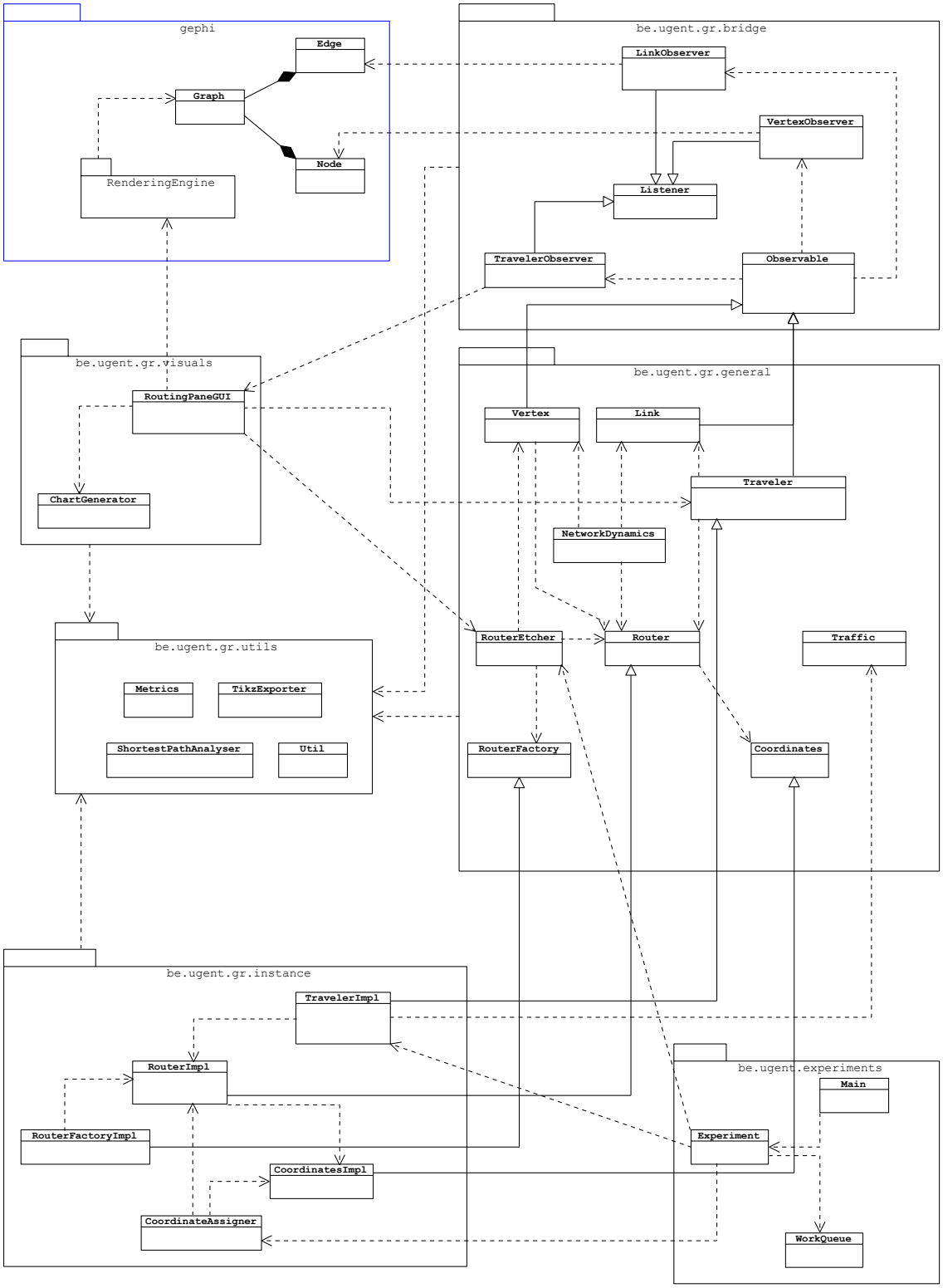


Figure 3.1: Architecture of the routing simulator

assign `CoordinatesImpl` to all routers based on a graph embedding scheme. `CoordinatesImpl` is a subclass of `Coordinates` which contains more specific information regarding a coordinate structure: e.g. `GeoCoordinatesA`, which requires the storage of its own length.

When all coordinates are generated, the `TravelerImpl` is called. This `TravelerImpl` contains all routing logic specific to a certain routing algorithm (e.g. `GeoRoutingA`). The more general logic, such as stretch or success ratio calculation is done by its parent class `Traveler`. This `TravelerImpl` is capable of generating routing paths based on the criteria specified in its logic. As such it simulates routing behaviour. This is done by altering weights of edges and vertices as well as calculating the stretch and success ratio. These weights are used to store link and node load information. Whenever traffic is generated over a link or node, its weight is increased (see Section 3.3 for more information). The specific style of augmenting this weight is done by the `Traffic` class, which is capable of generating traffic of different types, such as uniform traffic or Pareto distributed traffic. Furthermore the `NetworkDynamics` class can be used to generate link and node failures following a certain probability distribution with a specific frequency.

Utility functions are all combined into a single `utils` package. Examples of such functions are exporting the graph to Latex code with `TikzExporter`, analysing shortest path behaviour with `ShorestPathAnalyser` or calculating load balancing metrics with `Metrics`. All classes in this package are public such that other classes may freely use them. On top of this, there is a `Config` class, which is not depicted in Figure 3.1, containing all necessary configuration information regarding the routing simulator (e.g. parameter values). Every class has access to this `Config` class.

Now the package `bridge` is explained. Classes outside of the Gephi project are capable of manipulating `Vertex` or `Link` objects. For example, changing a vertex' position on screen, increasing its size, changing its colour etc. To be able to couple the manipulation of these objects to changes in graphical rendering of their complementary `Node` and `Edge` objects, an observer pattern was implemented. Now each object extending the `Observable` class will notify `Observer` classes. These `Observer` classes all implement a `Listener` interface. Whenever something is changed in one of the `Observable` objects (e.g. a field value has been altered), these observers will notify the correct classes in the package `visuals` and `gephi`. As such, these classes are notified and change their state. For example a certain `RouterImpl` is marked by a graph embedding algorithm. This will cause the router's corresponding `Vertex` object to change it colour. This object will notify the `VertexObserver` which will notify the corresponding `Node` object inside Gephi. This `Node` object will notify the `RenderingEngine` which will re-draw the node on screen (more information on the bridge can be found in Section 3.5).

In Figure 3.2 the routing simulator architecture is slightly altered. Here the `Traveler` and `CoordinateAssigner` are removed and four classes are added (which are depicted in red). Note that this is not an actual altered version of the routing simulator, these classes (e.g. `Traveler` and `Simulator`) coexist in one project and the distinction was simply made for clarity purposes. This picture highlights a different way of simulating routing behaviour. First a `Simulator` instance will be set up (instead of a `Traveler`). This `Simulator` will indefinitely (until stopped) call a `Router` checking function. This `Router` function represents the receipt of a possibly incoming

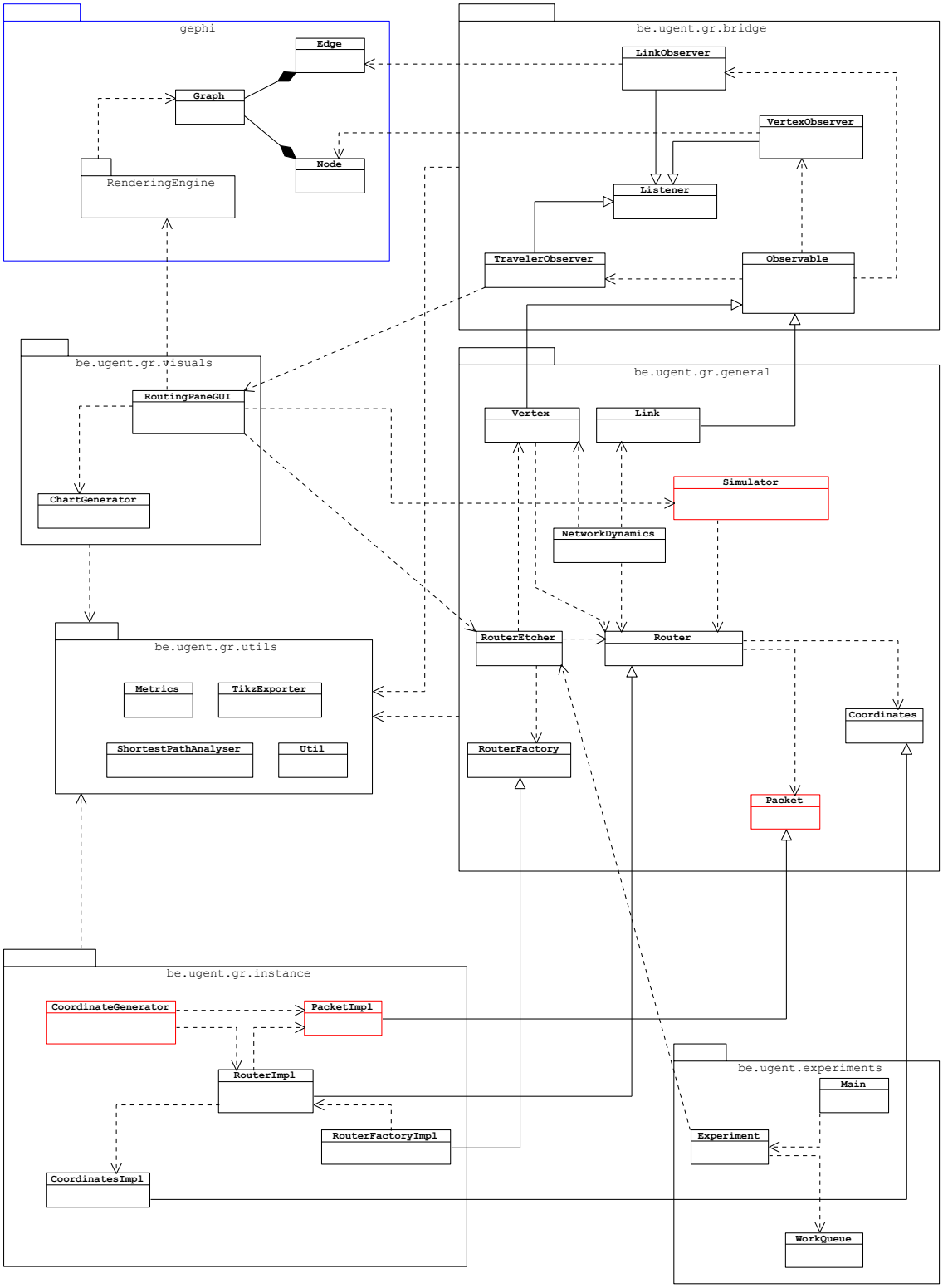


Figure 3.2: Architecture of the routing simulator: altered

packet. Whenever a router receives a packet, it will execute its routing engine which results in either dropping the packet or sending a new one to another router. As such, more realistic routing behaviour is simulated.

The `Simulator` will run several threads in parallel which allow a random `Router` to execute its routing logic. This `Router` may then execute the processing of one packet. Different types of packets (`Packet`) exist, which are represented by `PacketImpl`. For example this can be an acknowledgement packet, a coordinate update packet, an election packet etc. Each `Router` has an incoming queue, in which packets can be placed. Sending a packet by a router is done by placing a packet object of a certain type on the incoming queue of another router. As such, the behaviour of a distributed set of routers is more accurately modelled. Furthermore a `CoordinateGenerator` is used to keep track of the current status of the network. For example, to not be burdened with the control of the termination of different algorithms, e.g. when all routers have elected a certain node (see Section 4.1.1), this class will take care of the announcement of a final elected node. This class is also responsible for sending initial messages into the network, or synchronising different steps of a graph embedding algorithm.

Thus the main difference between using the `Simulator` and using the `Traveler` is that the `Traveler` is more abstract as the routing logic is not directly added to the `Router` instances, but resides in the `Traveler` itself. In case the `Simulator` is used, all routing logic is encapsulated in a `Router` instance, which will receive packets, execute different routing functions based on the contents of those packets and send out new messages to other routers. On the other hand, by allowing the `Traveler` to abstract the notion of packets, the implementation of different novel algorithms can be speeded up greatly. Because of the increased complexity of the `Simulator`, it is only used in certain algorithms for generating control information (e.g. the assignment of coordinates in SALT, see Chapter 4), and not for routing purposes. However, if required, new `PacketImpl` classes can easily be created and `RouterImpl` classes can be extended with new routing logic to collaborate with the `Simulator`.

3.3 Routing behaviour

The experiments used to evaluate routing behaviour are largely based on the generation of simulated traffic between different source-destination pairs simultaneously in a multi-threaded environment. Most traffic is generated following a Pareto distribution.[2] This means that traffic bandwidth usage $f(x)$ is based on the equation

$$f(x) = \left(\frac{x_m}{x}\right)^{\alpha} \tag{3.1}$$

with $x_m = 0.1$ and $\alpha = 0.8$; x was taken randomly from the interval $[0.1, 5]$. The function f represents the traffic bandwidth usage for all links along the constructed path between source and destination. If such traffic is generated for a path $P = \langle s, n_1, n_2, \ldots, n_m, d\rangle$, then for every link $e \in P$, thus $e \in \{(s, n_1), (n_1, n_2), \ldots, (n_m, d)\}$, its used bandwidth is augmented by $f(x)$. The simulation environment maps all links $e \in E$ to a link bandwidth weight w_e and all nodes to a node bandwidth weight w_v. This is done by storing this weight inside the corresponding `Edge` and

[2] as studied in Section 2.3

`Vertex` objects. For example, when a link e is assigned traffic $f(x)$ and its current traffic weight is set to w_e, its new weight will be set to $w_e + f(x)$. The initial weight is set to 0 for every link: $w_e = 0 \ \forall e \in E$. Load balancing metrics β and λ (see Section 2.4.2) can then be calculated based on the weight of every graph edge or vertex. Therefore the β-ratio is calculated as

$$\beta_X = \frac{\left(\sum_{x \in X} w_x\right)^2}{|X| \sum_{x \in X} w_x^2} \tag{3.2}$$

with w_x the weight of the edge x or the vertex x depending on whether X represents E or V for a graph $G = (V, E)$. The λ-ratio is calculated as

$$\lambda_X = \frac{\sigma_w}{\bar{w}_x} \tag{3.3}$$

which is the standard deviation of the weights σ_w, divided by the average weight $\bar{w}_x$. These calculations are all executed by the utility class `Metrics`.

The `Traveler` will set-up different data structures to initiate the routing simulation, e.g. generating a list which can contain a stretch distribution. This `Traveler` will then call the used `TravelerImpl` which will execute a simulation loop until it is forced to terminate, e.g. by a stop action from the `RoutingPaneGUI`. In each simulation loop a source and destination `Router` are generated by a certain generation function (e.g. random generation). Inside the simulation loop, another loop is executed which contains the routing logic. At every step there exists a `current` router, which is the current hop, for which a `next` router is generated that acts as the next hop. At the end of each iteration, `next` becomes `current` for the next round. This process continues until either the destination is reached or a void is encountered. In order to achieve some sort of concurrent behaviour, the `Traveler` will execute n loop threads by which multiple routing paths are generated simultaneously. On the other hand, when looking at the `Simulator` class (the `Traveler` counterpart which models routing behaviour more accurately) we see it is only used for control information, no actual routing behaviour is implemented. However, each `Router` can easily be extended to be able to model more realistic routing behaviour. Of course then a specific `PacketImpl` class should be created to store traffic information, e.g. a weight field.

3.4 Running experiments

As can be seen in Figure 3.1, there exists a package `experiments`. This package resides outside of the actual routing simulator environment. It is able to initiate routing simulation by calling the correct interface functions. The `Main` class calls a certain `Experiment` class, which is a specific experiment with a certain set of parameters (see Chapter 6). This `Experiment` class makes sure that the bridge decouples the routing simulator from Gephi. This is required to run the routing simulator on the high-performance computer (HPC), which does not support graphics rendering. This truly highlights the importance of the bridge structure and the modularity of the software architecture. The bridge makes it easy to split off Gephi from the simulator without having to

change the actual code.

The `Experiment` classes are capable of running different parallel threads to exploit the different CPU cores of a HPC node. The number of parallel instance launched simultaneously is controlled by a maximum thread counter. Because different cores may execute different sub-experiments (e.g. a different parameter value for the same experiment) at different speeds, a `WorkQueue` class is needed. This class makes sure that each HPC core is constantly working. It essentially generates a thread pool which is running on the HPC. For example when core #25 of the 32 cores is done executing a sub-experiment, this thread pool will notice that only 31 threads are currently running, and immediately schedule a new sub-experiment with the next parameter value. As such no HPC execution time is wasted.

Because threads can only be split over multiple cores of the same node, a distributed execution environment was added to the routing simulator based on the open-source grid framework Hazelcast. Hazelcast is an in-memory data grid management framework that allows the distribution of data structures (e.g. maps or lists) across different cluster nodes.[3] It enforces memory consistency over the cluster and automatic redundancy to protect against node failures. By doing this it is possible to spread the graph structure (the hash maps of vertices and links) over multiple cluster nodes. As such multiple nodes can run different `Traveler` instances in parallel. The downside however is that this requires communication between different cluster nodes, which greatly slows down the simulation behaviour. It is clear that executing software on multiple nodes is only beneficial when there is not much communication between different threads. However, in the routing simulator built here this is not the case as every node executing the generation of a routing path has to be aware of the state of the whole network. For example, load balancing traffic has to be based on the correct node or link weights. As such the different cluster nodes are constantly communicating, which slows the simulator down to the point where it is nearly unusable. Therefore, the experiments are always ran on a single cluster node with a large set of cores.

The output of the different experiments is written dynamically to data files in order to generate data even when a simulation experiment should fail. This output mostly contains parameter-metric value couples for a wide range of parameters, for example the different load balancing metrics for a changing number of embeddings k in Forest Routing (see Chapter 6). The properties and contents of the data output files can be adjusted in each `Experiment` class. This `Experiment` class is also responsible for calculating several derived attributes such as the average value, the standard deviation, boxplot values etc.

3.5 Modularity and extensibility

The software architecture has been built with modularity and extensibility in mind. Modularity is present in the way the Gephi project itself is separated from the routing simulator. This is done by the bridge structure which implements the observer pattern. Because of this bridge it is possible to decouple the routing simulator module from Gephi such that it is possible to run simulations

[3] `http://www.hazelcast.com/`

on a computer without graphics support (e.g. the HPC). Furthermore, the source code has been divided into multiple hierarchically and logically structure packages, consisting of classes in which general logic is separated from specific logic through the use of inheritance. All different simulation functions are divided in distinct classes which incorporate code that logically belongs together.

The framework is extensible as new routing behaviour can be easily introduced by creating the required subclasses of the classes in the `general` package. For example it is possible to create a second package `be.ugent.gr.instance2` which has its own `TravelerImpl` with specific routing logic, a `RouterImpl` for holding the necessary routing information, a specific way of generating coordinates by the `CoordinateAssigner` etc. This also holds for the `Simulator` class and its corresponding `CoordinateGenerator` and `PacketImpl` subclasses.

Additional routing experiments can be set-up in the `main` package by adding new `Experiment` classes. These classes hold all necessary simulation and parameter information and ensure that the output data, as well as derived experimental data, is correctly generated into data files, as explained in the previous section.

The code-base itself is available on the Ghent University Github which enforces automatic versioning. As such new project members are able to pull existing code, modify it and push it to the Git project repositories. Git allows for roll-back actions, branching strategies, code altering restrictions and policies which are needed to avoid source code pollution. It is possible to fork the existing project to create a slightly altered parallel project.

3.6 Challenges

Creating this routing simulator was not trivial. Several challenges appeared throughout the course of its design. One challenge was enforcing correct distributed behaviour, to support the `Traveler` class in its use of multiple threads. This required the use of Java monitors and locks to synchronise function calls in a multi-threaded environment. If left out, for example the weights of the different vertices and links would not be updated correctly, thus corrupting the routing simulation data. This also required the implementation of more advanced data structures, such as concurrent hash maps or lists.

Another big challenge was the implementation of Hazelcast for executing the routing simulator on the HPC on different nodes simultaneously. This required some conversions of the used data structures. For example the hash maps had to be altered to their Hazelcast counterparts such that Hazelcast was capable of automatically spreading their contents out over multiple nodes and enforcing consistency. As these advanced data structures behave slightly different than their standard version, some parts of the source code had to be refactored. However, due to the code modularity these refactoring actions could be contained.

Also the set-up and modification of Gephi such that it would be capable of interfacing with the routing simulator was challenging. The Gephi source code had to be extensively reengineered to alter important functionality. Only after quite some tinkering was it possible to fully integrate the original routing simulation code (based on JUNG) as an add-on in Gephi.

Chapter 4

Preliminary research

This chapter outlines the research that preceded the development of the Forest Routing (FR) mechanism that will be explained in Chapter 5. The goal of this thesis is to develop a geometric routing strategy that achieves traffic load balancing in combination with a low stretch. In order to fulfil this goal, a fitting a baseline routing strategy has to be identified first. Geometric routing mechanisms were originally designed for wireless ad-hoc networks, with fixed radio transmission ranges. Therefore literature regarding geometric routing generally targets UDG-type networks. Hence geometric routing will be investigated by examining its application to UDGs first. Later on its application to scale-free networks will be evaluated, as these networks model the Internet backbone more accurately. By doing so, the possibility of embedding scale-free networks into a two-dimensional Euclidean plane is examined, which is supported by the work of Wang *et al.* (2012). Herein the viability of their routing mechanism is highlighted by testing it both on synthetically generated scale-free networks and real Internet topology datasets.

To recapitulate, when physical geographic coordinates are unavailable, or do not have any meaning[1], virtual coordinates are needed. The assignment of coordinates in a mathematical space to vertices of a graph is called a graph embedding. When examining embedding construction procedures, it is clear that these can be categorized into two classes:

1. *iterative embeddings*: the graph embedding is constructed by assigning initial seed coordinates which are then updated iteratively, guided by an objective function. For example, assigning random coordinates to all network nodes after which they are updated iteratively by trying to minimize the difference in distance in the embedding space and the shortest path length between all node pairs.

2. *structured embeddings*: to construct these embeddings, a fixed structure is used to guide the embedding procedure. Most structures are spanning trees of the underlying network. For example the embedding of Kleinberg (2007) uses a spanning tree, in which the position of every node in the tree determines the coordinates it receives, combined with a hyperbolic tiling.

[1] e.g. routers in the same datacenter will have approximately the same geographic coordinates.

Because research regarding geometric routing based on iterative embeddings is more elaborated than research of geometric routing based on structured embeddings, routing mechanisms based on the former type are examined first. Ultimately it will become clear that this first type of system is not flexible enough to be applied to varying network topologies. Therefore the choice to design a structured embedding-based mechanism as the main routing algorithm of this thesis will be substantiated by experimental results.

4.1 Simulated Annealing Label Trees

A geometric routing system, with an iterative embedding strategy, was designed based on PSVC[2], GSpring[3] and NoGeo[4]. This system embeds the network at hand into the two-dimensional Euclidean plane, denoted as $\mathbb{E}^2$. Before commencing the embedding procedure, a set of anchor nodes are elected. Each anchor node has an approximately equal distance to every other anchor in terms of shortest path length. Afterwards, the anchor nodes are assigned coordinates in $\mathbb{R}^2$. Later on, non-anchor nodes use the anchors as reference points to calculate their own coordinates. Finally a relaxation algorithm is applied to iteratively make the faces of the network more convex in the embedded space. This last step aims at reducing the overlap of coordinates of different nodes and increases the greedy routing success rate.
Because packet delivery is of utmost importance in heavy-traffic networks such as the AS-level Internet, a fallback method is constructed to guarantee a 100% delivery rate, even when voids are encountered. This fallback procedure makes use of a greedy structured embedding based algorithm called RTP[5]. The combination of the main mechanism and its backup procedure is named Simulated Annealing Label Trees (SALT).

4.1.1 Election procedures

First off, some formalism regarding election procedures as they are used in this work is in order. In an election procedure the participating nodes —in this case all nodes of the network— elect a single common node. For a graph $G = (V, E)$, the election procedure consists of a set of $|V|$ election processes $\{p_0, p_1, \ldots, p_{(|V|-1)}\}$ which

- have no shared variables
- are only able to communicate via message passing

In our case, each process p_i is mapped to a node $v \in V$, therefore we will speak of nodes rather than processes. All network nodes select the exact same unique node $v \in V$. In case the election procedure should fail[6], a new election round should be initiated. Because every process has to elect the exact same node, it should be possible to identify them uniquely. The identifier of a node $v \in V$ is called a key $\mathcal{K}(v)$ and is taken from a key set $\mathcal{K}$. An example of such a key set is the union of all possible Media Access Control (MAC) addresses, which are very suited due to

[2]Zhou *et al.* (2010)
[3]Leong *et al.* (2007)
[4]Rao *et al.* (2003)
[5]Chávez *et al.* (2007)
[6]For example, this can happen when one of the election processes crashes.

their deterministic uniqueness. Every node taking part in the election procedure has two essential variables:

- $\mathcal{E} \in \mathcal{K}$: indicates the key of the current election candidate. In case the current node v has not elected any candidate yet, this variable is assigned the key $\mathcal{K}(v)$ of the node itself.
- $\mathcal{P} \in \{\texttt{false}, \texttt{true}\}$: indicates whether the node is engaged in the election procedure or not.

Each node also stores a set of attributes, denoted as $\mathcal{A}$. This may for example be the degree of the current election candidate. The election ends only when every node possesses the same $\mathcal{E}$-value. The election procedure is entirely based on message passing. Each message contains an $\mathcal{E}_r$-value along with an attribute set $\mathcal{A}_r = \left\{\mathcal{A}_r^{(0)}, \mathcal{A}_r^{(1)}, \ldots, \mathcal{A}_r^{(n)}\right\}$. Upon receiving an election message, a node will decide whether or not to update $\mathcal{E}$ and $\mathcal{A}$. Doing this requires the receiving node to check the attributes in order of increasing i-value. As long as $\mathcal{A}_r^{(i)} = \mathcal{A}^{(i)}$, the node keeps on checking additional attributes. Once an attribute $\mathcal{A}_r^{(i+1)} \neq \mathcal{A}^{(i+1)}$ is encountered, the node will choose to update the current candidate $\mathcal{E}$ and the corresponding attributes $\mathcal{A}$ (when $\mathcal{A}_r^{(i+1)} > \mathcal{A}^{(i+1)}$) or discard the message (when $\mathcal{A}_r^{(i+1)} < \mathcal{A}^{(i+1)}$). A restriction is enforced such that at least one of the attributes in the decision mechanism is $\mathcal{K}$ to make sure all nodes eventually elect the same candidate. The node key thus acts as a tie breaker for the other attributes. When all $n+1$ attributes are equal, the message is simply discarded.

Initially for every node $v \in V : \mathcal{E} = \mathcal{K}(v) \wedge \mathcal{P} = \texttt{false}$. A node is triggered to start the election procedure, either by some external event, or upon receiving an election message when $\mathcal{P} = \texttt{false}$. When a node is first initialized, it will send its current election candidate $\mathcal{E}$ (which is initially set to its own key) to its neighbours before possibly processing a received message. On top of that it will set $\mathcal{P}$ to `true`. As will be explained in Section 5.6.2, the root election procedure takes a time of at most $t_{root} \leq 2d(\xi^+ + \mu^+)$ with d the network diameter, ξ^+ the maximum one-way communication delay and μ^+ the maximum node packet processing time. Therefore every node can assume that the election has ended when $t_{root} + \Delta$ has passed since its election process was initiated, with Δ some arbitrary time offset. After this time a node will set its variable $\mathcal{P}$ back to `false`.

4.1.2 Anchor node election

Before geometric routing is possible, the network has to be embedded into a space. Herein, every network node is assigned coordinates. By using these coordinates, distances can be calculated between pairs of nodes, which forms the basis of a geometric routing system. To engage the graph embedding procedure, every node in the network has to take part in a distributed election procedure. This election procedure enables the detection of the anchor node set by employing a hop count based distance function, similarly to the NoGeo system by Rao *et al.* (2003). It differs from this latter system in the distance function used. NoGeo elects nodes that have a maximal distance to each other, while SALT strikes a balance between maximizing the distance between each anchor node and minimizing the standard deviation of the distance of an anchor node to each other anchor node. This results in more evenly spread anchor nodes. Before the anchor nodes are elected, an initial node x is elected with the following attributes (assume a receiving node u):

- $\mathcal{A}^{(0)} = \mathcal{K}(\mathcal{E})$, first the key of the election candidate is compared. Therefore the node with the highest node key is elected.
- $\mathcal{A}^{(1)} = -h(\mathcal{E}, u)$, in which $h(\mathcal{E}, u)$ is the hop count to the candidate. Therefore the lowest hop count to the candidate $\mathcal{E}$ is stored.

When the initial node x has been elected, a first anchor node $a^{(0)}$ is elected which with the following attributes:

- $\mathcal{A}^{(0)} = h(\mathcal{E}, x)$, select a node with a maximum hop count to the initial node x.
- $\mathcal{A}^{(1)} = \mathcal{K}(\mathcal{E})$, break ties based on the node key.

After x and $a^{(0)}$ have been elected, an arbitrary number of anchor nodes $a^{(i)}$ are chosen that meet the following property

$$a^{(i)} = \arg\max_{v \in V} \left\{ \frac{\bar{h}_i(v)^2}{\sigma_h(v)} \right\} \tag{4.1}$$

with $\bar{h}_i(v)$ being the average hop count from v to all the previously calculated anchor nodes defined as

$$\bar{h}_i(v) = \frac{1}{i} \sum_{j=0}^{i-1} h\left(v, a^{(j)}\right) \tag{4.2}$$

with $h(v, a^{(j)})$ the shortest path hop count of v to anchor node $a^{(j)}$. $\sigma_h(v)$ represents the standard deviation of the hop counts $h(v, a^{(j)})$. To elect these anchor nodes, the following attributes are used (assume a receiving node u):

- $\mathcal{A}^{(0)} = \bar{h}_i(\mathcal{E})^2/\sigma_h$, first the normalized average hop count to every previously calculated anchor node is evaluated.
- $\mathcal{A}^{(1)} = -h(\mathcal{E}, u)$, this attribute triggers an update when a candidate with the same normalized average hop count is received, but the hop count to it is lower. This makes sure that every node knows the shortest path length to every anchor node.
- $\mathcal{A}^{(2)} = \mathcal{K}(\mathcal{E})$, used to break ties based on the node key.

By employing these election attributes, nodes that are spread out evenly over the network are elected as anchor nodes. This is caused by maximization of the normalized average hop count. The reason for using election attribute $\mathcal{A}^{(1)}$ is to make sure that every node knows the shortest path length to all anchor nodes that have been elected. This hop count will recursively be part of the next election rounds. The network after this stage can be viewed in Figure 4.1. It shows a UDG with 500 nodes in which the anchor nodes are elected.

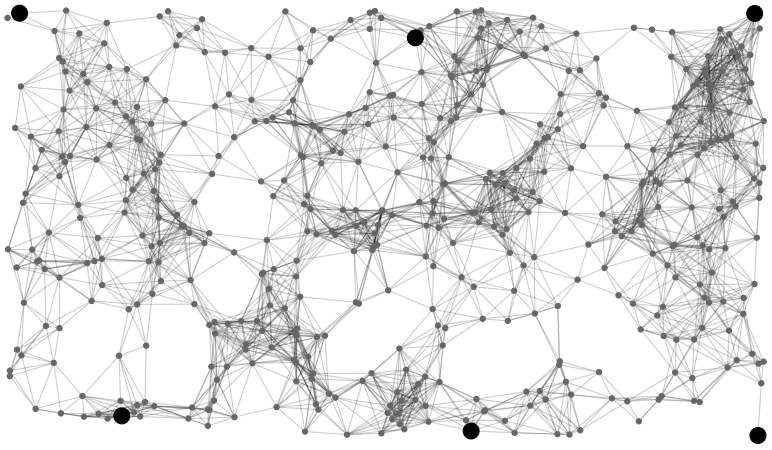


Figure 4.1: The election of six anchor nodes (the big black dots in the figure) by SALT.

4.1.3 Graph embedding procedure

Once the anchor nodes have been elected, they are embedded into the Euclidean plane by assigning coordinates in $\mathbb{R}^2$ to each of them.[7] The embedding procedure is much like the procedure described by Zhou *et al.* (2010). Assume a set of $n \leq |V|$ anchor nodes $\left\{a^{(0)}, a^{(1)}, \ldots, a^{(n-1)}\right\}$ has been elected by the previous election procedure. First, $a^{(0)}$ is placed at an arbitrary but fixed position, e.g. $a^{(0)} = (0, 0)$. Hereafter, $a^{(1)}$ is placed at a distance that equals the minimal hop count to $a^{(0)}$, multiplied by some constant scaling factor α. This hop count is known because it was the selection attribute $\mathcal{A}^{(1)}$ in the election procedure. Denote the hop count from anchor node $a^{(i)}$ to $a^{(j)}$ as $h_{i,j}$, then $a^{(1)}$ is placed somewhere on the circle with radius $R = \alpha h_{1,0}$ and centre point $a^{(0)}$. The justification is that it does not matter where the first and second anchor nodes are located because different locations only entail a translation, rotation or mirroring of the final embedding. The parameter α is a scaling factor that may be chosen arbitrarily. The third anchor node $a^{(2)}$ is placed at the point for which holds $\left(\left|a^{(2)} - a^{(0)}\right| = \alpha h_{2,0}\right) \wedge \left(\left|a^{(2)} - a^{(1)}\right| = \alpha h_{2,1}\right)$. It is thus placed at one of the crossings of the two circles centred at $a^{(0)}$ and $a^{(1)}$ with radius $R_0 = \alpha h_{2,0}$ and $R_1 = \alpha h_{2,1}$. The choice of which crossing to take is arbitrary as it will only result in a mirroring of the final embedding.

After the nodes $a^{(0)}$, $a^{(1)}$ and $a^{(2)}$ are embedded, the other anchor nodes have their coordinates assigned by a metaheuristic called simulated annealing (SA). This is different from the work of Zhou *et al.* (2010) in which a PSO optimization algorithm was used. In the former single-solution metaheuristic an objective function $f : \mathbb{R}^2 \to \mathbb{R}$ is minimized, defined as

$$f\left(a^{(i)}\right) = \sum_{j=0}^{i-1} \left(\sqrt{\left(a_0^{(i)} - a_0^{(j)}\right)^2 + \left(a_1^{(i)} - a_1^{(j)}\right)^2} - \alpha h_{i,j} \right) \tag{4.3}$$

It represents the difference between the distances in the embedded space to all the other anchor nodes known so far, and the corresponding scaled hop count distance. Thus the anchor nodes are

[7] Note that in the following text we will not make any distinction between the node and its corresponding point in the embedding space. For example say a graph $G = (V, E)$ is embedded into a space S. A vertex $v \in V$ also represents a point $v \in S$ with coordinates denoted as $(v_0, v_1, \ldots, v_n)$.

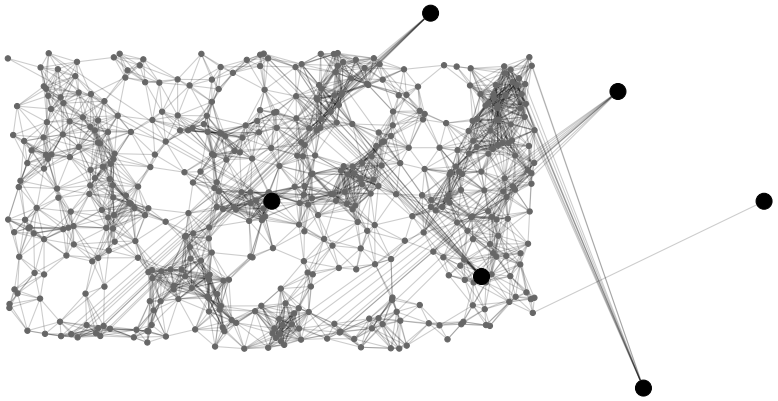


Figure 4.2: The anchor nodes are embedded into $\mathbb{E}^2$. Their position is reflected by the big black dots in the pictured graph.

essentially used as a reference point. Figure 4.2 depicts the network of Figure 4.1 after all anchor nodes have their coordinates assigned. Non-anchor nodes $v \in V$ have their coordinates assigned in the exact same way by minimizing $f(v)$, they can however take advantage of a complete set of anchor nodes.

The simulated annealing (SA) algorithm which minimizes f is inspired by the cooling of a physical substance. It mimics a substance that is strongly heated until it becomes completely fluid. Hereafter it is cooled, by which crystal structures are formed. When cooling too fast, imperfections arise. This is caused by the lack of a thermic equilibrium at each temperature. Strong and big crystals can only be formed by a long enough cooling procedure which, however, requires an infinite amount of time. SA as such, simulates the energy state of the system that is being cooled to a certain minimal temperature. This energy function is represented by the objective function.[8]. SA is a memoryless optimization algorithm.[9] This means that no information is kept regarding its previous system states. As a result it is not possible to steer the annealing process based on its history, e.g. to avoid loops. However, it has to be noted that the SA implementation used here keeps track of the global optimum found so far.

The algorithm starts at an initial temperature T_0. At every couple of iterations of the process, called the equilibrium criterion, this temperature is lowered by applying an update function. The temperature updating rule applied here is

$$T_{i+1} = \beta T_i \tag{4.4}$$

with $\beta \in [0, 1[$ the cooling rate. At every iteration a random neighbour is generated. This neighbour is then accepted or discarded, based on its objective function value. To be able to accept worsening solutions —and escaping local minima in the object landscape— the chance of accepting a worsening solution is modelled by the following Boltzmann distribution:

$$P(\Delta E, T) = e^{\frac{-\Delta E}{T}} \tag{4.5}$$

[8]Simulated annealing has been originally designed by Kirkpatrick *et al.* (1983)
[9]Although in our implementation the best solution found so far is memorized by the algorithm.

Algorithm 4.1: Embedding procedure for each vertex in SALT

input : vertex $v \in V$ of the graph $G = (V, E)$
output: coordinates $(v_0, v_1) \in \mathbb{R}^2$ for vertex v

$(v_0, v_1) \leftarrow (0, 0)$
$T \leftarrow T_{max}$
repeat
  $i \leftarrow 0$
  repeat
    r_1 and r_2 random numbers $\in [a, b]$ with $a, b \in \mathbb{R}$
    generate neighbour: $(v_0, v_1)' \leftarrow (v_0 + r_1, v_1 + r_2)$
    $\Delta E \leftarrow f((v_0, v_1)') - f((v_0, v_1))$ with f as objective function
    generate random number $r \in [0, 1]$
    if $r \leq e^{\frac{-\Delta E}{T}}$ **then**
      update the current solution: $(v_0, v_1) \leftarrow (v_0, v_1)'$
    end
    $i \leftarrow i + 1$
  until $i > i_{max}$ // i_{max} `is the equilibrium criterion`
  cooling schedule: $T \leftarrow \beta \cdot T$
until $T < T_{min}$ // T_{min} `is the stop criterion`

with T the temperature at the moment of evaluation and ΔE the difference between the new and the old objective value when hypothetically accepting the new solution. How this algorithm is applied to the embedding procedure is shown in pseudo-code in Algorithm 4.1.

As the temperature is lowered at every update, the chance of accepting a worsening solution also lowers. At the start of the heuristic search many worsening solutions are accepted in order to discover the full search space. As the search process goes on, neighbouring solutions of bad quality are increasingly rejected. The end of the search is determined by the stop criterion. This may be arbitrarily chosen, but in SALT it is determined by the reaching of a minimal temperature T_{min}. Figure 4.3 depicts the embedding of the network in Figure 4.2 after the SA algorithm has finished for all nodes.

4.1.4 Relaxation of the embedding

Finally the coordinates of the embedded nodes are relaxed by applying a spring-relaxation algorithm. Its goal is to make the graph faces more convex. For this, Hooke's law is utilized as Leong *et al.* (2007) and Zhou *et al.* (2010) describe. Based on this law a force $\vec{F}_{uv} \in \mathbb{R}^2$ is applied to all node $u \in V$, which is defined as

$$\vec{F}_{uv} = k \cdot (l_{uv} - |\vec{u} - \vec{v}|) \times n(\vec{u} - \vec{v}) \tag{4.6}$$

with $k \in \mathbb{R}$ being the spring constant; l_{uv} the relaxed spring length, and $\vec{u}, \vec{v} \in \mathbb{R}^2$ with $v \in N(u)$; $n : \mathbb{R}^2 \to \mathbb{R}^2$ a function that turns a vector into a unit vector. This force drives the distance $|\vec{u} - \vec{v}|$ towards its relaxed spring length, this constitutes the first term of Eq. (4.6) minus the factor k. A difficulty is estimating good relaxed spring lengths as only the graph topology is known. It is possible however, to estimate this length based on the principle that nodes close to each other

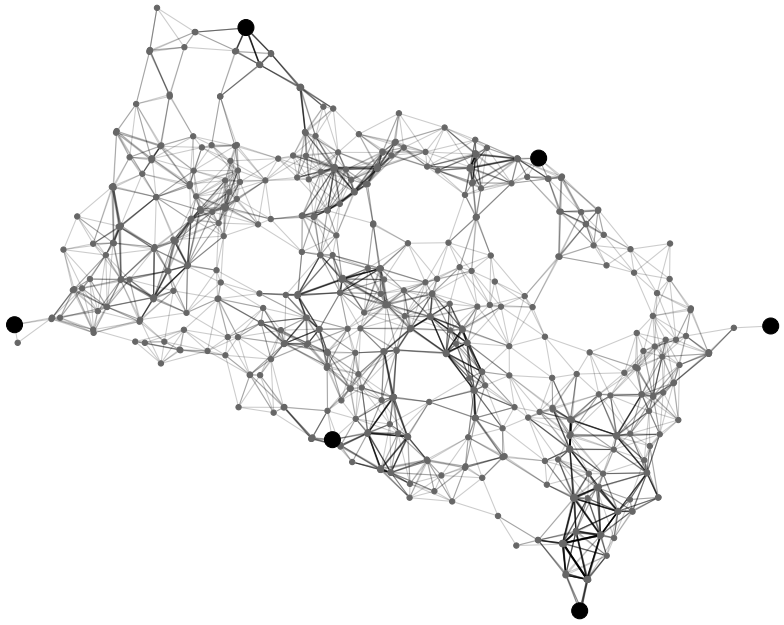


Figure 4.3: All nodes are embedded, nodes are drawn corresponding to their position in the embedding space.

in terms of hop count distance will probably have many neighbours in common. Of course the opposite is also true. Thus the length can be modelled as

$$l_{uv} = l^+ - r_{uv}(l^+ - l^-) \tag{4.7}$$

with $r_{uv} \in [0, 1]$ being the ratio of the number of common neighbours versus the total number of neighbours of the vertices u and v defined as

$$r_{uv} = \begin{cases} 0 & : N = 0 \\ \dfrac{|N_{uv}|}{N} & : \mathrm{N} > 0 \end{cases} \tag{4.8}$$

with l^+ and l^- the maximum and minimum spring lengths which can be defined arbitrarily; $N_{uv} = N(u) \cap N(v)$ is the number of distinct neighbours that the two nodes have in common; $N = (N(u) \cup N(v)) \backslash \{u, v\}$ is their total number of neighbours, excluding themselves. The forces $\vec{F}_{uv}$ for all nodes $v \in N(u)$ aggregate into a net force $\vec{F}_u$ of vertex u, calculated by summing all partial forces:

$$\vec{F}_u = \sum_{v \in N(u)} \vec{F}_{uv} \tag{4.9}$$

Now a node will update its coordinates iteratively according to the following equation

$$\vec{u} = \vec{u} + \frac{\min\left\{|\vec{F}_u|, \alpha(t)\right\}}{|\vec{F}_u|} \cdot \vec{F}_u \tag{4.10}$$

based on a counter t. At every update a node also broadcasts its coordinates to its surrounding

neighbours. These will then make use of the updated coordinates to calculate their own coordinates, better approximating the network topology. $\alpha(t)$ represents a damping constant of the spring relaxation system:

$$\alpha(t) = \begin{cases} \alpha_{max} & : t < T \\ \alpha_{max} \cdot e^{-\frac{t}{T}} & : t \geq T \end{cases}$$

with T being the maximum t value, which can be chosen arbitrarily. Good values for the constants k, l^+, l^-, α_{max} and T can be found in Leong *et al.* (2007) and Zhou *et al.* (2010). Figure 4.4 shows the embedding after the spring-relaxation algorithm has finished, applied to the network of Figure 4.3.

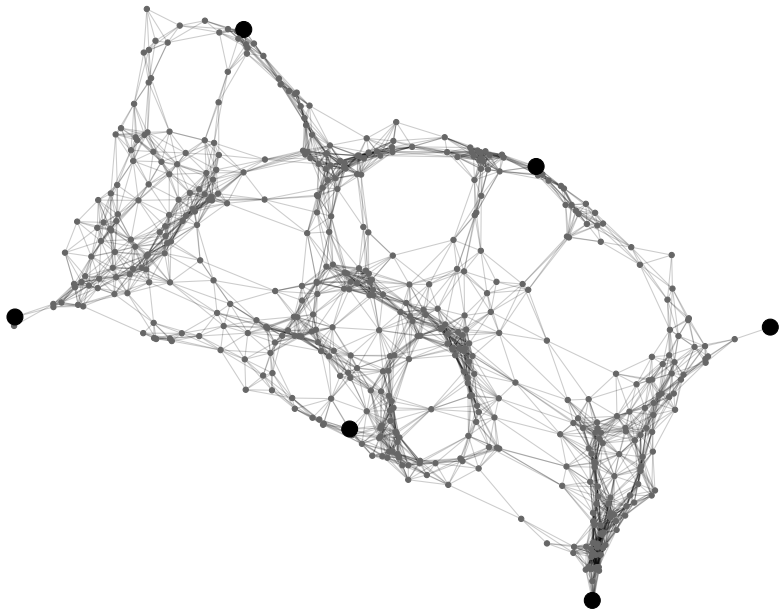


Figure 4.4: All nodes have their coordinates relaxed by the spring-relaxation algorithm. The graph layout reflects the positions of the nodes according to the embedding.

4.1.5 Results of applying SALT to generic networks

While geometric routing based on the Euclidean embedding generated by SALT with a corresponding Euclidean distance function works very well for networks that can be modelled as UDGs, it performs badly on scale-free networks, contrarily to the results of Wang *et al.* (2012). Apparently the characteristics needed to embed a network into a Euclidean plane are not present in this class of networks. Hence the embedding procedure of SALT cannot generate a graph embedding which matches the network topology. Therefore it can be concluded that geometric routing based on Euclidean embeddings can be excluded as possible candidates for geometric routing inside the Internet backbone. The results of SALT applied to different types of networks can be seen in Table 4.1.

The fact that the stretch is still relatively low is due to the backup system which employs a spanning tree-based embedding. Without this backup mechanism, the success rate of the algorithm is be

Table 4.1: Results of SALT applied to different types of network. For each network type SALT was ran 10 times for 10^6 random source-destination pairs, with Pareto distributed traffic (according to Eq. (3.1) in Chapter 6).

	UD1k	R1k	SF1k
β_E	0.534	0.074	0.060
β_V	0.168	0.001	0.001
ρ	1.029	1.930	1.710
λ_E	0.933	3.543	3.965
λ_V	2.226	28.87	31.41

99.75% for the network UD1k, while only 0.14% for R1k and 0.12% for SF1k (more information about these networks can be found in Appendix A). Therefore other types of geometric routing systems should be examined which allow routing on more general graphs. The results presented here lead us to believe that structured embeddings are more attractive and are more flexible. These structured embedding-based systems have coordinates assigned according on a fixed structure, which will be investigated next.

4.2 Structured embeddings

A second class of geometric routing systems are those based on structured graph embeddings. Herein the coordinate assignment in some mathematical space is guided by a fixed structure. For many systems this structure is embodied by a spanning tree of the underlying network. Using spanning trees is popular for two reasons. Firstly one may always construct a spanning tree of a graph, as long as it is connected. Secondly, because for every node a unique path along its ancestors towards the root node can be generated, it intuitively admits the construction of a greedy embedding. In the subsequent sections, three different tree-based geometric routing systems are investigated, each making use of a graph embedding in a different mathematical space.

4.2.1 Virtual Polar Coordinate Routing (VPCR)

Newsome & Song (2003) proposed a routing mechanism called Virtual Polar Coordinate Routing (VPCR). VPCR makes use of so-called polar coordinates, which are doubles $(\theta_a, \theta_b) \in \mathbb{R}^2 : 0 \leq \theta_a, \theta_b \leq 2\pi \wedge \theta_a < \theta_b$. These doubles represent, what the authors call, polar ranges. To construct the embedding, a spanning tree T is build. The root node r of T is first assigned a polar range $(\theta_a, \theta_b) = (r_0, r_1) = (0, 2\pi)$, the maximum polar range. The root will then calculate the polar ranges for its children $c^{(0)}, c^{(1)}, \ldots, c^{(n)}$ in the tree. These child nodes each get a non-overlapping slice of their parent's polar range:

$$c^{(i)} = \left(r_0 + i \left(\frac{r_1 - r_0}{n+1} \right), r_0 + (i+1) \left(\frac{r_1 - r_0}{n+1} \right) \right) \quad \forall i \in \{0, 1, \ldots, n\} \tag{4.11}$$

Because every polar range is unique and is a subset of the polar range of its parent, each node can be located in the tree. When the root node has allotted each of its children a set of coordinates, these will recursively compute and assign coordinates to their children, which is illustrated in

Figure 4.5.

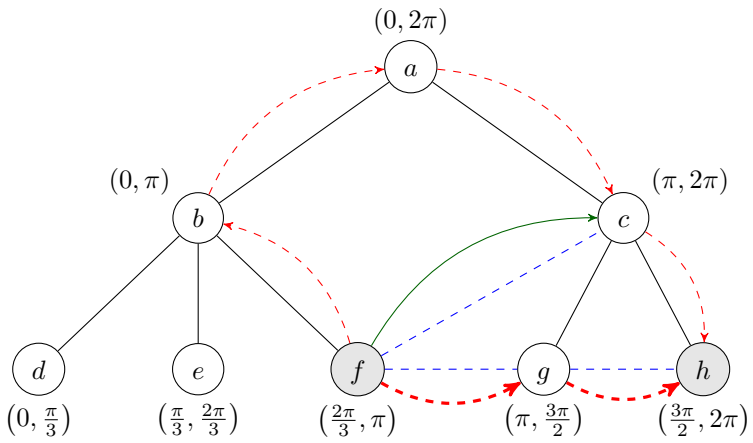


Figure 4.5: VPCR: when routing solely on the spanning tree itself, the path $\langle f, b, a, c, h\rangle$ would be taken. On the other hand when a link between f and c exists, shortcuts in the tree are possible, thus traffic is sent along $\langle f, c, h\rangle$. If a shortcut link between node f and g exists (the dashed edge (f, g)), smart routing allows the forwarding of traffic from f to g as $\left(\pi, \frac{3\pi}{2}\right)$ is closer to h than $\left(\frac{3\pi}{2}, \pi\right)$.

As the geometric routing paradigm dictates, forwarding in each node u is based on the coordinates of its local neighbourhood $N(u)$ and their distance to the destination. However in VPCR there is no distance function defined for the polar space. The forwarding decision in a node u is made by selecting the ancestor $a \in N(u)$ of the destination d with the smallest polar range. Also a set of nodes that are descendants of $D(d) \subseteq N(u)$ is selected, as well as the parent $p \in N(u)$ of the current node u. The node a is chosen such that $[d_0, d_1] \subseteq [a_0, a_1] \wedge (a_1 - a_0) = \min_{v \in N(u)}\{v_1 - v_0\}$. The set $D(d)$ is each node $w \in N(u) : [w_0, w_1] \subseteq [d_0, d_1]$. The selection of the current parent p is analogue to that of a. Forwarding in u is then done by selecting a random element $r \in D(d)$, after which a random selection between a and r is executed. When no such node can be selected because a and r do not exist, p is chosen as the next node. This node will always exist, due to the nature of the spanning tree used to form the embedding. The reason for using this complex forwarding decision is the absence of a distance function by which nodes can be evaluated. It cannot always be known how many hops a certain node is removed from the destination, only whether it will be able to reach the destination or not.

As the polar ranges are divided at every level of the tree, the precision required to represent those increases. The authors have proposed to expand the default $[0, 2\pi]$ range arbitrarily to $[a, b]$ with $a, b \in \mathbb{R}$. This does not change the asymptotic storage complexity of the coordinates though. Assuming a 3-regular tree, at every depth of the tree the range is divided by two. In a binary representation, this results in an extra required storage bit as the depth increases at the very least. The depth of a balanced 3-regular tree has a complexity of $O(\log |V|)$ for a graph $G = (V, E)$. Therefore the storage requirement has a similar complexity in an average-case scenario.

When no node a exists and the set $D(d) = \varnothing$, the traffic has to be routed upwards towards the root of T. This results in the links incident to nodes at lower tree depths getting congested. A

traffic bottleneck is formed around the root. The authors report a solution called *smart routing* in which traffic may be routed towards nodes with polar ranges that are closer to the destination (see Figure 4.5). This is however largely assumes that the underlying network is a UDG, because the polar ranges should correspond with some layout in the two-dimensional Euclidean plane. This is also hard to achieve in a distributed setting. Smart routing seems to diminish, but not eliminate the root traffic hotspot.[10]

4.2.2 Hyperbolic routing

Kleinberg (2007) has proven that there exists a greedy embedding into the hyperbolic plane $\mathbb{H}^2$ for a d-regular tree for all $d \geq 3$. The algorithm presented by Kleinberg requires that the node with the largest connectivity is known, e.g. by an election process. A more general online greedy embedding scheme for the hyperbolic plane was proposed by Cvetkovski & Crovella (2009) which is shown in Algorithm 4.2 and will be used in this thesis.

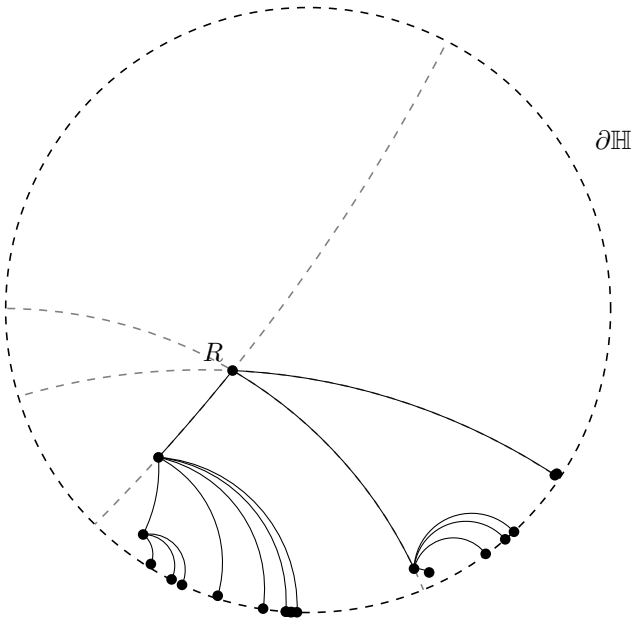


Figure 4.6: Graph embedding into $\mathbb{H}^2$ based on Algorithm 4.2. The nodes are drawn in the Poincaré disk model based on their assigned coordinates; the solid lines represent the tree edges while R is the root vertex.

The hyperbolic space $\mathbb{H}^2$ employed in the hyperbolic routing system is represented in the Poincaré disk model. Herein points in the n-dimensional hyperbolic plane $\mathbb{H}^n$ are points on a n-dimensional unit disk. In this model lines as known in the Euclidean space $\mathbb{E}^2$ are parts of circles that are inside of, and orthogonal to, the unit circle in $\mathbb{H}^2$. The centre forms the origin while the points on the unit disk $\partial\mathbb{H}$ are considered ideal points. Ideal points in the Euclidean space would be points at infinity. The distance between two points u and v with coordinates in $\mathbb{R}^2$ with a distance to the

[10] Newsome & Song (2003)

origin lower than one, is given by

$$\delta(u,v) = \operatorname{arccosh}\left(1 + 2\frac{|u-v|^2}{(1-||\vec{u}||^2)\,(1-||\vec{v}||^2)}\right) \tag{4.12}$$

with $\vec{u}$ and $\vec{v}$ the vectors associated with the points u and v. This represents the curved distance along the circle segment of the circle passing through both points, while being orthogonal to $\partial\mathbb{H}$. The biggest downside to using hyperbolic routing is perhaps that the distance calculation is computationally complex when compared to other schemes like RTP, which is discussed next.

Algorithm 4.2: Online greedy embedding scheme

input : a graph $G = (V, E)$ with a spanning tree $T = (V, E') : E' \subset E$ rooted at $r \in V$; every vertex knows its parent and children

output: coordinates $(v_0, v_1) \in \mathbb{R}^2 : -1 < v_i < 1 \;\forall\; v \in V$ in the Poincaré disk model

1 assign an initial coordinate double (r_0, r_1) to the root node r as specified by Cvetkovski & Crovella (2009)
2 set initial angles for root node $\alpha_r \leftarrow \pi$ and $\beta_r \leftarrow 2\pi$
3 calculate root node numbers $a_r, b_r \in \mathbb{C}$ with $a_r \leftarrow e^{i\alpha_r}$ and $b_r \leftarrow e^{i\beta_r}$
4 **foreach** $v \in V$ **do**
5 the parent of v, $p(v)$ sends its coordinates to v along with $\alpha_{p(v)}$ and $\beta_{p(v)}$.
6 starting angle for v is set to $\alpha_v \leftarrow \alpha_{p(v)}$
7 ending angle is the average of both parent angles $\beta_v \leftarrow \dfrac{\alpha_{p(v)} + \beta_{p(v)}}{2}$
8 parent starting angle is updated $\alpha_{p(v)} \leftarrow \beta_v$
9 complex numbers are calculated for v as $a_v \leftarrow e^{i\alpha_v}$ and $b_v \leftarrow e^{i\beta_v}$
10 calculate $m \leftarrow \dfrac{a+b}{2}$, then $R^2 \leftarrow \dfrac{1}{|m|^2} - 1$, finally $h \leftarrow \dfrac{1}{m^*}$ with m^* the complex conjugate of m
11 coordinates for v are calculated: $\dfrac{R^2}{p(v)^* - h^*} + h$ with $p(v)$ now representing the coordinates of its parent
12 α_v is prepared for recursive coordinate assignment of its children $\alpha_v \leftarrow \dfrac{\alpha_v + \beta_v}{2}$
13 **end**

4.2.3 Routing with Position Trees (RTP)

In Routing with Position Trees (RTP) an underlying spanning tree is labelled. In a labelling process every parent assigns a unique number to its children. These children will then construct a new label based on this unique number and the labels of their parents. The root node is assigned an arbitrary number as a label. These labels then act as coordinates that can be used in geometric routing. The distance between two nodes is the shortest path between them when only taking into account edges of the spanning tree. Routing based on labelled trees will be explained in great detail in Chapter 5.

4.2.4 Comparison

Hyperbolic routing was compared with RTP and VPCR. All systems had one single tree that was rooted at the highest-degree node. A breadth-first tree-growing process made sure a tree of minimal depth was formed in all occasions. Pareto distributed traffic (according to Eq. (3.1) in Chapter 6) was generated between random source-destination pairs of the network SF500 (see Appendix A for more information). The different routing schemes were evaluated on load balancing behaviour and average stretch, the results are presented in Table 4.3.

Table 4.2: Results of comparison between hyperbolic routing, RTP and VPCR on the graph SF500. The experiment involved 10 spanning tree constructions rooted at the highest-degree node for every routing mechanism. Then 10^6 random source-destination pairs were generated between which Pareto distributed traffic was simulated.

	hyperbolic	RTP	VPCR
β_E	0.127	**0.211**	0.047
β_V	0.107	**0.127**	0.058
$\bar{\rho}$	1.481	**1.325**	1.514
λ_E	2.615	**1.933**	4.485
λ_V	2.880	**2.611**	4.028

Table 4.3: Fraction of identical paths when using the same underlying spanning tree for the same source-destination pairs when comparing the algorithms two by two.

between	fraction
hyperbolic-label	9.6%
polar-hyperbolic	7.7%
tree-polar	9.4%

These results show that using RTP scores the best in terms of stretch and load balancing, while having a computational complexity far less than hyperbolic routing. It also avoids the requirement of floating-point coordinate representations by relying solely on integers. To test whether the differences in routing performance could be due to the way the spanning tree was formed, an experiment was set up that generated embeddings for the difference routing system using the exact same spanning tree. Next traffic was routed for random source-destination pairs for each of the routing systems. The results of this (see Table 4.3) show that only approximately 10% of the paths were identical. Therefore we are certain that the systems truly generate different routing paths, even when using the same underlying tree. This leads to the conclusion that the difference in performance is due to the design of the routing algorithms. Because of the beneficial results of RTP and its low computational complexity, RTP will be chosen as a foundation to develop the main routing mechanism of this thesis, called Forest Routing (FR), which will be explained in the next chapter.

Chapter 5

Forest Routing

Based on the preliminary research set forth in the previous chapter, a family of geometric routing systems called Forest Routing (FR) is proposed, which forms the main contribution of this thesis. This chapter will focus on the design of the algorithms while the experimentation and evaluation of their different parameters is done in Chapter 6.

5.1 Introduction

The FR system is based upon Routing with Position Trees (RTP)[1] and the tree labelling procedure of Korman *et al.* (2002). RTP was chosen over hyperbolic routing[2] or VPCR[3] based on the results of Table 4.3 and its low computational complexity. In RTP, nodes are labelled according to a spanning tree of the network. When interpreting these labels as coordinate sets, combined with an appropriate distance function, RTP can be seen as a geometric routing algorithm in which the node labelling process corresponds to the graph embedding procedure. Because the labelling procedure is entirely dependent on the spanning tree used, it can be classified as a structured embedding procedure. This is in contrast to the Simulated Annealing Label Trees (SALT) algorithm described earlier which was iterative by nature. Spanning trees are strong candidates to form a structured embedding as they are naturally present in any connected graph. As our target network, the Internet backbone, is a connected structure, it is always guaranteed that a spanning tree can be formed. Tree-like structures also naturally emerge from the hierarchical topology of scale-free graphs[4]. Another advantage of using labelled spanning trees is the preservation of information regarding the relative position of a node in its underlying tree. This makes it easy to define a distance function. In the subsequent sections the terms graph and network, vertex and node as well as link and edge will be used alternately to describe the same structures. To guide the reader, a roadmap for the upcoming sections is given in Figure 5.1 and at the end of every major section, a small recapitulation is given.

[1] Chávez *et al.* (2007)
[2] Kleinberg (2007)
[3] Newsome & Song (2003)
[4] Papadopoulos *et al.* (2010)

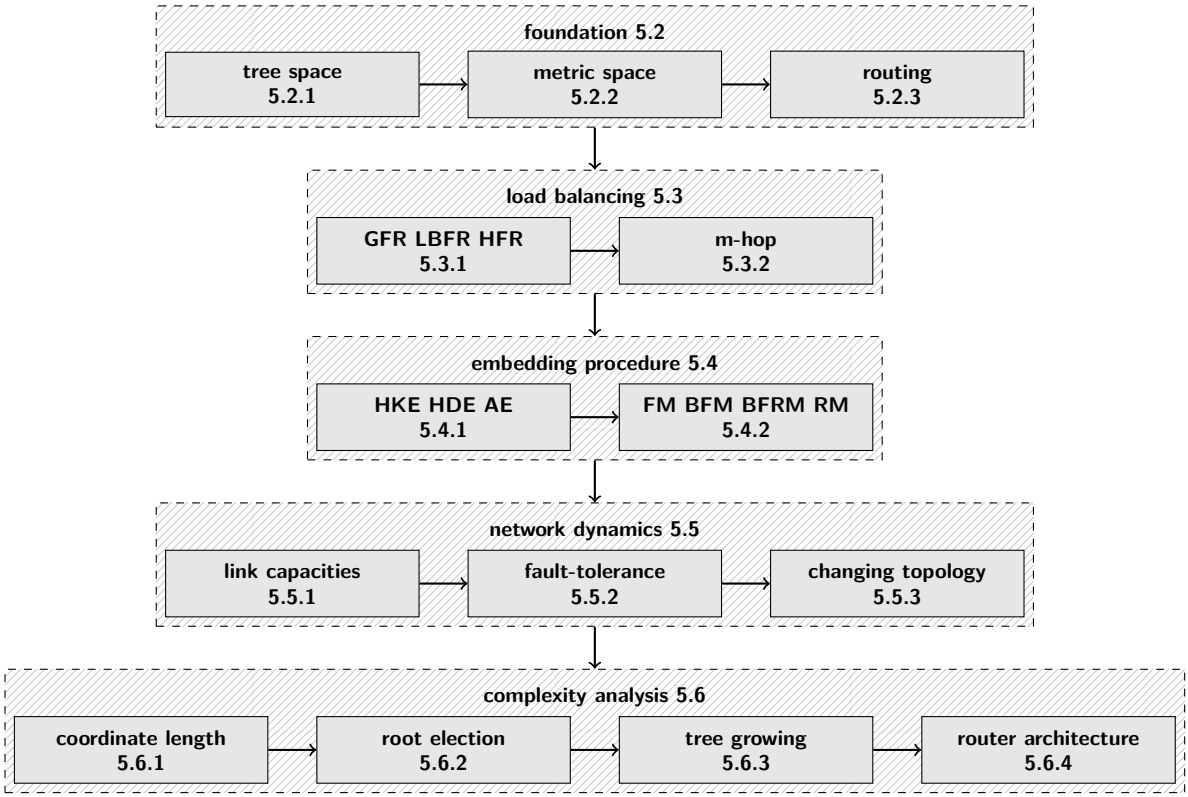


Figure 5.1: A roadmap of the structure of this chapter, it reveals how the sub-components of FR will be explained.

5.2 Foundation

In this section a theoretical foundation regarding geometric routing based on the labelling process of Korman *et al.* (2002) and Routing with Position Trees (RTP) of Chávez *et al.* (2007) will be set-up, which forms the basis of the FR system.

5.2.1 Tree space

Geometric routing systems make use of the graph embedding concept. Such an embedding is a mapping between the vertices of a graph and certain mathematical space as defined in Definition 2.7. Every vertex is assigned a unique tuple of coordinates corresponding to a point in the space. For example a graph can be embedded into the two-dimensional Euclidean plane $\mathbb{E}^2$ with Cartesian coordinates in $\mathbb{R}^2$ assigned to each vertex. However, instead of the Euclidean space, the FR mechanism makes use of a so-called tree space, denoted as $\mathbb{T}$, which will be defined formally later on. The main advantage of using $\mathbb{T}$ is the existence of a greedy embedding for every connected graph. This greedy embedding of a graph $G = (V, E)$ into $\mathbb{T}$ is a mapping $\mathcal{T} : V \to \mathbb{T}$. To explain what this space $\mathbb{T}$ represents we will depart from the construction of a possible greedy embedding, denoted as $\mathcal{T}$, into $\mathbb{T}$.

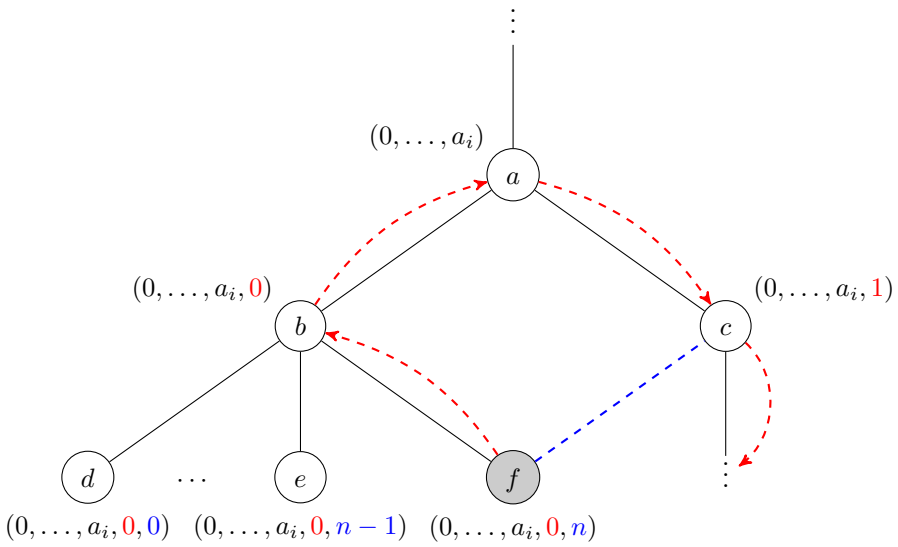


Figure 5.2: Coordinates assigned to a tree in $\mathbb{T}$. The red dashed line represents routing on the spanning tree with f as the source node and a destination node that is a predecessor of c. The blue dashed line represents a shortcut in the tree, when employing the tree distance metric this shortcut would be taken to route directly to c instead of the path $\langle b, a, c \rangle$.

To construct such a greedy embedding[5], a spanning tree $T = (V, E')$ of G is needed with $E' \subseteq E$ and a root vertex r. Such a spanning tree can be constructed by an arbitrary tree-generation procedure, e.g. the procedure explained in Section 5.4. Hereafter every vertex $u \in V$ can be assigned a set of unique coordinates $\mathcal{T}(u) \in \mathbb{T}$ by running a labelling process[6]. This labelling process starts by assigning r, the spanning tree T root node, the coordinate 1-tuple (0). This root node is said to reside at depth 1 in the tree. One level deeper in the tree, the root's children are assigned a number in $\{0, 1, \ldots, (d_G(r) - 1)\}$. From this follows that all vertices at depth two have coordinate 2-tuples (doubles) in the shape of $(0, a_1)$. Recursively, vertices at depth n have coordinate n-tuples $(0, a_1, a_2, \ldots, a_{n-1})$. This labelling process is depicted in Figure 5.2. Note that the coordinates of a node u can be computed solely by using information about its local 1-hop neighbourhood $N(u)$, simply by means of message passing.

Now the mathematical space $\mathbb{T}$ can be defined as the target space of the previously explained embedding. This means that every node corresponds to a point in $\mathbb{T}$. More generally, the coordinates of a vertex u are represented by an n-tuple in $\mathbb{N}^n$ for $n \leq d$, with d the depth of T and n the depth at which the vertex resides in T. The arity of the coordinate tuple therefore holds information about the vertex' relative position in the spanning tree T with a specific root. Therefore $\mathbb{T}$ can be defined as

$$\mathbb{T} = \bigcup_{n \in \mathbb{N}} \left((0)^\frown \mathbb{N}^n \right) \tag{5.1}$$

The function $(^\frown) : \mathbb{N}^m \times \mathbb{N}^n \to \mathbb{N}^{m+n}$ is the concatenation of two tuples. Note that $\mathbb{T}$ is very

[5] Chávez *et al.* (2007)
[6] This has been formally described by Korman *et al.* (2002).

general, and as such one may very well construct embeddings into $\mathbb{T}$ without making use of an underlying spanning tree. Though these would classify as graph embeddings into $\mathbb{T}$, they are not necessarily greedy. Whenever an embedding in $\mathbb{T}$ is denoted as $\mathcal{T}$ in this text, a greedy embedding is meant, constructed by employing a spanning tree T as previously explained.

5.2.2 Tree metric space

Next to the tree space and its corresponding graph embedding, the core of a geometric routing system consists of a distance function. The combination of a mathematical space and a distance function is called a metric space, formally defined subsequently.

Definition 5.1. *For a set X and a function $f : X \times X \to \mathbb{R}$ such that the following conditions hold $\forall u, v, w \in X$:*

1. $f(u, v) \geq 0 \wedge f(u, v) = 0 \Leftrightarrow u = v$
2. $f(u, v) = f(v, u)$
3. $f(u, w) \leq f(u, v) + f(v, w)$

Then f is called a metric and the double (X, f) is called a metric space.

For example, when $X = \mathbb{R}^d$ with $f(u, v) = \sqrt{\sum_{i=0}^{d-1} (u_i - v_i)^2}$, then (X, f) is the d-dimensional Euclidean metric space. In the same manner the tree space can be combined with its corresponding distance function: $X = \mathbb{T}$ and $f = \delta$. The distance δ between two points in $\mathbb{T}$ is defined as follows. It is the sum of the hop counts between the source and destination vertex and their common ancestor, which corresponds to the shortest path between the two vertices when only using edges in E' of the tree $T = (V, E')$. This is illustrated with an example in Figure 5.2. Assume a source vertex e and a destination vertex c, now their common ancestor is vertex a. The hop count of e to a equals 2 and the hop count from c to a equals 1. The distance in the tree space is thus $\delta(e, c) = (2 + 1) = 3$. This distance can thus be written down as

$$\delta(u, v) = \text{depth}(u) + \text{depth}(v) - 2 \cdot \text{depth}(w) \tag{5.2}$$

with w the common ancestor of u and v in T. The function depth(u) represents the depth of u in the tree T, which is the position of u on the shortest path between the root of T (start vertex) and u (end vertex). This distance can be easily calculated using implicit information included in the coordinates of a node. The length of the coordinate tuples represent the depth at which the vertex resides in the tree used to build the embedding. To illustrate this, assume two vertices u and v with coordinates

$$\mathcal{T}(u) = (a_0, a_1, a_2, b_3, \ldots, b_n),\ n \in \mathbb{N} \tag{5.3}$$

$$\mathcal{T}(v) = (a_0, a_1, a_2, c_3, \ldots, c_m),\ m \in \mathbb{N} \tag{5.4}$$

then the common part is the triple (a_0, a_1, a_2). For the remainder of this text, no distinction will be made between the notations u and $\mathcal{T}(u)$, because every vertex corresponds to exactly one set of coordinates in its embedding space. From the context in which the symbol is used it should be

clear whether the vertex itself is meant or its corresponding point in the embedding. The distance equation can therefore be re-written as

$$\delta(u, v) = |u| + |v| - 2|\phi(u, v)| \tag{5.5}$$

with $\phi : \mathbb{N}^n \times \mathbb{N}^m \to \mathbb{N}^p$ $(p \leq \min\{m, n\})$ the function generating the largest common prefix of two tuples; $|u|$ represents the length of the coordinate tuple u. This results in a distance function $\delta : \mathbb{T} \times \mathbb{T} \to \mathbb{N}$ that, together with the space $\mathbb{T}$, forms the metric space $(\mathbb{T}, \delta)$ which can easily be proven.

5.2.3 Routing

After a graph embedding $\mathcal{T}$ into $\mathbb{T}$ has been constructed, the underlying tree T used for the construction is no longer needed and is thus discarded. This means that routing happens entirely by the principles of geometric routing, based on a graph embedding $\mathcal{T}$ and a distance function δ. It is important to note that routing based on $\mathcal{T}$ is different than routing on spanning tree T. In spanning tree routing a routing path can only use tree edges. However here, assuming we have a spanning tree $T = (V, E')$ of a graph $G = (V, E)$ with $E' \subseteq E$, *shortcuts* can be taken. These shortcuts are edges e for which holds $e \in E \wedge e \notin E'$. As a result, traffic congestion in links between nodes at lower tree depths[7] is partially avoided, which is a major downside of spanning tree routing. This is thus a form passive load balancing. However, the effect of using shortcuts alone is not enough to obtain true traffic load balancing. How active load balancing using link load information can be achieved will be explained in Section 5.3.

5.2.4 Recapitulation

At this point a baseline geometric routing mechanism has been established. This baseline mechanism consists of a theoretical tree space $\mathbb{T}$ with a corresponding greedy embedding $\mathcal{T}$ and a distance function δ. In the following sections this baseline mechanism will be extended to tackle the problem of load balancing.

5.3 Load balancing

As mentioned earlier, the spanning tree root node suffers from high congestion on its incident links under a uniform traffic matrix. A possible solution would be to utilize the ad-hoc routing decision making of geometric routing schemes for every packet independently. Allowing nodes to alter their forwarding decisions based on the current link load would potentially lead to load balancing. This principle of actively manipulating traffic according to the current load in order to reduce congestion is called active load balancing. But first, a maximum of passive load balancing should be established. The final routing scheme should then maximize the total load balancing effect by effectively combining these two types.

[7] Newsome & Song (2003)

5.3.1 Multiple embeddings

A first step towards load balancing is the use of k spanning trees $T_i = (V, E'_i)$ with $E'_i \subseteq E \ \forall i \in \{0, 1, \ldots, k-1\}$ to form k greedy embeddings of the same graph $G = (V, E)$ into $\mathbb{T}$ instead of only one.[8] The different embeddings are denoted as $\mathcal{T}_i$ with $i \in \{0, 1, \ldots, k-1\}$. The notation δ is now used to denote the k-tuple of distances in all of the k embeddings. Therefore δ_i represents the tree space distance for the i-th embedding $\mathcal{T}_i$. By using a multi-embedding it might be possible to avoid the traffic congestion at nodes residing at lower tree depths. The reason is two-fold: firstly, because of the existence of k different root nodes, traffic congestion should be divided amongst them. Secondly due to having multiple embeddings, the chances of having shortcuts available should increase, yielding a less crowded root and an overall lower stretch. The trade-off made here is an increased storage requirements of the packet headers given that the destination now has a set of coordinates for every embedding, in favour of more forwarding freedom. It also leads to an increased computational complexity of the forwarding decision making. However due to its high parallelizability, increased processing time can be mitigated as will be shown in Section 5.6.4.

5.3.1.1 Greedy Forest Routing (GFR)

A straightforward way of routing with multiple embeddings would be to allow a node to freely alternate between different embeddings because of their individual greediness. However, this naive selection mechanism fails because of the lack of mechanism that avoids the introduction of cycles in routing paths. When routing along a distance-decreasing path in $\mathcal{T}_i$, the distance in an alternative embedding $\mathcal{T}_j$ may increase. A node further down the routing path may decide to use $\mathcal{T}_j$ and routes the packet back towards its starting point. Enforcing the restriction that the distance should decrease in every embedding is hardly an option, as it cannot be guaranteed that this aggregation of greedy embeddings is greedy on its own. Also the exclusion of nodes that have already been reached is not an option as it cannot be guaranteed that a packet will not encounter a void. A solution is ensuring that the path decreases its lowest distance (out of all the embeddings) at every hop. This way of working is similar to Tree Cover Based Geographic Routing with Guaranteed Delivery (TCGR)[9] in which the minimum of distance for multiple embeddings is used to guarantee packet delivery. For this reason a new distance function $\epsilon : \mathbb{T}^k \times \mathbb{T}^k \to \mathbb{N}$ is defined as

$$\epsilon(u, v) = \min_{0 \leq i < k} \{\delta_i(u, v)\} \ \ \forall u, v \in V \tag{5.6}$$

The multiple embeddings are now treated as a single embedding in the k-dimensional space $\mathbb{T}^k$. Now a new relaxed-metric space can be defined as $(\mathbb{T}^k, \epsilon)$. It cannot be regarded as true metric space because the triangle-equality (property three in Definition 5.1) does not hold any longer, hence we call it relaxed. Routing is now done greedily based on a graph embedding into the space $\mathbb{T}^k$ with the ϵ distance function. When a node has multiple neighbours with the same ϵ-distance, a random forwarding selection is made among them. This routing mechanism is called Greedy Forest Routing (GFR), which is a special case of the final Forest Routing (FR) system. The goal is to obtain a decreased stretch in combination with improved load balancing behaviour compared to

[8] The use of multiple tree embeddings is supported by the work of Tang *et al.* (2010) in which the authors use multiple embeddings to achieve passive load balancing for greedy routing.

[9] Tang *et al.* (2010)

routing using a single tree-based embedding. This should logically follow from the fact that there is a larger chance of finding a shortcut in k embeddings compared to using a single embedding. As a result shorter paths should be taken which avoid the root node, a natural hotspot in $\mathcal{T}$. Furthermore traffic should now be spread over k different root nodes, dividing the original root hotspot among different areas of the network. The effect of using different k-values on the stretch and the load balancing behaviour of GFR is analysed in Section 6.1 of the results chapter.

5.3.1.2 Load Balanced Forest Routing (LBFR)

To supplement the passive load balancing behaviour emerging from a multi-embedding, an active load balancing technique has been developed. In this system, nodes $u \in V$ make use of information regarding the traffic load of their incident edges $e \in I(u)$. Solely using local link load information is advantageous as it is scalable by nature and therefore fitting for a large-scale distributed setting. This new system called Load Balanced Forest Routing (LBFR) relaxes the distance-decreasing requirement of GFR. Routing on varying embeddings $\mathcal{T}_i$ independently is now allowed. Because naive switching between embeddings may introduce cycles, it is guided by an auxiliary function κ. Hereby the coordinate system that is inherently oblivious (see Definition 2.6) is extended with a non-oblivious function that is dependent on the path followed by a packet. The value of κ of the previous node along a path is sent on to the next node by including it as additional information in the packet header. As such every node is able to calculate their own κ-value without having to query all nodes along the travelled path. The function κ makes use of a function $\delta^* : \mathcal{P} \times \mathbb{T}^k \to \mathbb{N}^k$ which outputs a k-tuple storing the minimal distance to a destination d attained by a packet so far along its routing path P_u, before arriving at the current node u, for each of the k embeddings. The set $\mathcal{P}$ represents the union of all possible paths of the network. The i-th element of δ^* is denoted as δ_i^*. Assume a packet has been routed along the path $P = \langle p_0, p_1, \ldots, p_n \rangle$ towards a destination vertex d, the function κ is then of the type $\mathcal{P} \times \mathbb{T}^k \to \mathbb{N}$ and is defined as

$$\kappa(P_{p_n}, d) = \sum_{i=0}^{k-1} \delta_i^* \left(P_{p_n}, d\right) \tag{5.7}$$

with $\delta^*(P_{p_n}, d)$ a function of the type $\mathcal{P} \times \mathbb{T}^k \to \mathbb{N}^k$ that is defined recursively as

$$\delta_i^*(P_{p_0}, d) = \delta_i(p_0, d) \tag{5.8}$$

$$\delta_i^*(P_{p_n}, d) = \min\{\delta_i^*(P_{p_{n-1}}, d), \delta_i(p_n, d)\} \quad \forall i \in \{0, \ldots, k-1\} \wedge \forall n > 0 \tag{5.9}$$

Herein P_u represents the path P until u has been reached, it consists of the vertices that a packet arriving at u has traversed. p_0 is the source vertex of the path P. Each of the minimum distances of the k embeddings is thus represented by an element $\delta_i^*(P_u, d)$. The LBFR system will route a packet along a distance-decreasing path for each $\mathcal{T}_i$ individually but with a restriction that κ has to decrease strictly monotonically along the routing path:

$$\kappa(P_{p_n}, d) < \kappa(P_{p_{n-1}}, d) < \ldots < \kappa(p_0, d) \tag{5.10}$$

The LBFR forwarding decision making process is depicted in Algorithm 5.1. Herein the compu-

tation of the recursive function δ^* is done based on the δ^*-value of the previous node, which has been added to the packet header. By doing this, κ can be easily computed at every node. All neighbouring nodes whose κ-function value is strictly lower than the node u making the forwarding decision are added to a set, denoted as $S(u)$. A packet forwarded to any element of $S(u)$ will eventually arrive at its destination. In order to balance traffic on each node's outgoing links, the node $v \in S(u)$ is chosen for which $(u, v) \in I(u)$ has the lowest load. Because LBFR focuses solely on link load balancing, its link load balancing behaviour should improve quickly as k increases. As a side-effect, the stretch might become very large because the distance gain at every routing hop is sacrificed in favour of load balancing. The effect of using different k-values on the stretch and the load balancing behaviour of LBFR is analysed in Section 6.2 of the results chapter.

Algorithm 5.1: Load Balanced Forest Routing (LBFR): forwarding decision making procedure

input : vertex $v \in V$ of the graph $G = (V, E)$ at which a forwarding decision needs to be taken for a packet `packet` received from vertex $u \in V$; v knows its neighbours $N(v) \subseteq V$ and the current load of its currently incident edges $I(v) \subseteq E$; all vertices have been embedded into $\mathcal{T}_i$ for $0 \leq i < k$.

output: packet is forwarded to a vertex $n \in N(v)$

```
1  δ*(u) (k-dimensional array) ← packet.getMinDistances()
2  δ*(v), S(v) ← ∅
3  κ(v) ← 0
4  foreach i ∈ {0, 1, ..., k − 1} do // for all embeddings
5  |  δ*_i(v) ← min{δ*_i(u), δ_i(u, d)}
6  |  κ(v) ← κ(v) + δ*_i(v)
7  end
8  foreach i ∈ {0, 1, ..., k − 1} do // for all embeddings
9  |  foreach w ∈ N(v) do
10 |  |  if κ(w) < κ(v) then // the κ(w) calculation based on δ*(v) is not shown
11 |  |  |  S(v) ← S(v) ∪ {w}
12 |  |  end
13 |  end
14 end
15 minLoad ← +∞
16 R(v) ← ∅
17 foreach w ∈ S(v) do
18 |  if L(v, w) < minLoad then // load between on link (u,w)
19 |  |  R(v) ← ∅
20 |  |  minLoad ← L(v, w)
21 |  |  R(v) ← R(v) ∪ w
22 |  else if L(v, w) = minLoad then
23 |  |  R(v) ← R(v) ∪ w
24 |  end
25 end
26 n ← random element from R(v)
27 packet.setMinDistances(δ*(v)) and send packet to n
```

In order to guarantee delivery for every source-destination pair the following theorems are introduced.

Theorem 5.1. *Let $G = (V, E)$ be a graph with k embeddings $\mathcal{T}_i$ for $0 \leq i < k$ in the metric space $(\mathbb{T}, \delta)$. Let d be the destination node, then for every path $P \in \mathcal{P}$ of a graph with $v \in V$ as last element and where d has not been reached yet, thus $d \notin P$, the set of neighbours $S(v)$ for which the value of the κ-function strictly decreases is not empty.*

Proof. Assume a packet arriving at a vertex v by following a path $P = \langle \ldots, u, v \rangle$. Assume this packet has to be forwarded to a destination vertex d and that $d \notin P$. This means that, making use of the notations in Algorithm 5.1, $S(u) \neq \varnothing$. Therefore $\kappa(v) < \kappa(u)$. Because of the definition of κ as the sum defined by Eq. (5.7): $\exists i \in \{0, 1, \ldots, k-1\} : \delta_i^*(P_v, d) < \delta_i^*(P_u, d)$. Combining the definition of δ^* in Eq. (5.9) with the definition of the min-function gives $\delta_i(v, d) < \delta_i^*(P_u, d)$ and $\delta_i^*(P_u, d) \leq \delta_i(u, d)$. Therefore, again because of Eq. (5.9), $\delta_i^*(P_v, d) = \delta_i(v, d)$. Also, $\delta_i(v, d) < \delta_i(u, d)$ which means that the distance towards d in embedding $\mathcal{T}_i$ has decreased. Because $\mathcal{T}_i$ is a greedy embedding and Definition 2.8: $\exists w \in N(v) : \delta_i(w, d) < \delta_i(v, d)$. Thus because of Eq. (5.9) $\delta_i^*(P_v \frown w, d) = \delta(w, d)$ and therefore $\delta_i^*(P_v \frown w, d) < \delta_i^*(P_v, d)$. Combining this with Eq. (5.7) leads to $\kappa(P_v \frown w, d) < \kappa(P_v, d)$ based on the fact that δ_i^* never increases along a path. From this follows: $S(v) \neq \varnothing$. As such, any element from $S(v)$ is a suitable next vertex to which the packet can be forwarded without violating the LBFR restrictions.
This theorem also holds for a source vertex s. Because of Eq. (5.8), every value δ_i^* is equal to the distance δ_i. Therefore any vertex for which the distance towards the destination decreases (and such a vertex exists due to every $\mathcal{T}_i$ being a greedy embedding) leads to a lower κ-value. Thus at the source vertex $S(s) \neq \varnothing$. □

Theorem 5.2. *The path followed by a packet routed on a graph $G = (V, E)$ by LBFR is never a cycle.*

Proof. Assume a destination vertex d and a packet travelled along $P = \langle \ldots, u, v, \ldots, w \rangle$ arriving at $w \in N(v)$. When arriving at v for the first time, $\delta_i^*(P_v, d) \leq \delta_i(v, d)$ because of Eq. (5.9). Since the values of δ^* can never increase due to the definition of the min-function, upon calculating the κ-function value for the second time for vertex v (this time from w): $\nexists i \in \{0, 1, \ldots, k-1\} : \delta_i(v, d) < \delta_i^*(P_w, d)$ because if there would exists such an i then δ^* would already have been updated to this value the first time the packet has arrived at v. Thus the κ-value cannot decrease the second time v is encountered. Therefore no vertex can appear twice along the path followed by a packet routed by LBFR which enforces κ to be strictly monotonically decreasing along a routing path. □

Theorem 5.3. *A packet routed according to the principles of LBFR on a graph $G = (V, E)$ will arrive at its destination.*

Proof. Because κ is strictly monotonically decreasing and the initial κ-value at the source vertex is finite (assuming $|V|$ is finite), it will become 0 after a finite number of vertices have been traversed unless, it would have been routed along a cycle or unless it would have encountered a void. These two last cases are impossible because of Theorem 5.1 and Theorem 5.2. When $\kappa(P_v, d) = 0$ holds for a vertex $v \in V$ and a destination d, this means that $\forall i \in \{0, 1, \ldots, k-1\} : \delta_i(v, d) = 0$. Because of property 1 in Definition 5.1 which defines a metric space, $v = d$. Therefore the destination has been reached at vertex v. □

Because of the previous theorems, the LBFR scheme is always able to route a packet to its destination without encountering cycles or voids. This is important because routing on a network such as the Internet backbone has to be guaranteed for all source-destination pairs.

Example To illustrate the LBFR system an example is given for a multi-embedding consisting of three embeddings, thus $k = 3$, which is depicted in Figure 5.3.[10] This is the equivalent of saying that there is a single embedding into $\mathbb{T}^3$. Thus every vertex has three sets of coordinates, one for each embedding. Assume a source node s and a destination node d for which $\delta(s, d) = (3, 5, 7)$. This means that the distance δ is 3 in embedding $\mathcal{T}_0$, 5 in embedding $\mathcal{T}_1$ and 7 in embedding $\mathcal{T}_2$. When routing naively the following scenario can occur:

- $\delta(n_1, d) = (3, 4, 7)$, decide to route from s to n_1 in embedding $\mathcal{T}_1$
- $\delta(n_2, d) = (3, 6, 6)$, decide to route from n_1 to n_2 in embedding $\mathcal{T}_2$
- $\delta(n_3, d) = (3, 5, 7)$, decide to route from n_2 to n_3 in embedding $\mathcal{T}_1$
- $\delta(n_1, d) = (3, 4, 7)$, decide to route from n_3 to n_1 in embedding $\mathcal{T}_1$

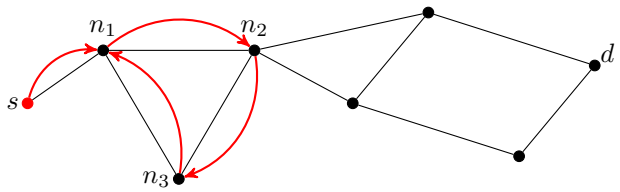


Figure 5.3: Naive routing with multiple embeddings

Here a cycle has been introduced because although the packet is routed greedily in each embedding individually, this does not hold for their aggregation. When routing intelligently with the κ-function restriction the following happens (see Figure 5.4): $\delta^*(\langle s \rangle, d) = \delta(s, d) = (3, 5, 7)$, thus $\kappa(\langle s \rangle, d) = 3 + 5 + 7 = 15$.

- $\delta(n_1, d) = (3, 4, 7)$, s checks n_1 to route on $\mathcal{T}_1$, thus

  $$\delta^*(\langle s, n_1 \rangle, d) = (\min\{3, 3\}, \min\{5, 4\}, \min\{7, 7\}) = (3, 4, 7)$$

  which means that $\kappa(\langle s, n_1 \rangle, d) = 3 + 4 + 7 = 14$. Because $\kappa(\langle s, n_1 \rangle, d) < \kappa(\langle s \rangle, d)$ the next hop n_1 is accepted.

- $\delta(n_2, d) = (3, 6, 6)$, n_1 checks n_2 to route on $\mathcal{T}_2$, thus

  $$\delta^*(\langle s, n_1, n_2 \rangle, d) = (\min\{3, 3\}, \min\{4, 6\}, \min\{7, 6\}) = (3, 4, 6)$$

  which means that $\kappa(\langle s, n_1, n_2 \rangle, d) = 3 + 4 + 6 = 13$. Because $\kappa(\langle s, n_1, n_2 \rangle, d) < \kappa(\langle s, n_1 \rangle, d)$ the next hop n_2 is accepted.

[10] Note that the network is simply for illustration purposes: the depicted network might not allow an embedding as it is presented here.

- $\delta(n_3, d) = (3, 5, 7)$, n_2 checks n_3 to route on $\mathcal{T}_1$, thus

$$\delta^*(\langle s, n_1, n_2, n_3\rangle, d) = (\min\{3,3\}, \min\{4,5\}, \min\{6,7\}) = (3,4,6)$$

  which means that $\kappa(\langle s, n_1, n_2, n_3\rangle, d) = 3 + 4 + 6 = 13$. Because $\kappa(\langle s, n_1, n_2, n_3\rangle, d) = \kappa(\langle s, n_1, n_2\rangle, d)$ the next node n_3 is rejected as a potential next hop.

- a node n_3' with $\delta(n_3', d) = (2, 5, 7)$ would be accepted by n_2 because

$$\delta^*(\langle s, n_1, n_2, n_3'\rangle, d) = (\min\{3,2\}, \min\{4,5\}, \min\{6,7\}) = (2,4,6)$$

  which means that $\kappa(\langle s, n_1, n_2, n_3'\rangle, d) = 2 + 4 + 6 = 12$ which is lower than $\kappa(\langle s, n_1, n_2\rangle, d)$.

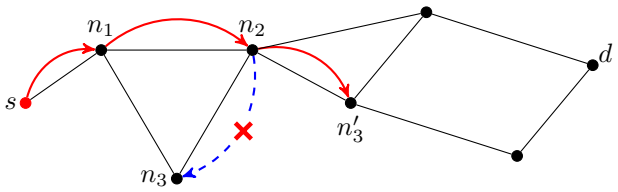


Figure 5.4: LBFR avoids the introduction of routing cycles

5.3.1.3 Hybrid Forest Routing (HFR)

As the results indicate (see Section 6.3) the stretch may be very large when striving for active link load balancing. In terms of stretch and load balancing, GFR and LBFR are two opposites: GFR attains low stretch but has no load balancing while LBFR achieves high load balancing but the stretch becomes unbearably large. To combine the best of both worlds a new mechanism has been developed called Hybrid Forest Routing (HFR). It makes a trade-off between increased stretch and improved load balancing by employing a cost function that combines current link load information with the distance to the destination. This cost function $C : V^3 \to \mathbb{R}$ is defined as

$$C(u, n, d) = \min_{0 \le i < k} \left\{ \gamma \cdot \hat{L}(u, n)^\alpha + (1 - \gamma) \cdot \delta_i(n, d)^\alpha \right\} \tag{5.11}$$

This equation selects the minimal cost over all embeddings $\mathcal{T}_i$. These individual cost function consist of a load balancing term $\hat{L}(u, n)$ defined by Eq. (5.17). These terms can be tuned by the factor γ and the exponent α. The second term describes the distance in embedding $\mathcal{T}_i$. It can be tuned by the factor $(1 - \gamma)$ and the exponent α. The factor γ is introduced to shift between load balancing focus and distance focus while α is introduced to obtain non-linear behaviour of the cost terms. This means that difference between the load and the distance results in a higher cost. For example, say that $\alpha = 1, \gamma = 0.5$ and $\hat{L}(u, x) = 2$ with $\delta_i(x, d) = 2$. This leads to a cost 2. For another node y the following could be true: $\hat{L}(u, y) = 3$ with $\delta_i(y, d) = 1$, which also leads to a cost 2. However if we want to avoid that an imperfect load balancing is compensated by a lower distance (or the opposite) the following parameter could be used: $\alpha = 2, \gamma = 0.5$. Now node x would result in a cost 4 while node y would result in a cost 5, which means that x is now preferred.

In Eq. (5.16) the right-hand side can be rewritten as follows

$$\min_{0\leq i<k}\left\{\gamma\cdot\hat{L}(u,n)^{\alpha}+(1-\gamma)\cdot\delta_i(n,d)^{\alpha}\right\} \tag{5.12}$$

$$=\gamma\cdot\hat{L}(u,n)^{\alpha}+\min_{0\leq i<k}\{(1-\gamma)\cdot\delta_i(n,d)^{\alpha}\} \tag{5.13}$$

$$=\gamma\cdot\hat{L}(u,n)^{\alpha}+(1-\gamma)\cdot\min_{0\leq i<k}\{\delta_i(n,d)\}^{\alpha} \tag{5.14}$$

$$=\gamma\cdot\hat{L}(u,n)^{\alpha}+(1-\gamma)\cdot\epsilon(n,d)^{\alpha} \tag{5.15}$$

which leads to

$$C(u,n,d)=\gamma\cdot\hat{L}(u,n)^{\alpha}+(1-\gamma)\cdot\epsilon(n,d)^{\alpha} \tag{5.16}$$

for $n\in N(u)$ and ϵ defined by Eq. (5.6). The function $\hat{L}(u,v)$ represents the normalized traffic load of the edge between u and v.[11] This is the traffic load of the link (u,v) divided by the average load of the node's incident links $I(u)$. This normalized load is defined by

$$\hat{L}(u,n)=\frac{d_G(u)\cdot L(u,n)}{\sum_{v\in N(u)}L(u,v)} \tag{5.17}$$

The factor $\gamma\in[0,1]$ is a weight factor to scale between greedy and load balanced routing. The power $\alpha\in]0,+\infty[$ allows for non-linear tuning. As can be seen, HFR also uses the relaxed-metric space $(\mathbb{T}^k,\epsilon)$. However, because the cost function is an extension of the ϵ-function, it does not employ greedy routing. It is not even necessarily distance-decreasing in ϵ. To guarantee packet delivery, the κ-function from LBFR is used to steer packets towards their destination. Hence, it is called a hybrid mechanism. It is able to attain strong load balancing while keeping the stretch down, which is shown in Section 6.3. GFR and LBFR can now be seen as two special instances of HFR on the opposite side of the spectrum. When $\gamma=1$ the LBFR mechanism is recreated. When $\gamma=0$ the HFR reverts to GFR. The HFR forwarding procedure can be created when replacing $L(u,w)$ in Algorithm 5.1 on lines 18, 20 and 22, with $C(v,w,d)$ as defined in Eq. (5.16). The effect of using different parameter values for γ and α on the stretch and the load balancing behaviour of HFR is analysed in Section 6.3 of the results chapter.

5.3.2 Extension to an m-hop neighbourhood

A way to generate better paths in terms of load balancing and stretch, is by allowing nodes to use more network state information. This can be done by interchanging the default 1-hop neighbourhood with an m-hop neighbourhood. Now not only information about a node's incident edges $I(u)$ is used, but also the information about the links incident to its neighbours $I\left(N(u)\right)$, and recursively to their neighbours. The m-hop neighbourhood of a node u is denoted as $N_m(u)$. A larger neighbourhood may improve routing quality in terms of stretch or load balancing, however this comes at a cost. A simple example of what the effect can be on load balancing by using an m-hop neighbourhood can be witnessed in Figure 5.5.

The trade-off made here is having more potentially useful information in favour of a higher state

[11] The representation of the load may be arbitrarily chosen but should be consistent for all network links.

requirement in every node, and a higher computational complexity of the forwarding decision making. A forwarding node u will now not only check the state of the links $I(u)$ but all possible paths within $N_1(u), N_2(u), \ldots, N_m(u)$. For all these possible paths the average normalized link load value will be calculated and used in Eq. (5.16). This is combined with the distance to the destination of the final node along these investigated paths, increased with the path length. As such m-hop paths of different size can be compared.

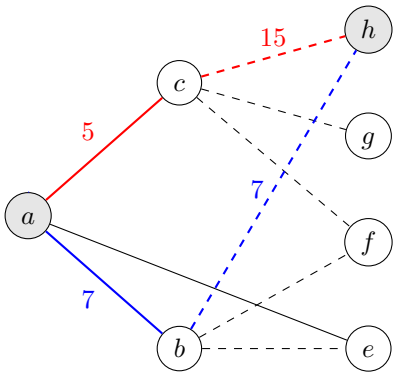
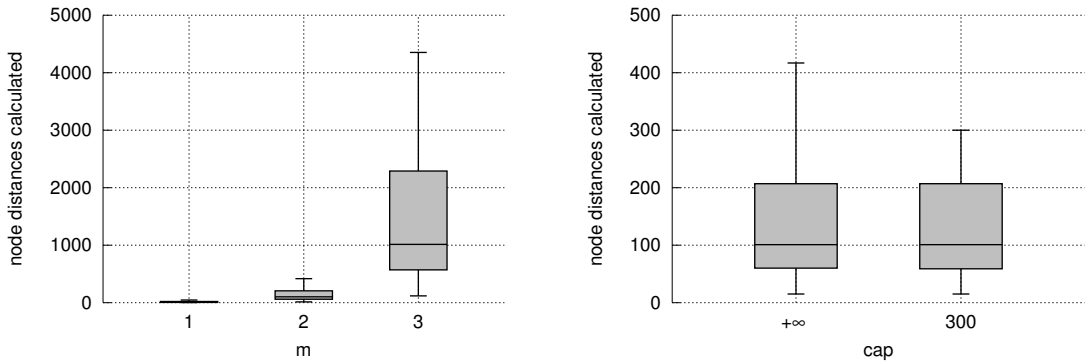


Figure 5.5: Path calculation possibilities for m-hop neighbourhoods, in this case $m = 2$. By conveying a larger neighbourhood, better paths become available. When optimizing load-balancing, a will now send the packet to b instead of c to arrive at h because the average path load for $\langle a, c, h\rangle$ is 10 while for $\langle a, b, h\rangle$ this is only 7.

When a path P has been selected as being the most cost-effective path within the neighbourhood $N_m(u)$, the current node u will simply forward the packet to the first node of P, which is a direct neighbour of u. This receiving node will then again calculate an m-hop neighbourhood, which ensures that a node always uses live load information. Using Figure 5.5 this means that a will calculate an m-hop neighbourhood in which it concludes that $\langle b, h\rangle$ is the most cost-effective path. Then it will forward the packet to b, which will again calculate an m-hop neighbourhood. Another possible implementation would be to encode the forwarding path P in the packet header. By doing this only one router decides which path to follow, every m hops. Using Figure 5.5 this means that a will calculate the path $P = \langle b, h\rangle$ which will be encoded in the packet header. When this packet is received at b, it will simply forward it to h without any cost calculations. However, because this path is constructed at node u, by the time the packet reaches the last node of P, the traffic load distribution might have altered severely. The implementation in this thesis is based on the former method (calculating the m-hop neighbourhood at every node).

If the average degree of $G = (V, E)$ is $d_G(V)$, then using an m-hop neighbourhood requires a state complexity of $O\left(d_G(V)^m\right)$, meaning that it scales exponentially with the neighbourhood size. When m is increased until it matches the diameter of the network, the paths generated will be shortest path when using a greedy routing mechanism. This also means that the state requirement will be comparable to the state requirement of routing tables. Nodes have to communicate the necessary information by sending status updates periodically. So not only storage and computational resources are sacrificed, but also link bandwidth.

A problem with using a fixed m-hop neighbourhood is that high-degree nodes might get overloaded with information. This can happen especially in scale-free networks where a small number of nodes have a very high degree compared to the rest. Simply decreasing the factor m would punish the large number of low-degree nodes and is therefore no option. This is why a cap on the number of elements inspected in the m-hop neighbourhood has been introduced. The nodes are investigated (their cost function is calculated) in order of increasing m and increasing degree to cover as many different paths as possible. Thus first nodes in $N_1(u)$ are investigated, then in $N_2(u)$ and so on in a lowest-degree first order. It has to be noted all elements in $N_1(u)$ will be investigated, even if $|N_1(u)|$ is larger than the cap. Therefore the cap only limits nodes in a neighbourhood $N_m(u)$ with $m > 1$.



(a) Boxplot of neighbourhood size for varying m-values on SF8k.

(b) Boxplot of neighbourhood size for $m = 2$ and two different cap values on SF500.

Figure 5.6: Effect of an m-hop neighbourhood with and without cap on the state and computational complexity.

In Figure 5.6(a) the number of cost function calculations is set out in function of m. It can be seen that the number of potential neighbours increases very fast in function of the neighbourhood size m due to the exponential complexity $O\left(d_G(V)^m\right)$. In Figure 5.6(b) the effect of capping the number nodes is shown. In this figure an arbitrary cap of 300 is set simply to show the capping effect. Say that a node has 500 neighbours within its m hop neighbourhood, then it will now only calculate the cost function (or distance) for 300 of them. The figure indicates that this only affects a minority of nodes in scale-free graphs because the first, second (median) and third quartile of the boxplot remain the same. This means that the number of cost functions calculated by a node is only limited by the cap for nodes with a high degree. This results in a good trade-off where high-degree nodes do not get overloaded with information while low-degree nodes can experience the information gain of using a whole m-hop neighbourhood. The effect of using different m-values in combination with different caps on the stretch and the load balancing behaviour is analysed in Section 6.3.3 of the results chapter.

5.3.3 Recapitulation

The use of multiple embeddings has been researched which forms the foundation of the Hybrid Forest Routing (HFR) scheme. This routing scheme has two other variants called Greedy Forest

Routing (GFR) and Load Balanced Forest Routing (LBFR) which are special cases of HFR. The routing schemes establish passive routing behaviour through the use of multiple embeddings. GFR focuses on attaining a low stretch but has no form of active load balancing while LBFR focuses on active load balancing but neglects the stretch. HFR is a combination of these two mechanisms and seeks a trade-off between load balancing and stretch. Furthermore the use of larger neighbourhood sizes has been investigated, which can be applied to all three routing mechanisms. Using a larger neighbourhood size should be favourable in terms of load balancing and stretch, but it also increases the node state requirement, the computational complexity of the forwarding decision making and the control traffic bandwidth usage. At this point the graph embedding procedure itself is still unexplored, therefore different coordinate assignment procedures will be investigated in the next section.

5.4 Graph embedding procedure

The FR systems make use of a graph embedding into $\mathbb{T}^k$. How this embedding is constructed will be explained in this section. This embedding is based on multiple spanning-tree based embeddings into $\mathbb{T}$. Therefore first k different spanning trees T_i are built. While this tree construction process is ongoing, coordinates are assigned to each network node. The embedding procedures in this work consist of two phases:

1. root nodes of the k different spanning trees T_i are elected.
2. the root nodes will initiate the tree forming procedure. While every tree T_i is formed, coordinates are also assigned. This process of forming a tree and generating coordinates will therefore be denoted as the graph embedding procedure.

The next section explains the root election procedure. Afterwards the graph embedding procedure itself is explained.

5.4.1 Root node election

The procedure for acquiring the k greedy graph embeddings $\mathcal{T}_i$ starts with the construction of k different spanning trees T_i. This construction begins with the election of a root vertex $r^{(i)}$ for each tree T_i. As the foundation of this election procedure is based on Section 4.1.1, the same notations will be employed. In this work multiple root election variants were used:

- *highest-key node election* (HKE): $\mathcal{A}^{(0)} = \mathcal{K}(\mathcal{E})$. As node keys are arbitrarily assigned, this will be simulated in practice as a random root election procedure.
- *highest-degree node election* (HDE): $\mathcal{A}^{(0)} = d_G(V)$. Electing the highest degree node as root may lead to a tree that corresponds to the hierarchical structure of a scale-free network.
- *anchor node election* (AE): the SALT anchor nodes are elected to act as root nodes as defined in Section 4.1.2.

These settings are used in the election procedure described in Section 4.1.1. To obtain k different root nodes, the election procedure is run k times in parallel. The effect of using these different election variants will be investigated in Section 6.3.4 of the results chapter.

5.4.2 Graph embedding procedure

After having elected a set of k root nodes $r^{(i)}$, the spanning trees T_i can be constructed along with the corresponding greedy graph embeddings $\mathcal{T}_i$. In what follows the construction of a single embedding $\mathcal{T}$ will be explained, multiple embeddings can be seen as a parallel extension. The embedding procedure itself is exactly the same as explained in Section 5.2.1. The difference lies within the way the trees are formed. First, the root node r is assigned the coordinate (0). Next it will send its coordinates to its neighbours $N_1(r)$.[12] Furthermore each recipient is assigned a unique number by the sender. The receiving nodes will then compute their own coordinates and send an assignment message to their neighbours $N_2(r)$. In turn these will recursively send it to all other nodes $N_j(r)$ (if they accept the coordinates). When a node which already has coordinates assigned to it receives a new coordinate assignment message, it may react in different fashions. These fashions are called different tree growing modes:

- *first mode* (FM): discard the message in all cases.
- *breadth-first mode* (BFM): override the previous coordinate tuple only if the newly received one leads to a lower coordinate tuple length. This leads to a tree of minimal depth.
- *redundant mode* (RM): previous coordinates may be overridden only when their associated cost function value is lower. This cost function is based on the coordinate tuple lengths and the overlap of the spanning trees.
- *breadth-first redundant mode* (BFRM): a hybrid mode combining the breadth-first mode (BFM) with the redundant mode (RM).

Algorithm 5.2: Tree growing: first mode (FM)

input : vertex u has sent its coordinates to its neighbours; current vertex $v \in N(u)$ is listening for packet receipt
output: vertex v has coordinates assigned according to embedding $\mathcal{T}$ in $\mathbb{T}$

```
1  receive incoming coordinate packet of node u with coordinates u and a child number c_u
2  look up own coordinates v corresponding to embedding T
3  if v = ∅ then
4  |  c_u ← child number c_u stored in packet
5  |  calculate own coordinates v ← u⌢{c_u}
6  |  foreach n ∈ N(v) do
7  |  |  send(packet containing v and neighbour number c_v) to n
8  |  end
9  else
10 |  drop packet
11 end
```

Embedding the graph (shown in Algorithm 5.2) in first mode (FM) is a baseline method to compare the other assignment procedures with. It is very easy to construct and its termination can easily be proven. Furthermore it is naturally cycle-free: no cycle avoidance mechanism is required. All nodes get assigned a parent in the tree (except the root) which is never altered. Once all nodes

[12] Remind that $N_i(u)$ represents the i-th hop neighbours of u.

have been assigned coordinates by their parent, the embedding procedure terminates. The major downside of this method is that the tree may be very unbalanced. As a result, very long paths between the root and leaf nodes are possible, leading to long coordinates.

Algorithm 5.3: Tree growing: breadth-first mode (BFM)

input : vertex u has sent its coordinates to its neighbours; current vertex $v \in N(u)$ is listening for packet receipt
output: vertex v has coordinates assigned according to embedding $\mathcal{T}$ in $\mathbb{T}$

```
1  receive incoming packet of node u with coordinates u, a child number c_u and key K(u)
2  look up own coordinates v and parent key K(p) corresponding to embedding T
3  if v = ∅ ∨ K(u) = K(p) ∨ (K(u) ≠ K(p) ∧ (|v| < |u| − 1)) then
4  |  c_u ← child number c_u stored in packet
5  |  calculate own coordinates v ← u⌢{c_u}
6  |  update own parent p ← u
7  |  foreach n ∈ N(v) do
8  |  |  send(packet containing v and neighbour number c_v) to n
9  |  end
10 else
11 |  drop packet
12 end
```

The breadth-first mode (BFM) is described in Algorithm 5.3. As can be seen on line 3, coordinates are only overridden when the coordinate tuple resulting from the received set has a lower length than the current tuple, or its parent node has acquired a new set of coordinates. In the latter case, the child node should be updated accordingly. By doing so, a tree of minimal depth is built rooted at r. An advantage of using BFM is that the introduction of cycles is impossible. It also has a fixed upper embedding procedure time. The downside is that the tree redundancy is rather high. With tree redundancy the overlap of different spanning trees of the same network is meant. For example when two trees $T_i = (V, E')$ and $T_j = (V, E'')$ have a maximal redundancy, they completely overlap, which means that $E' = E''$. Because of this, there is no advantage in using them both. However in case their sets of links have a low overlap (low tree redundancy), the set $E' \cap E''$ is small and as such more routing paths are available.

The high tree redundancy of the two first modes has two disadvantageous effects:

1. low fault-tolerance: if a link fails which happens to be the link between parent and child for a large fraction of the trees, this will likely cause routing voids.

2. little shortcuts are taken: it is beneficial to let shortcut in a embedding $\mathcal{T}_i$ be the parent-child link in embedding $\mathcal{T}_j$. The reason for this is that a packet is generally routed to its parent when no shortcut is available (or when its destination is an ancestor). This contributes to the root hotspot effect which we want to avoid.

To bypass these issues, a third mode has been developed called redundant mode (RM). Now emphasis is on minimal tree redundancy, attaining maximal tree spreading. This is done by making sure that every link is an edge in approximately the same number of trees. Attaining this

requires a metric to measure the amount of tree redundancy of the spanning trees inducing the different embeddings. Every edge $e \in E$ is part of a number of sets E'_i for a number of different trees $T_i = (V, E'_i)$. If minimal redundancy is desired, the goal is to make this number more or less equal for every edge in E of $G = (V, E)$. Therefore the metric τ is defined by the following definition.

Definition 5.2. *Assume k spanning trees. Denote the number of different sets E'_i, corresponding to $T_i = (V, E'_i)$ with $0 \leq i < k$, that an edge $e \in E$ in a graph $G = (V, E)$, belongs to as n. The amount of tree redundancy, termed the τ-ratio, is defined as the standard deviation $\sigma(n)$ divided by the average $\bar{n}$ over all $e \in E$. This can be formulated as $\tau = \dfrac{\sigma(n)}{\bar{n}}$.*

A high τ-value indicates that the trees are spread evenly over the network while a low τ-value indicates that there is a huge difference in the number of spanning trees a link of the network is part of. Also note that $\tau \geq 0$. To decrease the tree redundancy while constructing the embedding, a cost function $f_i : V^3 \to \mathbb{R}$ was defined for each embedding $\mathcal{T}_i$ in the system:

$$f_i(u, p, p') = \eta a_i(u) + \beta b_i(u) + \chi c_i(u, p, p') \tag{5.18}$$

with $\eta, \beta, \chi \in \mathbb{R}$ adjustable parameters with default values $\eta = 1$, $\beta = \frac{1}{2}$ and $\chi = \frac{3}{2}$. a_i, b_i and c_i are cost terms defined as

$$a_i(u) = |u'| - |u| \tag{5.19}$$

$$b_i(u) = \begin{cases} 0 & \text{if } a_i(u) \leq 0 \\ |u'| - |u^*| & \text{if } a_i(u) > 0 \end{cases} \tag{5.20}$$

$$\begin{aligned} c_i(u, p, p') = {} & |\Psi(u, p') - \bar{\Psi}(u) + 1| + |\Psi(u, p) - \bar{\Psi}(u) - 1| \\ & - \big(|\Psi(u, p') - \bar{\Psi}(u)| + |\Psi(u, p) - \bar{\Psi}(u)|\big) \end{aligned} \tag{5.21}$$

with $u \in V$ also representing the current coordinates for embedding $\mathcal{T}_i$; u' the new coordinates for embedding $\mathcal{T}_i$; u^* the coordinates of the lowest length encountered so far for embedding $\mathcal{T}_i$; $\Psi(x, y)$ represents the number of trees T_i that are making use of the edge $(x, y) \in E$; $\bar{\Psi}(x)$ is the average of $\Psi(x, y)$ $\forall y \in N(x)$; p is the current parent of u and p' the potentially new parent.
Thus $a_i(u)$ indicates the decrease in length of the new coordinates compared to the current coordinates of u. $b_i(u)$ represents the decrease in length of the coordinates under consideration for node u when comparing them to the coordinates of the lowest length that have been assigned (and possible overridden) so far for u. $c_i(u, p, p')$ describes how much accepting the new parent will equalise the number of trees each link $e \in I(u)$ is part of. In this cost term, $|\Psi(u, p') - \bar{\Psi}(u) + 1|$ describes the offset of the number of trees link (u, p') is part of versus the average of this value, denoted as $\bar{\Psi}(u)$, over all links in $I(u)$, when p' is chosen as the new parent of u. To this is added $|\Psi(u, p) - \bar{\Psi}(u) - 1|$ which describes the offset of number of trees link (u, p) is part of when p is no longer a parent of u, versus the average over the links $I(u)$, denoted as $\bar{\Psi}(u)$. The sum of these two offsets is compared to the sum of the offsets without the replacement of the parent, which is $\big(|\Psi(u, p') - \bar{\Psi}(u)| + |\Psi(u, p) - \bar{\Psi}(u)|\big)$. Therefore c_i describes whether changing the parent will reduce the variance in the total number of trees each links of $I(u)$ is part of. Also note that $c_i(u, p, p')$ is bounded by $-2 \leq c_i(u, p, p') \leq 2$.

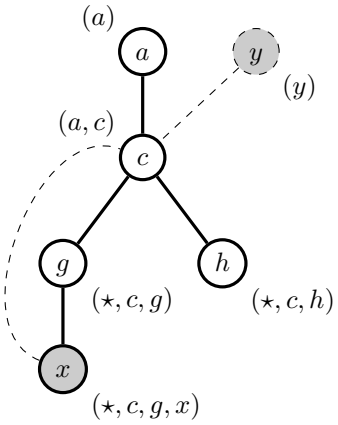
Algorithm 5.4: Tree growing: redundant mode (RM)

input : vertex u has sent its coordinates to its neighbours; current vertex $v \in N(u)$ is listening for packet receipt
output: vertex v has coordinates assigned according to embedding $\mathcal{T}$ in $\mathbb{T}$

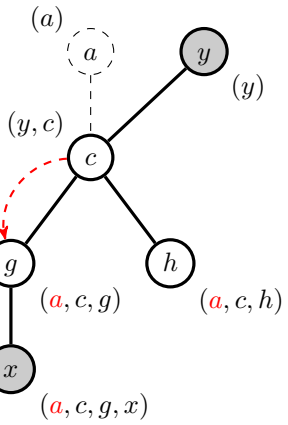
```
1  receive incoming packet packet of node u along with its coordinates u, a child number c_u
   and key K(u)
2  look up own coordinates v and parent key K(p) corresponding to embedding T
3  if v = ∅ then
4  |  go to line 11
5  else if K(u) = K(p) ∨ (K(u) ≠ K(p) ∧ f'_i(v, p, u) < 0) then
6  |  if K(v) ∈ P ∧ K(u) ≠ K(p) then
7  |  |  cycle avoided, drop packet
8  |  else if K(v) ∈ P ∧ K(u) = K(p) then
9  |  |  cycle detected because parent update message received, resolve cycle
10 |  else
11 |  |  c_u ← child number c_u stored in packet
12 |  |  calculate own coordinates v ← u⌢{c_u}
13 |  |  update own parent p ← u
14 |  |  record own key in packet in sequence of traversed nodes: P ← P⌢K(v)
15 |  |  foreach n ∈ N(v) do
16 |  |  |  send(packet containing v and neighbour number c_v) to n
17 |  |  end
18 |  end
19 else
20 |  drop packet
21 end
```

The graph embedding procedure in redundant mode (RM) is described in Algorithm 5.4. When growing a tree in RM, precautions have to be taken in order to avoid the introduction of cycles, especially when using low α-values in Eq. (5.18) as nodes will then frequently switch parents. The fact that now longer coordinates may override shorter ones is the main reason for the potential introduction of cycles. This happens when an ancestor node accepts new coordinates from one of its descendants. As a result greedy forwarding can no longer be guaranteed as the embedding $\mathcal{T}_i$ may no longer be greedy. A solution at first glance is to check whether a receiving node is an ancestor of the sending node by checking if $|\phi(u', u)| \neq |u|$ holds. But even then it is still possible to generate cycles as explained in Figure 5.7. To counter this, every coordinate assignment packet also holds information about the current path P up to the root. The key of every vertex $v \in P$ is recorded. If a node u receives a coordinate assignment message, it will first check whether it is part of P by examining if its own key is present in this set of recorded keys. Only when $u \notin P$ will it consider accepting the new coordinates. This mechanism is called cycle avoidance.

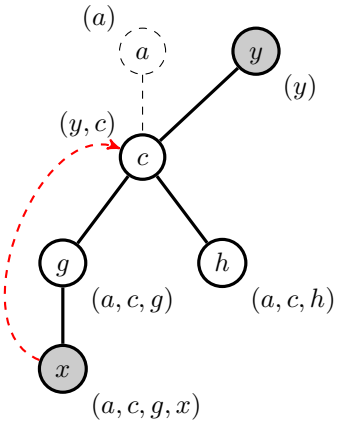
However, even with the cycle avoidance active, it is still possible to introduce cycles in certain cases that are hard to avoid due to the system's distributed nature. Therefore, a cycle resolution procedure has been implemented. This can be seen in Algorithm 5.4 at line 8. When a node receives a coordinate assignment packet from its parent, it should normally accept these new coordinates. However, when this packet already has its own key in the P field, a cycle was introduced. In node u the P field will look like $(\ldots, u, a, b, \ldots, c)$. Therefore it knows that the cycle is the path



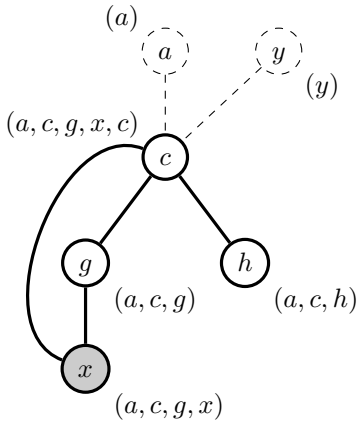
(a) The tree (a, c, g, h, x) exists after a has assigned coordinates to c. This leads to updating g, h and x. Hereafter, the parent a of c is replaced by y because of a decreasing cost.



(b) Node c updates its coordinates according to its new parent y. The tree now becomes (y, c, g, h, x). At the same time the update message of the old coordinates of c reaches x.



(c) Node x sends the updates to all of its neighbours, which includes c. It does not notice that c is an ancestor because the coordinates of c have already been updated according to its new parent y.



(d) Node c accepts the new coordinates of x because of a decreasing cost function. However, this was still based on the non-updated coordinates and thus a cycle is formed, corrupting the tree.

Figure 5.7: Cycle introduction by faulty coordinate updates which results in an endless loop of updates.

$\langle u, a, b, \ldots, c, u \rangle$. It will simply send on the coordinate assignment packet to its children in order to notify those children which are part of the cycle. Every time a cycle is detected, the node will search a neighbour which is not part of the cycle and make it its parent. This is done by querying a new parent p and asking for its coordinates along with a new child number c_p. After the packet has traversed all nodes of the cycle, these will all have had the opportunity to switch parent, hence the cycle is resolved.

The advantage of generating an embedding in RM is that the tree redundancy τ is very low. However it is very hard to give an estimated upper bound on the embedding procedure time due

to the complex interactions of the cycle resolution and avoidance mechanism. Also the coordinates will be larger than when building a tree of minimal depth. Therefore a hybrid mechanism was established by combining the BFM with the RM, called breadth-first redundant mode (BFRM). Now a tree of minimal depth is combined with a cost function to minimize τ. This basically comes down to assigning parameter values $\eta > 2$ and $\beta = 0$ in Eq. (5.18). This is then applied to Algorithm 5.3 by altering the check ($|\mathrm{u}| < |\mathrm{v}| - 1$) at line 3 to ($f_i'(v, p, u) < 0$) based on Eq. (5.18). Because of the breadth-first behaviour, no cycle avoidance or resolution is required, which greatly diminishes the complexity of the embedding procedure. This is a result of the fact that a node can never increase its coordinate tuple length. What the exact trade-off of the BFRM is can be seen in the results chapter.

5.4.3 Recapitulation

At this point three variants of the Forest Routing (FR) system have been proposed, which incorporate load balancing by using multiple embeddings, cost functions or larger neighbourhood sizes while focusing on a low stretch. In the previous sections the graph embedding procedure itself was investigated. Four schemes which can be combined with three root election mechanisms were proposed. How the tree redundancy τ resulting from the different embedding procedures affect the stretch, the load balancing behaviour and the fault-tolerance will be shown in Section 6.3.4 of the results chapter. Next, network dynamics will be investigated. FR will be altered to be able to cope with non-uniform link bandwidth, a changing network topology and failure scenarios.

5.5 Network dynamics

5.5.1 Non-uniform link capacities

The load balancing mechanisms explained in the previous sections try to spread traffic evenly over all network links by allowing each node to balance traffic on its outgoing links. An underlying assumption made here is that every link has an equal traffic capacity, however in real networks this is not necessarily the case. To avoid this limitation, a new load function $\hat{L}(u, v)$ is introduced to replace the normalized average load in Eq. (5.16). This new function is defined as

$$\hat{L}(u,v) = f(u,v) + \begin{cases} 0 & : f(u,v) < \omega \\ (1 - f(u,v))^{-1} & : f(u,v) \geq \omega \end{cases} \tag{5.22}$$

$$f(u,v) = \frac{L(u,v)}{L^+(u,v)} \tag{5.23}$$

with $L^+(u, v)$ the capacity of the link (u, v). The parameter $\omega \in [0, 1]$ determines the traffic cut-off point. When the load-versus-capacity fraction f crosses ω, $\hat{L}(u, v)$ is augmented with a non-linear term to prevent link congestion. Eq. (5.22) thus tries to attain an equilibrium such that every link is evenly filled. For example when $\omega = 0.9$, the system attempts to keep the link load beneath 90% to avoid congestion. Because different link capacity values can be handled, the congestion-aware load balancing (CALB) mechanism can be viewed as a generalization of the default load balancing (LB) mechanism. How using CALB affects the behaviour of FR will be investigated in Section 6.3.5

of the results chapter.

5.5.2 Fault-tolerance

In this section the fault-tolerance of the FR mechanism is investigated, which is its capability of handling node or link failures. This is of utmost important as the Internet will always be subject to failures due to its vast size. Link failures are defined as any a link state in which the link is no longer capable of transmitting data between its endpoints. A node failure is defined as the inability of a node to process packets correctly. Starting by looking at failures from a theoretical point of view, first geometric routing with a single embedding ($k = 1$) is studied. Herein the network is modelled by a graph $G = (V, E)$ with an embedding $\mathcal{T}$ into $\mathbb{T}$, induced by a spanning tree $T = (V, E')$ with $E' \subseteq E$. Within this model, two types of link failures can be distinguished:

1. A link $e \in E$ fails for which $e \notin E'$ holds, therefore T does not become disconnected.
2. A link $e \in E$ fails for which $e \in E'$ holds, as such T becomes disconnected.

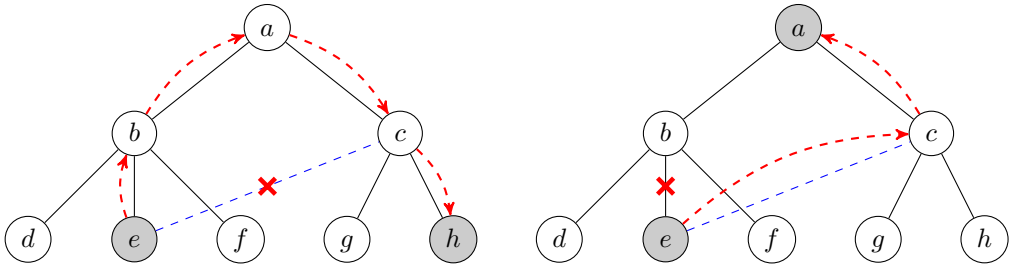
The first type of link failures does not critically affect routing in a negative way. Packet delivery is still guaranteed because e is a shortcut link (see Figure 5.8(a)), although the stretch and the load balancing behaviour may deteriorate. In contrast, the second kind of failure is critical. Now packet delivery can be obstructed for certain source-destination pairs. This is due to the underlying embedding potentially losing its greedy property (see Figure 5.8(b)). On the other hand, node failures can also be divided into two classes:

1. A node $v \in V$ fails which is a leaf node of T, thus T does not become disconnected.
2. A node $v \in V$ fails which is not a leaf node of T, therefore T becomes disconnected.

In case a leaf node fails, no harm is done. Of course this prevents the node from becoming the source or destination of any path, but this is a trivial case. The failed node will never be a part of the sole path between two other nodes. Non-leaf nodes can however be critical transit nodes for the network traffic. In case no shortcut exists between two disjoint sub-trees, all traffic has to be routed along their common ancestor node a. Therefore when a fails, all traffic between the two trees will encounter a void at a neighbour of a. However, due to the likelihood of the existence of shortcuts, negative effects of such node failures are diminished, but no guarantees can be made.

By expanding to an embedding into the k-dimensional space $\mathbb{T}^k$ with $k > 1$, routing redundancy is introduced. This makes routing more fault-tolerant, as will be experimentally verified in the results chapter. A link or node failure must now disconnect all of the embedding-inducing trees $T_i = (V, E'_i)$ to obstruct a geometric routing algorithm. Link and node failures maybe once more be of different types:

1. A link $e \notin \bigcap_{0 \leq i < k} E'_i$ fails, thus only a fraction m of all trees T_i become disconnect.
2. A link $e \in \bigcap_{0 \leq i < k} E'_i$ fails, thus all k trees T_i become disconnected.
3. A node $v \notin \bigcap_{0 \leq i < k} V^{(i)}$ fails with $V^{(i)}$ the set of non-leaf nodes of tree T_i. Only a fraction m of the trees T_i becomes disconnected.



(a) A shortcut link has failed, this is non-critical: traffic that was previously routed along $\langle e, c, h\rangle$ is rerouted along $\langle e, b, a, c, h\rangle$. Note that a path $\langle b, e, c, h\rangle$ will never be taken due to way the distance δ is defined.

(b) A link in the tree itself has failed: traffic routed along $\langle e, b, a\rangle$ is rerouted over $\langle e, c, a\rangle$. Though the link is part of the tree and the failure is therefore critical, traffic can be rerouted over a shortcut link. If such a shortcut would not exist, e would be a routing void.

Figure 5.8: Failure scenarios for an embedding into $\mathbb{T}$

4. A node $v \in \bigcap_{0 \leq i < k} V^{(i)}$ fails. Now all k trees T_i become disconnected.

In the first and the third type, geometric routing is still theoretically possible due to the existence of a distance-decreasing path between every source-destination pair in the remaining $(k - m)$ embeddings $\mathcal{T}_i$. In case two and four, routing once more cannot be guaranteed. Therefore a backup mechanism is required, such as the *gravity-pressure* mechanism mentioned in Section 2.5. The FR mechanism has been equipped with a system that employs the principles used in Cvetkovski & Crovella (2009). Contrary to their gravity-pressure technique, the FR backup mechanism is based on the metric space $(\mathbb{T}^k, \epsilon)$.

When routing without any sort of backup routing mechanism, a packet traverses a path P based on a cost function that is computed at every node $u \in P$ as Eq. (5.16) dictates. This function is only calculated for the set $S(u)$, the set of viable neighbouring nodes as explained in Section 5.3.1.2. When $S(u) = \varnothing$, based on Theorem 5.3 a link or node failures must have happened. In this case, the backup routing mechanism becomes active. Now the whole neighbourhood $N(u)$ is investigated instead of just a subset $S(u)$. However, when this is done headlong, the routing paths may become cycles[13], thus prudence is prevalent. In case the backup mechanism is active, the keys of the visited nodes, along with the number of times they were visited, are saved in the header of the data packet being routed. To avoid cycles, each node will inspect these visitation numbers. A packet will be forwarded to a node with the lowest visitation number out of all considered nodes. When multiple nodes have an equal visitation number, the node with the lowest distance will be selected. Cvetkovski & Crovella (2009) have proven that this mechanism is able to attain 100% routing success rate even in severe failure scenarios. The algorithm is described in Algorithm 5.5. When a packet encounters a void (and the backup system is not active), the current ϵ-distance is saved in the packet header after which the packet enters the *backup mode*, until a lower ϵ-distance is encountered. This has much in common with the greedy- and face-mode employed by Karp & Kung (2000) in which also a backup mechanism is used to escape voids. The use of the backup

[13] Cvetkovski & Crovella (2009) and Section 5.3.1.2

routing mechanism, the effect of using different graph embedding procedures and FR in general will be tested on fault-tolerance in Section 6.3.6 of the results chapter.

Algorithm 5.5: Backup routing in case of failures

input : current node u knows its neighbours $N(u) \in V$ and the current load of its currently incident edges $I(u) \in E$; a k-dimensional embedding $\mathcal{T}^k$ is assumed
output: next node v to forward the packet `p` to

receive incoming packet `p` that has to be forwarded
calculate $S(u)$ as in the default HFR mechanism
if $S(u) = \varnothing$ **then**
  if `p.backup` = `false` **then**
    `p.map` $\leftarrow \varnothing$
    `p.backup` $\leftarrow$ `true`
  end
  if `p.map` contains entry for $\mathcal{K}(u)$ **then**
    $h \leftarrow$ `p.map.value`$(\mathcal{K}(u))$
    `p.map.value`$(\mathcal{K}(u)) \leftarrow h + 1$
  else
    `p.map.value`$(\mathcal{K}(u)) \leftarrow 1$
  end
  $v \leftarrow$ selectNext() (see Algorithm 5.6)
else
  $v \leftarrow$ apply default HFR selection procedure
end

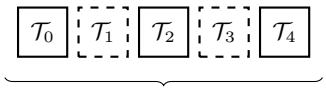
Algorithm 5.6: Explanation of selectNext() in Algorithm 5.5

input : input of Algorithm 5.5; map `p.map` containing visitation number of nodes along the path that p traversed; a destination node d
output: next node v to forward the packet to

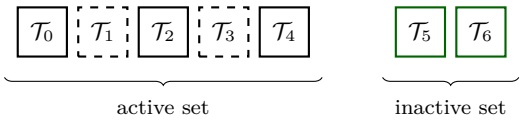
$\mathtt{h}_{\mathtt{min}}$, $\epsilon_{min} \leftarrow +\infty$
$R(u) \leftarrow \varnothing$
foreach $m \in N(u)$ **do**
  $\mathtt{h}_m \leftarrow 0$
  if `p.map` contains entry for $\mathcal{K}(m)$ **then**
    $\mathtt{h}_m \leftarrow$ `p.map.value`$(\mathcal{K}(m))$
  else
    $\mathtt{h}_m \leftarrow 0$
  end
  if $\mathtt{h}_m < \mathtt{h}_{\mathtt{min}} \vee (\mathtt{h}_m = \mathtt{h}_{\mathtt{min}} \wedge \epsilon(m,d) < \epsilon_{min})$ **then**
    $R(u) \leftarrow \varnothing$
    $\epsilon_{min} \leftarrow \epsilon(m,d)$
    $\mathtt{h}_{\mathtt{min}} \leftarrow \mathtt{h}_m$
    $R(u) \leftarrow R(u) \cup \{m\}$
  else if $\mathtt{h}_m = \mathtt{h}_{\mathtt{min}} \wedge \epsilon(m,d) = \epsilon_{min}$ **then**
    $R(u) \leftarrow R(u) \cup \{m\}$
  end
end
$v \leftarrow$ random element from $R(u)$

5.5.3 Changing topology

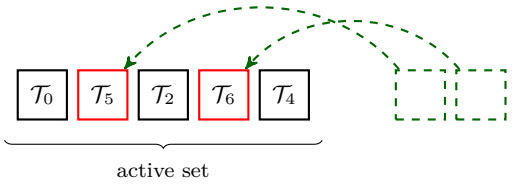
The underlying network may have a dynamic topology in which links are added or removed in function of the time. This results in the spanning trees, inducing the graph embeddings of the FR system, losing their connectedness because they do not adapt to the dynamic topology. Consequentially, routing performance deteriorates as time goes on which can be witnessed by an increased stretch. When facing a severely altered network, even routing voids may be introduced. Therefore a solution may be to create new embeddings while traffic is being routed through the network. These new embeddings may then replace older ones. The solution proposed here enables nodes to initiate a voting process, or a voting process is initiated with a certain frequency, in which nodes decide whether or not a new spanning tree —and thus a new embedding— should be created. When a voting is successful, the graph embedding procedure as explained in Section 5.4 is initiated. An important property should be that as the network changes over time, the chances of a passing vote should increase. A specific implementation of such a voting scheme is left as future work.



(a) time t_0: embeddings $\mathcal{T}_1$ and $\mathcal{T}_3$ are deteriorated, vote has passed and the construction of new embeddings is started.



(b) time $t_0 + \Delta_1$: two new embeddings $\mathcal{T}_5$ and $\mathcal{T}_6$ are created. Nodes start communicating these sets with their neighbourhood.



(c) time $t_0+\Delta_1+\Delta_2$: all nodes have informed their neighbourhood about their new coordinate sets. Each node replaces $\mathcal{T}_1$ and $\mathcal{T}_3$ with $\mathcal{T}_5$ and $\mathcal{T}_6$.

Figure 5.9: The regeneration of two embeddings in a set of five embeddings, thus $k = 5$ and $n = 2$.

When using a k-dimensional tree space $\mathbb{T}^k$, induced by k spanning trees $T_0, T_1, \ldots, T_{k-1}$ a new tree T_k is generated accompanied by new coordinates. When the embedding generation procedure based on T_k has started at a point in time t_0, all nodes will wait a certain time Δ_1 before they conclude that the embedding procedure has ended. At this point $t_0 + \Delta_1$ there exist $(k + 1)$ embeddings. This is generalizable such that multiple spanning trees may be created in parallel. Then there might exist $(k + n)$ embeddings at some point, from which there are k active and n non-active ones. This procedure is illustrated in in Figure 5.9. At the point $t_0 + \Delta_1$ all nodes will

communicate their new sets of coordinates with their neighbourhood (Figure 5.9(b)).

After some time Δ_2 has passed, all nodes should know the n new coordinate sets of their neighbours. Then after the point $(t_0 + \Delta_1 + \Delta_2)$ in time, all nodes will discard n of the old embeddings from the original working set and exchange them with the newly created ones (Figure 5.9(c)). Note that every node has to exchange the same set of embeddings. A potential issue could be the requirement to frequently exchange embeddings in a scenario where the network is highly dynamic. This is due to the fact that node coordinates also represent host identifiers, therefore the Domain Name System (DNS) should be constantly updated as well. Also hosts should then be constantly informed about the new coordinate sets. More specifically, the DNS and hosts should be aware of the new coordinate sets within the time period $(\Delta_1 + \Delta_2)$ after the start of the generation of the new embeddings, as hereafter the new coordinate sets will be used for routing purposes. The effects of using an embedding regeneration scheme on the stretch and the success ratio of FR will be investigated in Section 6.3.7 of the results chapter.

5.5.4 Recapitulation

In the previous sections, dynamics in the underlying network were examined. First the used formulas of the FR system were adapted to be able to cope with non-uniform link capacities. Hereafter, the effect of multiple graph embeddings on the fault-tolerance of geometric routing was theoretically investigated. From this could be concluded the use of multiple embeddings should result in passive fault-tolerance, which will be substantiated in the results chapter. As passive fault-tolerance might not be enough, an existing backup mechanism was tuned to use multiple embeddings such that 100% routing success rate is guaranteed, even in severe cases of link or node failures. Lastly the issue of a dynamically changing network topology was tackled by introducing a mechanism that is able to hot-swap older deteriorated embeddings with newer ones that model the current network topology more accurately. In the next sections a time and computational complexity analysis will be given for the FR system.

5.6 Complexity analysis

In this section the time and computational complexity of the designed algorithms will be investigated theoretically. The analyses presume a synchronous communication model such that two neighbouring nodes have a clearly defined upper bound for their one-way communication delay, denoted as ξ^+. The lower bound ξ^- may always be trivially set to zero. A bounded communication delay is necessary to detect node failures. When no upper bound is assumed, the time it takes for a message to travel between two nodes might be infinite. Heartbeat failure detection systems would therefore be futile. Furthermore it would be impossible to guarantee any upper time bound for any of the algorithms. We also assume the existence of a maximum processing time, denoted as μ^+. This represents the time required for a node to receive a message, process it and put it on one of its outgoing lines or simply drop it. Of course the minimal processing time, denoted as μ^-, is again trivially zero.

5.6.1 Coordinate size

According to the FR graph embedding procedures each child of a vertex $v \in V$ is labelled with a number in $\{0, \ldots, (d_G(v) - 1)\}$ with $d_G(v)$ the degree of v. The root node enjoys a special treatment and gets the coordinate (0) assigned. Assuming an underlying spanning tree $T = (V, E')$ of $G = (V, E)$, then $d_G(v)$ is at most $(|V|-1)$. Therefore the binary representation has a complexity $O(\log |V|)$.[14] A balanced k-regular tree with $k > 2$ has a depth complexity of $O(\log |V|)$ and the coordinate lengths can be at most equal to depth of the tree. From this follows that every coordinate tuple can be represented with at most $O(\log^2 |V|)$ bits. This poly-logarithmic complexity in $|V|$ means that the coordinate representations are succinct.[15] Furthermore, scale-free networks have a diameter $d \sim \log(\log |V|)$ which means that in the worst-scenario (when the root node has a distance to another node equal to the diameter d), using a minimal-depth tree building algorithm such as Algorithm 5.3, the tree depth is at most equal to the network diameter.[16] Thus the complexity of the tuples becomes $O(\log(\log |V|) \times \log |V|)$. However when taking into the most general scenario, a network can have a diameter that is nearly equal to the number of vertices in the network, namely $(|V| - 1)$. In this worst-case scenario (worst-case network diameter) the complexity becomes $O(|V| \times \log |V|)$.

5.6.2 Root election time

To be able to estimate the graph embedding procedure time, the root election time has to be determined. It is assumed that every node u knows its own key $\mathcal{K}(u)$ and the keys $\mathcal{K}(N(u))$ of its neighbours. This information about neighbouring nodes is stored in a neighbour table. The upper bound on the root election time (based on the election procedure explained in Section 5.4.1) is

$$t_{root} \leq 2d\left(\xi^+ + \mu^+\right) \tag{5.24}$$

with d the network diameter. This can be shown as follows. For a graph $G = (V, E)$, assume an arbitrary node $u \in V$ that starts the election procedure, and an arbitrary node $r \in V$, which will be eventually elected as root node. Assume the shortest path distance between u and r is $\delta_\Pi(u, r) = d$, which is the maximally possible shortest path distance. After u has initiated the process, it sends a message to each of its neighbours $N(u)$. Each neighbour receiving such a message will send it on to its neighbours. This process goes on recursively until all nodes have been reached. Therefore at least one message will travel along the shortest path $\Pi(u, r)$. This packet will take a time $t = d\left(\xi^+ + \mu^+\right)$ to reach r, as u itself does not need to process the message. Upon receiving the message, r will start sending its key $\mathcal{K}(r)$ to its neighbours as part of the election procedure. Because r will eventually be elected, as assumed, all neighbours will send this message further to their neighbours. One of these backwards travelling messages will flow along the shortest path $\Pi(r, u)$, because of the reasons previously mentioned. This also takes a time t. Therefore t_{root} is maximally $2t = 2d\left(\xi^+ + \mu^+\right)$.

After a time $t \geq t_{root} + \Delta$, with Δ an arbitrary time offset, has passed since the initiation of the election procedure, the nodes assume the election process has converged. Next, the root node

[14] Korman *et al.* (2002)
[15] As defined by Eppstein & Goodrich (2008)
[16] Cohen & Havlin (2003)

r will start the tree growing procedure. This can be naturally extended to multiple root nodes that are elected in parallel. It has to be noted that not all root nodes $r^{(i)}$ have to be elected yet before growing the trees. The growing mechanism for an embedding $\mathcal{T}_i$ may start while the root node election procedure for embedding $\mathcal{T}_j$ is still in progress. There is no interdependence between different graph embedding procedures and as such they may run in parallel.

5.6.3 Tree growing time

For the analysis of the tree growing time complexity, it is assumed the election of root nodes has already happened. When using a first mode (FM) style assignment, no node will alter the coordinates it was first assigned. In the breadth-first mode (BFM) style, nodes will let their coordinates be overridden by coordinates of lower length. In either case the upper bound of the coordinate assignment time will be the same. It can be said that the assignment time is equal to

$$t_{assign} = d\,(t_{proc} + t_{prop}) \tag{5.25}$$

with d the network diameter, t_{proc} the node processing time required to process the newly received coordinates and t_{prop} the transmission delay. Thus it can be said that

$$t_{assign} \leq d\left(\mu^{+} + \xi^{+}\right) \tag{5.26}$$

This can be shown by the fact the root will start spreading the tree coordinates to its neighbours, which on their turn will spread this information. In the worst case scenario, the root node has a shortest path that equals d to another node. As coordinates may only be overwritten when the node's new position has a lower depth, the coordinate assignment packet reaching a node will always have travelled less or equal than d hops. This means that the unchronological arrival of the messages is due to variation of the node processing time or link propagation time. The path in the tree between a node and the root node is always a shortest path. The value of d can be correlated to the network size by $d \sim \log(\log|V|)$ for scale-free networks.[17] Thus for scale-free networks, the initialization time is of the order of $O(\log(\log|V|))$. Of course when dealing with the worst-case scenario, for a general network, $d = |V| - 1$. Therefore the worst-case complexity becomes $O(|V|)$. This analysis holds true for the breadth-first redundant mode (BFRM) in which also a tree of minimal depth is created. The complexity analysis of RM is left as future work due to its inherently complex interactions between the cycle avoidance and resolution mechanisms.

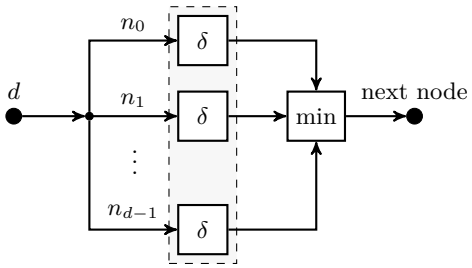
5.6.4 Router processing time

As the number of neighbours can be quite large in scale-free networks for some nodes, the number of calculations needed for forwarding decision making can be equally large. Next, the router processing time for packet forwarding using the GFR mechanism will be examined. For each neighbouring node, the distance in k embeddings has to be calculated to generate the ϵ-distance. Hereafter the best distance has to be selected, e.g. by a tournament selection algorithm. When computing this in a sequential way this will take a processing time equal to

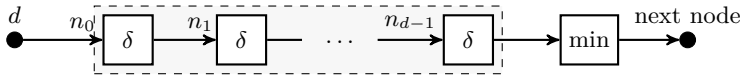
[17] Proven by Cohen & Havlin (2003)

$$t_{proc}(u) = k \cdot d_G(u) \cdot t_{dist} + \log_2(k \cdot d_G(u)) \cdot t_{comp} \quad \forall u \in V \tag{5.27}$$

for every packet. In this equation t_{comp} represents the time required to compare two distances and select the lowest one; t_{dist} stands for the time required to calculate a single distance between two points in the tree coordinate space; $d_G(u)$ represents the degree of the node and k is the number of embeddings used in the routing system. An advantage is, however, that the routing system is naturally parallelizable. The system can be broken down in two pieces: the distance calculation and the computation to select the next node. The distance calculation can be performed in parallel as the calculations of the different distances are not interdependent. The comparison computation has to be done after the distance calculations. This processing scheme is depicted in Figure 5.10.



(a) parallel packet processing



(b) sequential packet processing

Figure 5.10: Schematic overview of router architecture: routers can parallelize distance calculations towards a destination node d for the different neighbouring nodes $n_0, n_1, \ldots, n_{d-1}$. The comparison operation is the only sequential part.

When computing the distances in parallel, the total time becomes

$$t_{proc}(u) = t_{dist} + \log_2(k \cdot d_G(u)) \cdot t_{comp} \quad \forall u \in V \tag{5.28}$$

Herein t_{dist} is not scaling with the number of neighbours or number of embeddings at all, which leads to a constant complexity. The term with t_{comp} does scale however, but comparing two values and extracting the minimum is not a computationally intensive task. For the HFR calculations Figure 5.11 depicts the serial and parallel steps. The symbols used in the figure are based on Eq. (5.5), Eq. (5.6) and Eq (5.16). Although many calculations are necessary, it can be seen that they are very parallelizable: all neighbouring node cost values can be calculated independently. Note that in this figure we assume $\alpha = 1$ in Eq. (5.16) and disregard the random selection mechanism required when different nodes have an equal cost value. Also it is assumed that a 1-hop neighbourhood is utilized, for the sake of clarity.

For any neighbour, ϵ can be calculated in parallel by calculating every δ-value independently for each embedding. Assuming a node u has $d_G(u)$ neighbours and the embedding was k-dimensional,

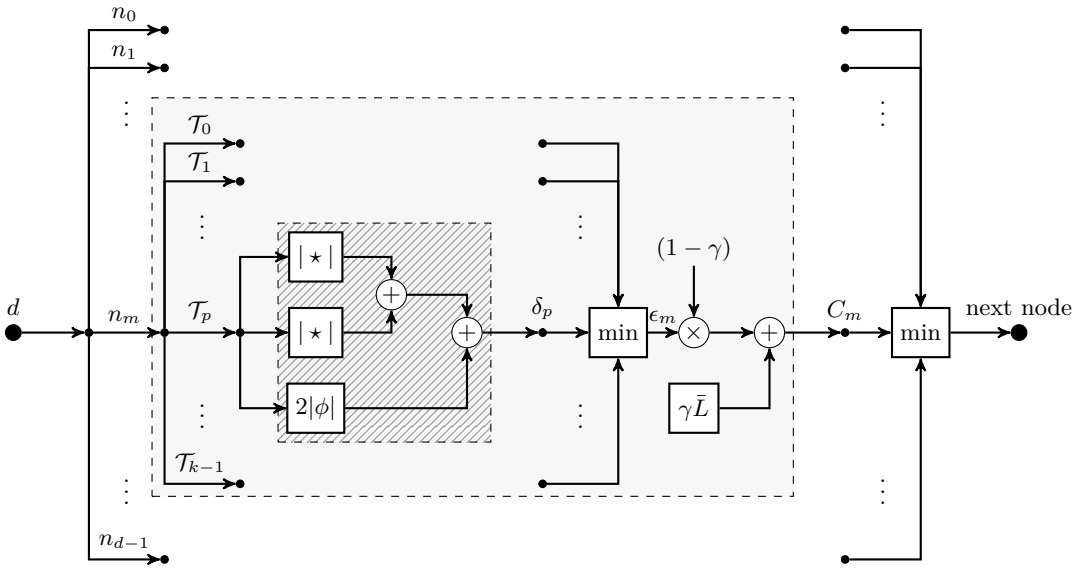


Figure 5.11: Overview of sequential and parallel processing steps for the HFR forwarding decision (κ-calculation not included). The shaded box represents the δ-function based on Eq. (5.5), which is calculated in parallel for every embedding $\mathcal{T}_i$ while the dashed gray box represents the cost function C based on Eq. (5.16) which is calculated for every neighbour n_i with a destination node d.

then δ has to be calculated $d_G(u) \cdot k$ times in parallel. The output δ_p value is then fed to a minimum selection mechanism, which is in turn fed to the cost function based on Eq. (5.16). In this step the link traffic load information is added. Finally once more a minimum selection mechanism is used to extract the node with the best cost value. This node will become the next node along the routing path to the destination. What is not depicted in this figure is the calculation of the κ-function for clarity reasons. This can easily be implemented by only allowing neighbours to enter the second minimum selection mechanism when one of their δ_i values is lower than the received δ_i^* values (see Section 5.3.1.2). By doing so, the actual calculation of κ is bypassed using information that has already been computed. The cost of the load balancing behaviour itself is only minimal as it only requires the read-out of variable holding the current traffic information of an outgoing link, which has to be added to the distance calculation. This load balancing component is depicted as the block $\gamma\bar{L}$ in Figure 5.11.

5.6.5 Recapitulation

After having explained the main algorithm called Forest Routing (FR) along with its variants, its time and computational complexity were analysed theoretically. The forwarding decision making process, which looks very complex at first sight, can be mitigated by exploiting the system's high parallelisability. Coordinate lengths can be kept low by employing minimum-depth tree growing algorithms (FM, BFM, BFRM). On top of that some estimation about the graph embedding procedure times are given. This can be helpful when determining whether a frequent re-embedding scheme is feasible or not as presented in Section 5.5.3. In the next chapter the FR system will be tested and its results will be discussed.

5.7 Conclusion

In this chapter a theoretical base regarding tree-based geometric routing was set up. This served as a foundation to construct a family of geometric routing algorithms called Forest Routing (FR) based on the use of multiple greedy graph embeddings. When focusing on low stretch without any form of load balancing, Greedy Forest Routing (GFR) can be used. Contrary to GFR, Load Balanced Forest Routing (LBFR) focuses entirely on achieving link load balancing but pays no attention to the stretch. These two routing schemes are combined into one scheme called Hybrid Forest Routing (HFR) in which a factor γ can be tuned to focus on a specific combination of stretch and load balancing behaviour. The correctness of the FR family has been proven and it can be seen as a advancement of the state-of-the-art of geometric routing as load balancing was still an open question in scientific literature. To construct the greedy embeddings for FR, four different graph embedding schemes were developed which can be used in combination with three different root election mechanisms. These graph embedding procedures all focus to a greater or lesser extent on achieving low tree redundancy and generating coordinates of low length. Furthermore different network dynamics scenarios were investigated, such as non-uniform link capacities, fault-tolerance or a dynamically changing network topology. Also a theoretical complexity analysis was given regarding the coordinate lengths, the forwarding decision making and the graph embedding procedure time. In the next chapter, these different aspects of FR will be experimentally investigated for a large set of parameter values.

Chapter 6

Results and discussion

This chapter presents experiments regarding the behaviour of the mechanisms described in Chapter 5. The different routing mechanisms are tested in the same order as they are presented in the previous chapter. Every experiment investigates relevant parameter values for various algorithms and outputs the effect on the stretch and load balancing metrics. For each experiment, the set-up is written down, followed by the results. Hereafter the results are explained and discussed. Many experiments are based on the generation of a number random source-destination pairs between which traffic is simulated, which is executed multiple times. To support the fact that this number is sufficient, most plots also show the standard deviation over these multiple runs. This standard deviation shows how much different runs (each based on a number of random source-destination pairs) vary in their output value. When this variance is sufficiently low compared to the average value, this indicates that the number of source-destination pairs is sufficient to obtain solid metric values. Also the number of test iterations for the CAIDA graph experiments is lower than the number of iterations of the experiments of other graphs. This has been done deliberately due to large size of the graph. Executing the experiments in the same fashion for the CAIDA graph as for the other graphs was too computationally expensive.

To simulate routing behaviour a synthetic simulation framework was built. The goal is not to approximate realistic traffic scenarios, but to illustrate the behaviour of the various algorithms developed (see Chapter 3). All experiments were executed on the high-performance computer (HPC), property of Ghent University.

6.1 Forest Routing: greedy variant (GFR)

6.1.1 Sensitivity analysis: tree space dimension k

The GFR mechanism focusses on attaining a low stretch by using an embedding into $\left(\mathbb{T}^k, \epsilon\right)$. Its only adjustable parameter is the dimension k of the embedding space $\mathbb{T}^k$. By increasing k, more spanning trees are created which are used to generate different embeddings into $\mathbb{T}$. Remind that $\mathbb{T}^k$ is the aggregation of these individual embeddings. As a result, nodes should have a higher chance of finding a neighbour that is connected by a shortest path to the destination of the packet being forwarded. Therefore the logical result of increasing k would be that the stretch goes down.

Experiment 1. *For each value of k between 1 and 30 with a step size of 1, for the scale-free graphs SF500, SF2k, SF8k, 10^5 random source-destination pairs were generated. Between these pairs, Pareto distributed traffic was simulated. The graph embedding was generated in redundant mode (RM) with highest-key node election (HKE) and randomly assigned keys. This trial was repeated 10 times for each k value, over which the average and standard deviation of the different metrics were calculated. For the CAIDA graph, k was drawn from $\{1, 5, 10, \ldots, 30\}$ and the trial was repeated 5 times for each k-value.*

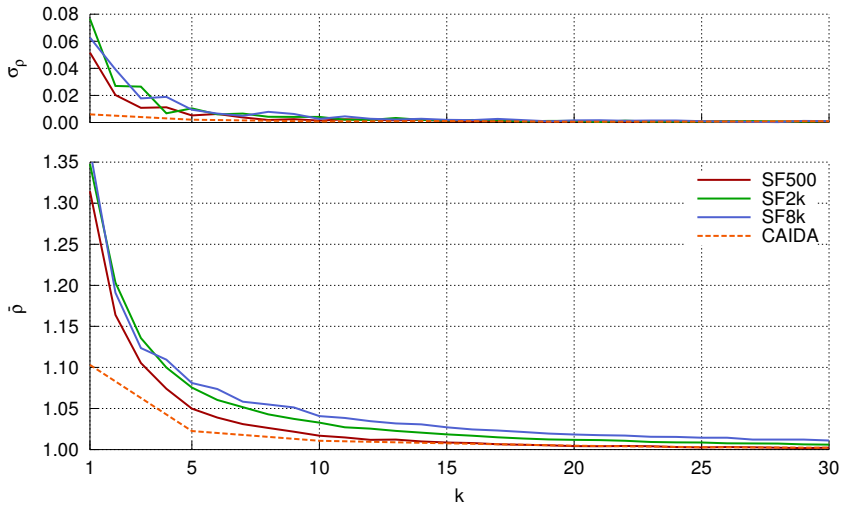
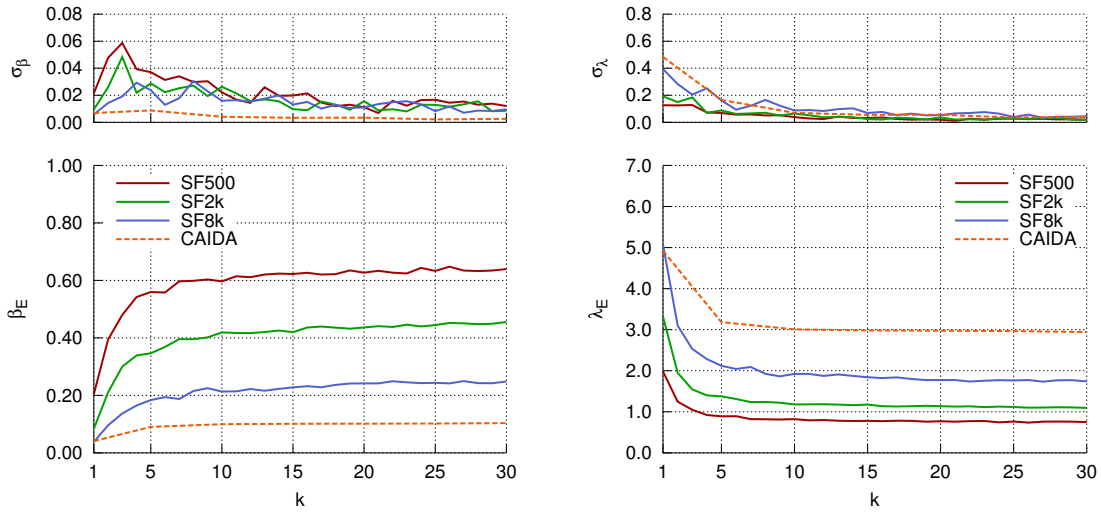


Figure 6.1: GFR: average stretch $\bar{\rho}$ and its standard deviation σ_ρ set out in function of a varying number of embeddings k, tested on different scale-free networks as well as the CAIDA graph.

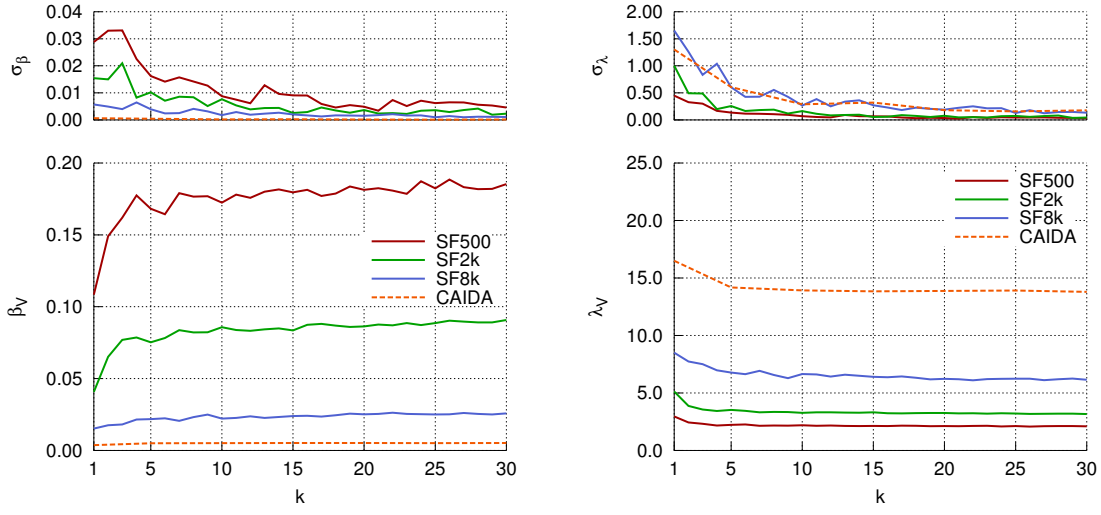
Figure 6.1 shows the average stretch $\bar{\rho}$ in function of the number of embeddings k. Also the standard deviation of $\bar{\rho}$, denoted as σ_ρ is plotted. Asymptotic behaviour $\bar{\rho} \to 1$ can be noticed as $k \to +\infty$. This can be explained by the availability of more embeddings, which allows more routing decision making freedom because ϵ is the minimum of all δ_i-distances. As a result of this higher freedom, there is an increased chance that one of the embeddings (or a combination of embeddings) will lead to a short path between two nodes, resulting in a very low $\bar{\rho}$. When the number of vertices increases, the dimension k also needs to rise to maintain the same stretch. Though k needs to increase for larger graphs, no scalability problems can be observed, which is indicated by the proximity of the different curves to each other. For the CAIDA graph, even at low k-values, low $\bar{\rho}$ values are attained, even though the graph is substantially larger than the largest scale-free graph, SF8k. A possible explanation for this low stretch is that not many paths have a lot of alternative paths within the CAIDA graph, therefore diminishing the chance that a path generated by GFR is not a shortest path.

Figure 6.2 shows the link load balancing metrics β_E and λ_E. As k increases the load balancing behaviour improves as well, which is remarkable as there is no mechanism actively steering for traffic load balancing in GFR. These plots thus highlight the passive load balancing behaviour of GFR. This can be explained by two effects of using multiple embeddings:



(a) β_E-ratio in function of k (b) λ_E-ratio in function of k

Figure 6.2: GFR: load balancing metrics for links (β_E and λ_E) set out in function of a varying number of embeddings k, tested on different scale-free networks as well as the CAIDA graph, along with their corresponding standard deviations σ_β and σ_λ.



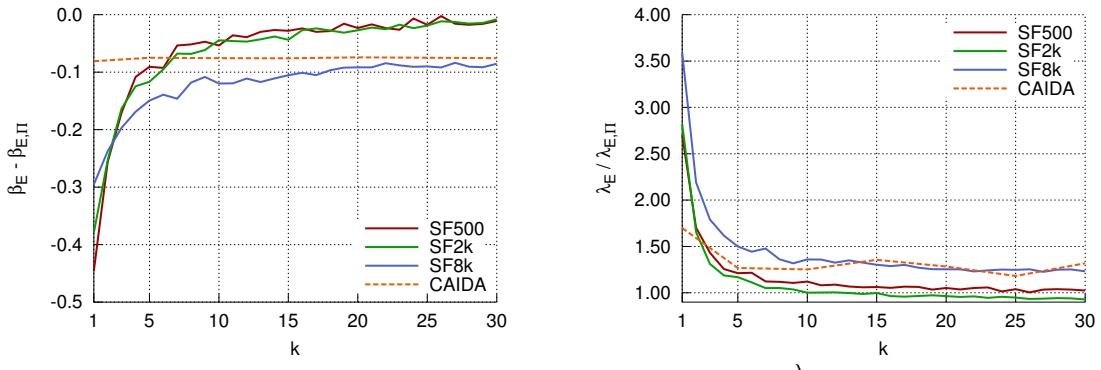
(a) β_V-ratio in function of k (b) λ_V-ratio in function of k

Figure 6.3: GFR: load balancing metrics for nodes (β_V and λ_V) set out in function of a varying number of embeddings k, tested on different scale-free networks as well as the CAIDA graph, along with their corresponding standard deviations σ_β and σ_λ.

- Because of the existence of multiple roots, the root hotspot behaviour is split among k roots, therefore improving the overall load balancing.
- Shorter paths are possible, thus decreasing the reliance on the root to act as a transit hub for traffic between different parts of the network.

For the CAIDA graph nearly no improvement in link load balancing can be noticed. This could be due to the fact that the graph possesses little alternative paths, as mentioned before. This is consistent with the CAIDA graph only having approximately a 2-to-1 links-to-vertices ratio, compared to the scale-free graphs having approximately a 3-to-1 ratio while having larger average degrees (see Appendix A). Therefore when focussing on attaining a low stretch no load balancing can be achieved without increasing the stretch. Figure 6.3 shows the node load balancing metrics β_V and λ_V. Although a similar trend can be witnessed, the difference between the highest and the lowest metric values is less distinct here. This can be explained by the fact that in a scale-free network certain high-degree nodes will act as traffic hubs. This causes traffic to be concentrated around these hubs. This problem does not occur when examining link load balancing as their high degree makes it easy to distribute traffic on their outgoing links.

When looking at the curves of the different graphs in Figure 6.2 and Figure 6.3, the positive effect of multiple embeddings on the load balancing behaviour lessens as the graph size increases. This could be due to the fact that as the graph size increases, the chance of finding an alternative next hop that has the exact same ϵ-distance decreases. As GFR always selects the next hop with the lowest ϵ-distance, these paths are never taken.

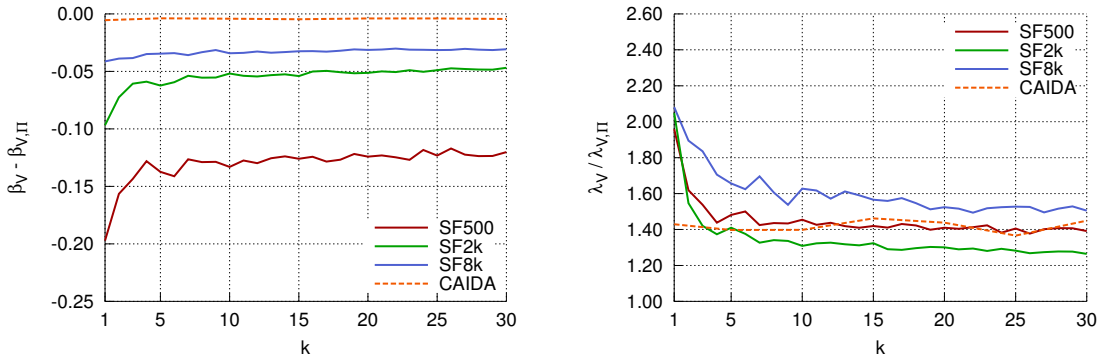


(a) $\left(\beta_E - \beta_{E,\Pi}\right)$ for varying k

(b) $\dfrac{\lambda_E}{\lambda_{E,\Pi}}$ for varying k

Figure 6.4: GFR: load balancing metrics for links, β_E and λ_E, set out in function of a various dimension k versus their shortest path counterparts, $\beta_{E,\Pi}$ and $\lambda_{E,\Pi}$, tested on different scale-free networks as well as the CAIDA graph.

Figure 6.4 and Figure 6.5 show the load balancing behaviour of GFR compared to the passive load balancing behaviour of shortest path routing. For the β-metric, the offset versus the shortest path β_Π-metric is chosen, as working with ratios gives a distorted image. For example, when increasing β from 0.1 to 0.2 this is no different than increasing β from 0.8 to 0.9, although the former has a



(a) $\left(\beta_V - \beta_{V,\Pi}\right)$ for varying k

(b) $\dfrac{\lambda_V}{\lambda_{V,\Pi}}$ for varying k

Figure 6.5: GFR: load balancing metrics for nodes, β_V and λ_V, set out in function of a various dimension k versus their shortest path counterparts, $\beta_{V,\Pi}$ and $\lambda_{V,\Pi}$, tested on different scale-free networks as well as the CAIDA graph.

far greater ratio. For the λ-metric, the ratio is used to compare GFR with shortest path routing because it has a range of $[0, +\infty[$, so this is a more natural choice. When looking at the link load balancing results it can be noticed that at low k-values the load balancing effect is a lot worse than shortest path routing. At higher k values this difference starts to disappear. The CAIDA graph has a steady offset versus shortest path routing. For links the offset or ratio does not seem to be heavily affected by the graph size. For nodes this difference with shortest path routing diminishes as the graph size increases. These results indicate that GFR scales well with increasing graph size. When the graph properties are unknown and a k-value has to be chosen, a possible solution is to test different k-values and monitor current routing performance. Based on this data, k can be adjusted accordingly. The effects of using different graph embedding procedures in GFR will be investigated in Section 6.3.4. A general conclusion is that as k goes up, GFR gains more passive load balancing behaviour and is able to attain a lower stretch.

6.2 Forest Routing: load balanced variant (LBFR)

6.2.1 Sensitivity analysis: tree space dimension *k*

In LBFR, contrary to GFR, every node attempts to balance traffic on its outgoing links. Therefore the emphasis lies completely on load balancing, while none on attaining low stretch. As the dimension k increases, the load balancing metrics should change more severely than with GFR. Furthermore, a heavy stretch increase should be observed as well.

Experiment 2. *For each different k-value between* 1 *and* 30 *with a step size of* 1*, for the scale-free graphs SF500, SF2k, SF8k,* 10^5 *random source-destination pairs were generated. Between these pairs, Pareto distributed traffic was simulated. The graph embedding was generated in redundant mode (RM) with highest-key node election (HKE) and randomly assigned node keys. This trial was repeated* 10 *times for each k-value, over which the average and the standard deviation of the different metrics were calculated. For the CAIDA graph, k was drawn from* $\{1, 5, 10, \ldots, 30\}$ *and*

the trial was repeated 5 times for each k-value.

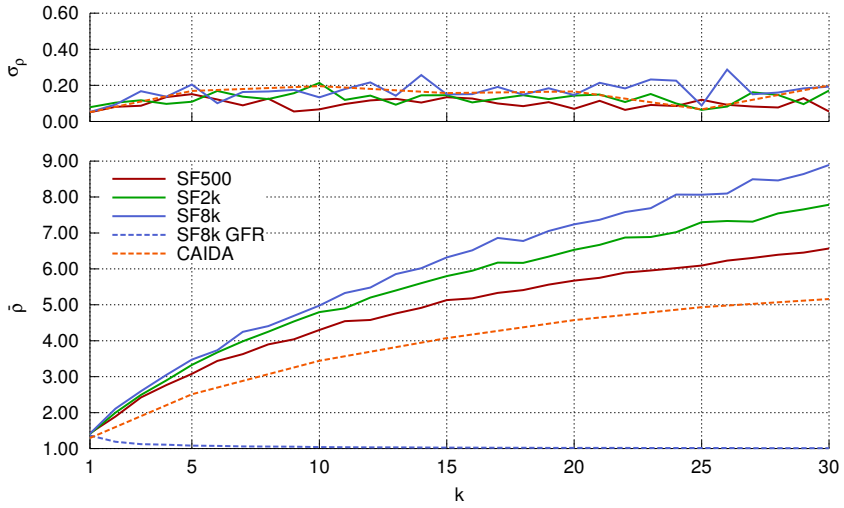
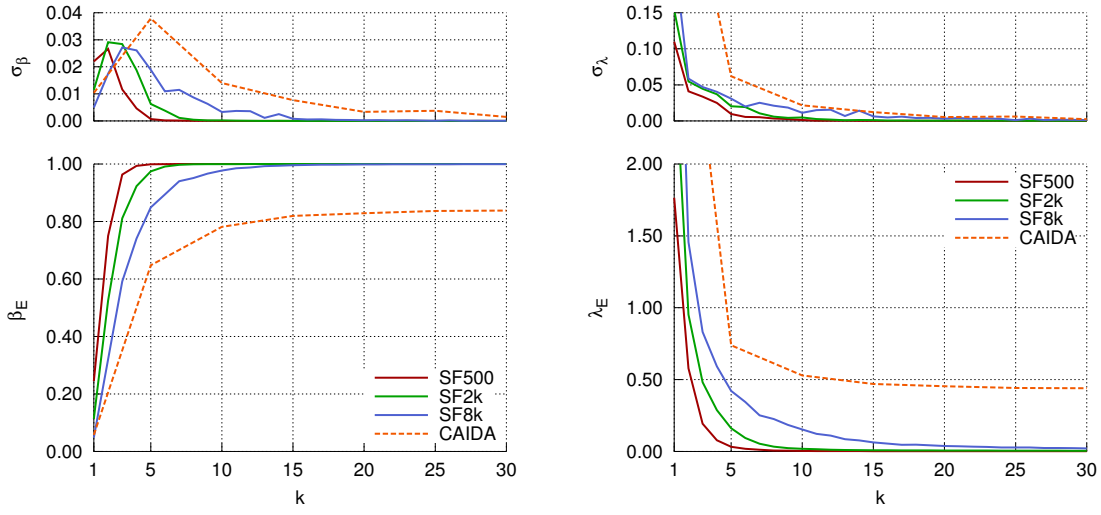


Figure 6.6: LBFR: average stretch $\bar{\rho}$ and standard deviation σ_ρ set out in function of a varying number of embeddings k, tested on different scale-free benchmark networks as well as the CAIDA graph. To compare, the average stretch $\bar{\rho}$ of GFR has been set out in function of k for the graph SF8k.

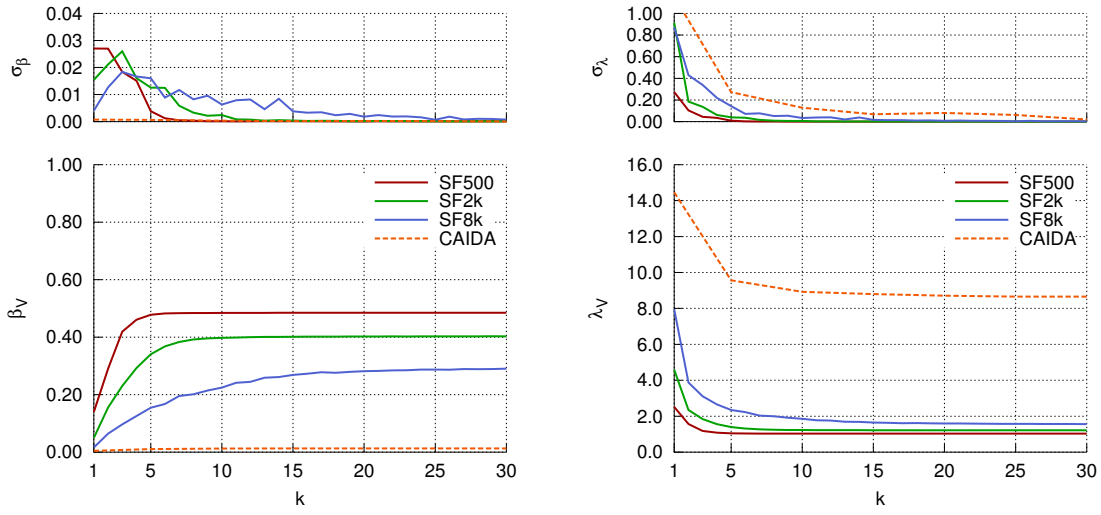


(a) β_E-ratio in function of k

(b) λ_E-ratio in function of k

Figure 6.7: LBFR: load balancing metrics for links (β_E and λ_E) set out in function of a varying number of embeddings k, tested on different scale-free benchmark networks and the CAIDA graph, along with their corresponding standard deviations σ_β and σ_λ.

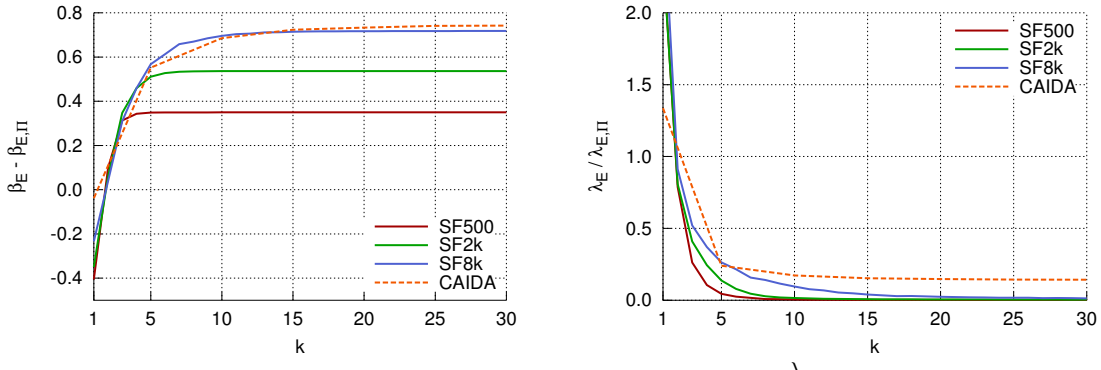
Figure 6.6 shows a steep increase in stretch as the number of embeddings k goes up. To compare, the average stretch of GFR in function of k has also been plotted. The average stretch of GFR for the other graphs was omitted for clarity reasons. Because more embeddings are available, there



(a) β_V-ratio in function of k (b) λ_V-ratio in function of k

Figure 6.8: LBFR: load balancing metrics for nodes (β_V and λ_V) set out in function of a varying number of embeddings k, tested on different scale-free networks as well as the CAIDA graph, along with their corresponding standard deviations σ_β and σ_λ.

exist more forwarding candidates that respect the κ-restriction of LBFR (the κ-function should be strictly monotonically decreasing along the path followed). Therefore, each node will have more selection freedom in choosing a next node to forward traffic to. As every node focuses entirely on balancing the load of its outgoing links, the stretch deteriorates quickly.



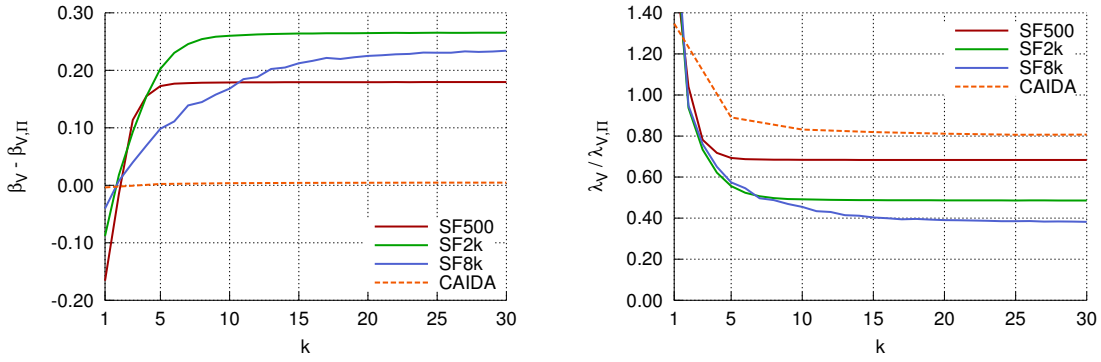
(a) $\left(\beta_E - \beta_{E,\Pi}\right)$ for varying k (b) $\dfrac{\lambda_E}{\lambda_{E,\Pi}}$ for varying k

Figure 6.9: LBFR: load balancing metrics for links, β_E and λ_E, set out in function of a various dimension k versus their shortest path counterparts, $\beta_{E,\Pi}$ and $\lambda_{E,\Pi}$, tested on different scale-free networks as well as the CAIDA graph.

In Figure 6.7 and 6.8 the load balancing metrics β and λ are shown. As k goes up, the system quickly attains maximum load balancing on its links. The node load balancing metrics converge

to a horizontal asymptote. The fact that better load balancing can be attained for links than for nodes is logical as LBFR focuses solely on link load balancing (see Algorithm 5.1). The node load balancing behaviour is simply a side-effect. Another explanation is that certain high-degree nodes in scale-free networks act as traffic hubs. Therefore traffic will naturally concentrate around these nodes, worsening the node load balancing metric values.

It can also be noticed that when the graph size increases, more embeddings are required to obtain the same load balancing performance. Also the average stretch that comes with this performance is higher. This could be due to the load balancing metrics being slightly biased regarding graphs of different sizes. Another explanation could be that it is simply more difficult to spread traffic evenly over all links in larger graphs. It could also be due to the lower achievable passive load balancing behaviour in larger graphs, which is also indicated by the lower passive load balancing performance of shortest path routing as the graph size increases (the load balancing metrics for shortest path routing can be seen in the next section in Figure 6.13 and Figure 6.14 as the horizontal dashed lines).



(a) $\left(\beta_V - \beta_{V,\Pi}\right)$ for varying k

(b) $\frac{\lambda_V}{\lambda_{V,\Pi}}$ for varying k

Figure 6.10: LBFR: load balancing metrics for nodes, β_V and λ_V, set out in function of a various dimension k versus their shortest path counterparts, $\beta_{V,\Pi}$ and $\lambda_{V,\Pi}$, tested on different scale-free networks as well as the CAIDA graph.

Figure 6.9 and Figure 6.10 show the difference between shortest path routing and LBFR in terms of load balancing. At lower k-values shortest path routing performs better regarding load balancing, but as soon as the cap of $k = 2$ is crossed the system outperforms shortest path routing. The CAIDA graph seems to follow a similar trend to SF8k for link load balancing. Node load balancing for CAIDA does not seem to improve much over shortest path routing as k increases, this is similar to what could be witnessed with GFR. As with GFR, when the graph properties are unknown and a k-value has to be chosen, a possible solution is to test different k-values and monitor current routing performance. Based on this data, k can be adjusted accordingly.

Although the load balancing metrics show that the traffic is distributed equally among all links, this does not mean that the total traffic in the network is at its lowest point. Because the stretch increases by a large factor, the total network traffic also goes up. This can be explained as follows:

when two nodes send traffic of x Gb/s between each other over a shortest path with 2 hops, this will result in $2x$ of total traffic when summing the traffic of all links these nodes use. Thus when the stretch equals 2, the number of hops becomes 4, which means that the total traffic over the used links becomes $4x$. Therefore focusing on low stretch achieves not a low average path length, but also keeps down the total traffic in the network. For this reason the stretch should receive a higher priority than the load balancing behaviour. The effects of using different graph embedding procedures in LBFR will be investigated in Section 6.3.4. A general conclusion is that as k goes up, the load balancing performance quickly converges to a horizontal asymptote which is accompanied by a heavy stretch increase.

6.3 Forest Routing: hybrid variant (HFR)

6.3.1 Sensitivity analysis: trade-off function γ-parameter

The main routing algorithm of this thesis is HFR. For this routing scheme a sensitivity analysis has been conducted for its parameter γ in Eq. (5.16). Based on the theory in Section 5.3.1.3, as $\gamma \to 0$, HFR behaves like GFR. On the other hand, as $\gamma \to 1$, HFR approximates LBFR. To test this hypothesis, the effect of shifting γ between its two extremes on the stretch and load balancing behaviour has been investigated for networks of different types and sizes.

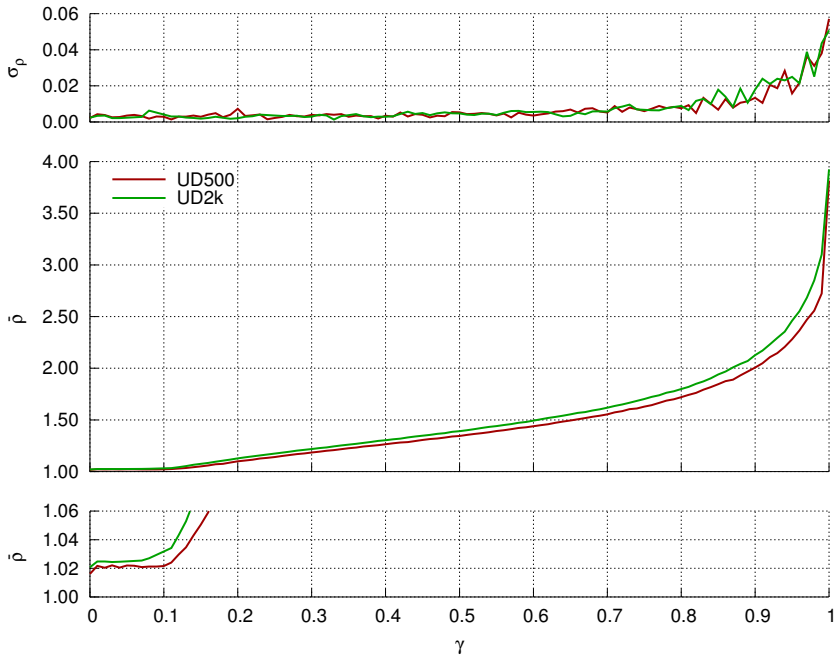
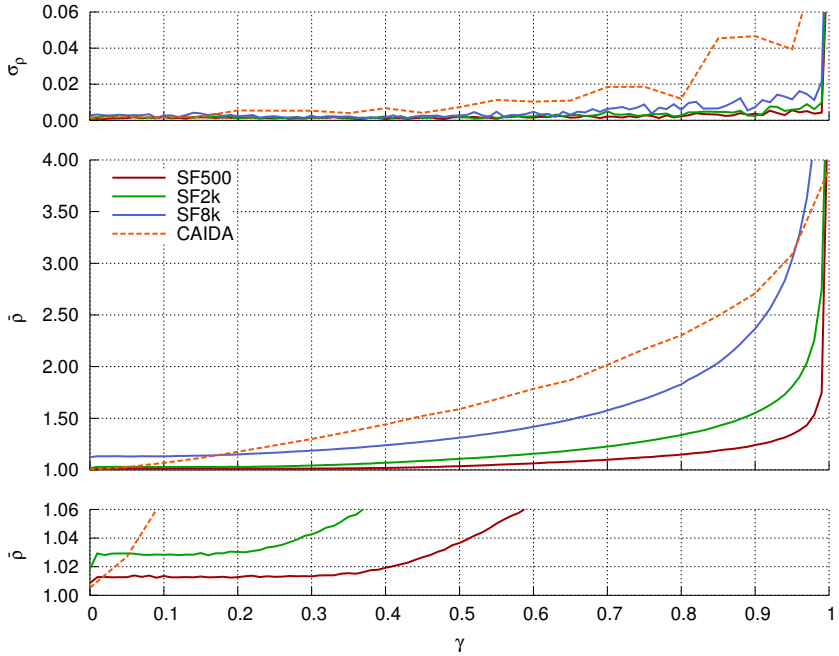
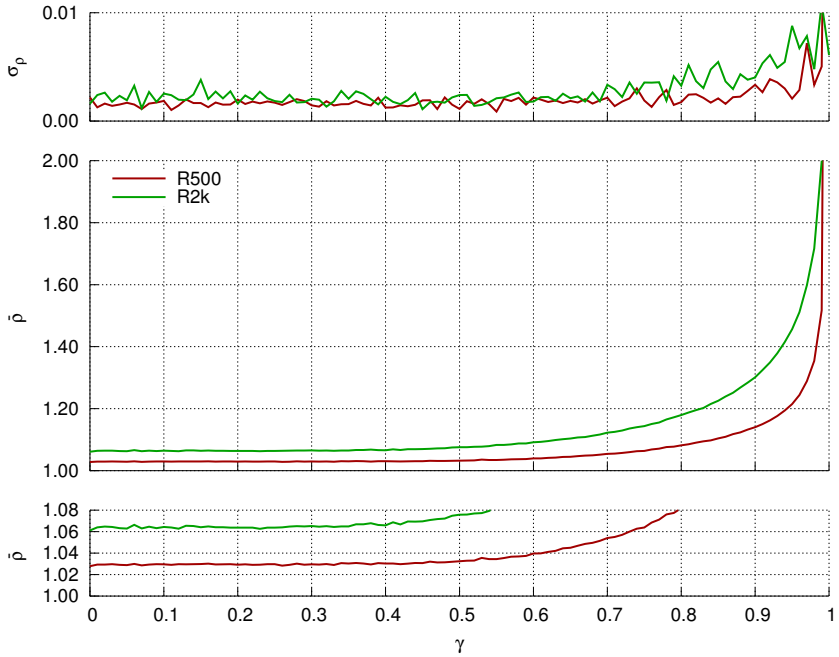


Figure 6.11: HFR: average stretch $\bar{\rho}$ and its standard deviation σ_ρ set out in function of varying γ-values, tested on different UDG-type networks embedded into $\mathbb{T}^{15}$.

Experiment 3. *For different γ-values from 0 to 1 with a step size of 0.01 and $\alpha = 1$ for $k = 15$ embeddings, 10^5 random source-destination pairs were generated. Pareto distributed traffic was simulated between them. The HFR system was based on a graph embedding constructed in RM*

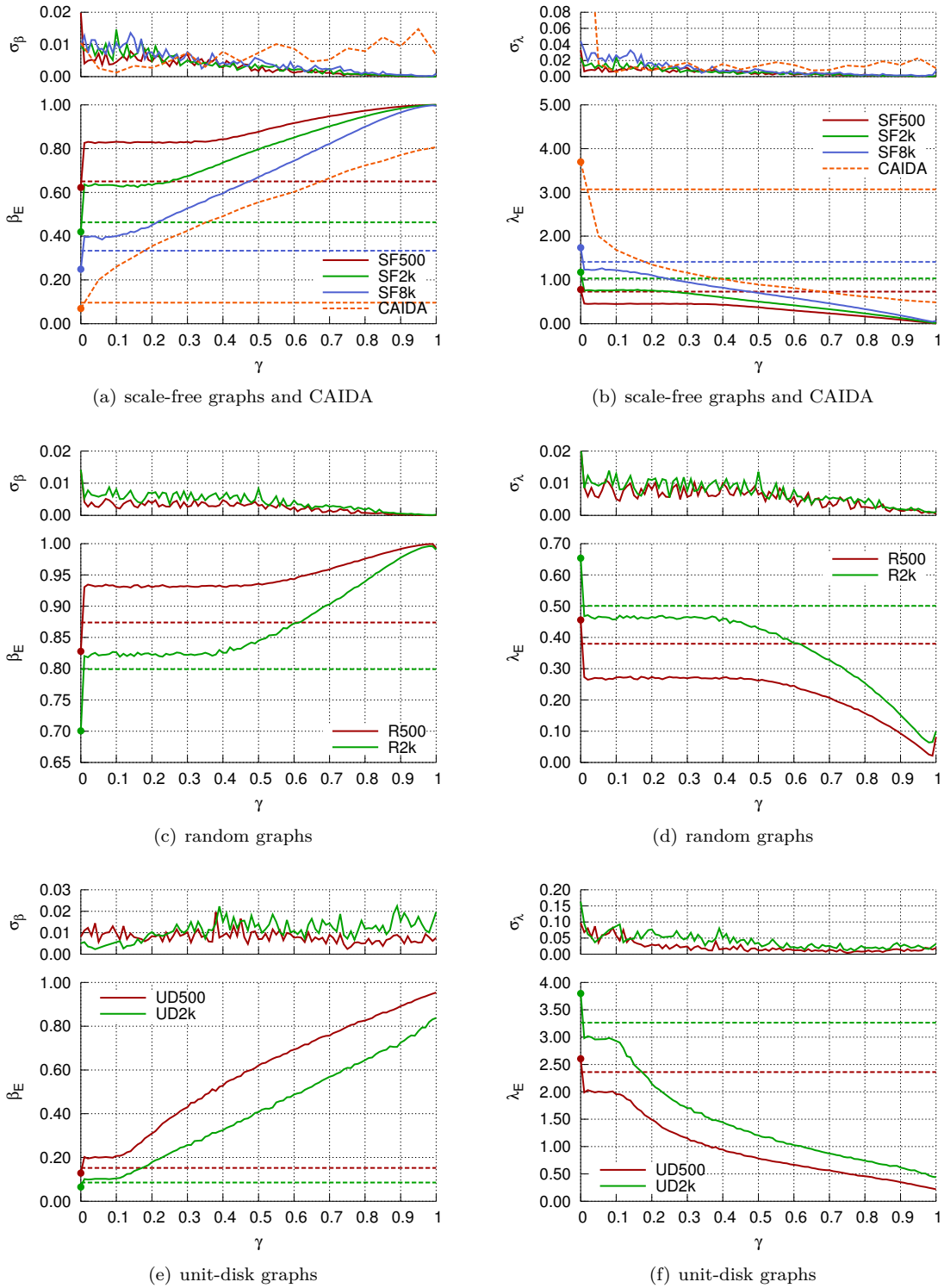


(a) scale-free graphs and CAIDA



(b) random graphs

Figure 6.12: HFR: average stretch $\bar{\rho}$ and its standard deviation σ_ρ set out in function of varying γ-values, tested on different scale-free networks and random networks as well as the CAIDA graph, embedded into $\mathbb{T}^{15}$.



(a) scale-free graphs and CAIDA

(b) scale-free graphs and CAIDA

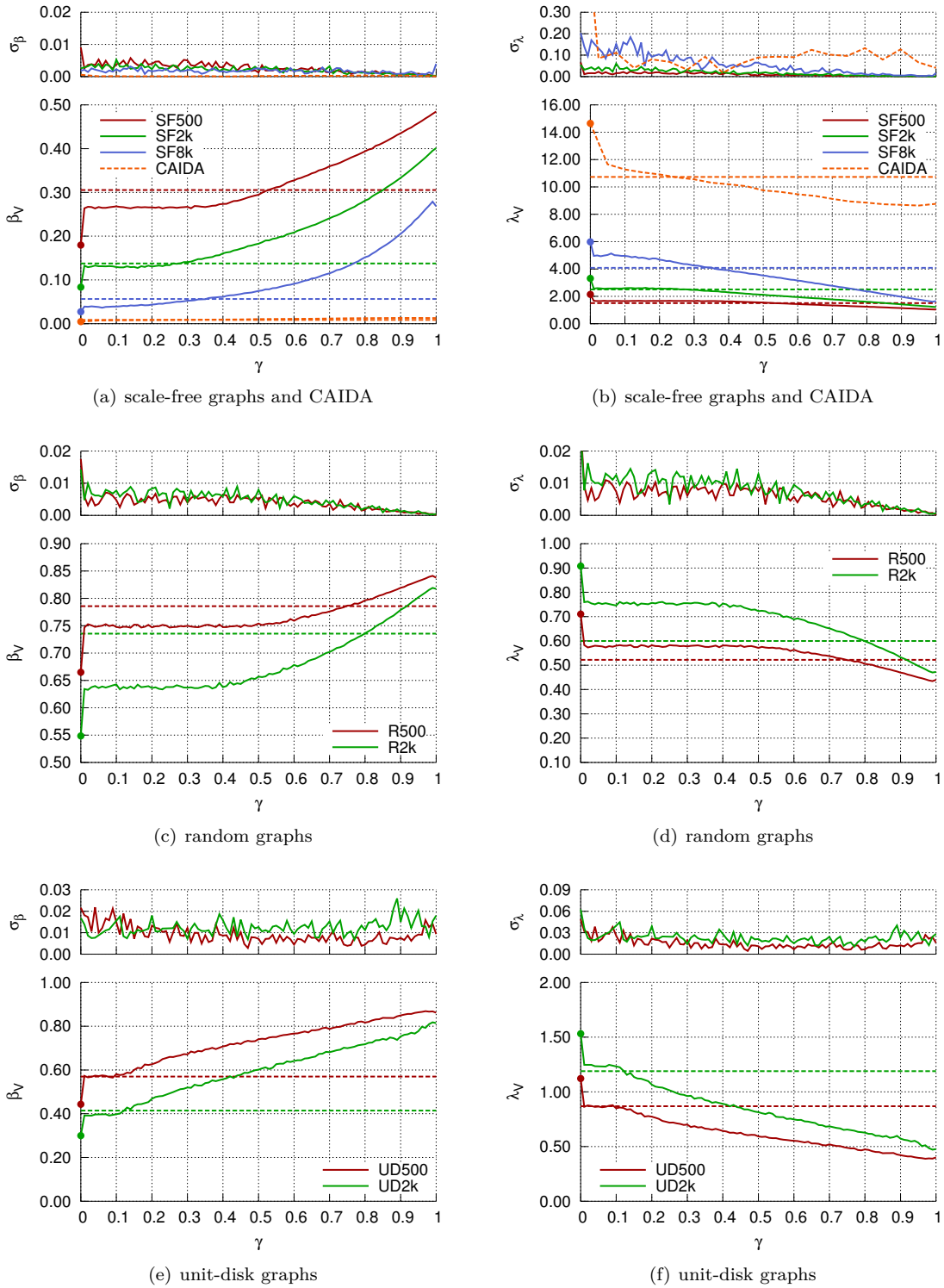
(c) random graphs

(d) random graphs

(e) unit-disk graphs

(f) unit-disk graphs

Figure 6.13: HFR: load balancing metrics for links (β_E and λ_E) and their standard deviation (σ_β and σ_λ) set out in function of varying γ-values, tested on different networks embedded in $\mathbb{T}^{15}$. The dashed horizontal lines represent $\beta_{\Pi,E}$ and $\lambda_{\Pi,E}$, the load balancing metrics for shortest path routing.



(a) scale-free graphs and CAIDA

(b) scale-free graphs and CAIDA

(c) random graphs

(d) random graphs

(e) unit-disk graphs

(f) unit-disk graphs

Figure 6.14: HFR: load balancing metrics for nodes (β_V and λ_V) and their standard deviation (σ_β and σ_λ) set out in function of varying γ-values, tested on different networks embedded in $\mathbb{T}^{15}$. The dashed horizontal lines represent $\beta_{\Pi,V}$ and $\lambda_{\Pi,V}$, the load balancing metrics for shortest path routing.

with HKE based on randomly assigned node keys. For each γ-value this random source-destination generation process was executed 10 *times. Over these* 10 *runs, the average and standard deviation of the different metrics were calculated. This experiment is executed on the scale-free networks SF500, SF2k and SF8k; the random networks R500 and R2k; the UDG-type networks UD500 and UD2k. For the CAIDA graph the γ-values ranged from* 0 *to* 1 *with a step size of* 0.05 *and the experiment was ran* 5 *times for each γ-value.*

Figure 6.12(a) show that at low γ-values the stretch is equally low as with GFR for scale-free graphs. The stretch remains steady for $0 \leq \gamma \leq 0.5$. Afterwards, $\bar{\rho}$ starts to incline quickly as $\gamma \to 1$. This can also be seen in Figures 6.11 and 6.12(b). This is consistent with HFR approximating LBFR as $\gamma \to 1$ and with HFR approximating GFR as $\gamma \to 0$. It can also be noticed that the stretch increases faster for larger graphs. Although it has to be noted that the stretch increase between graphs of different sizes does not differ a lot for low γ-values.

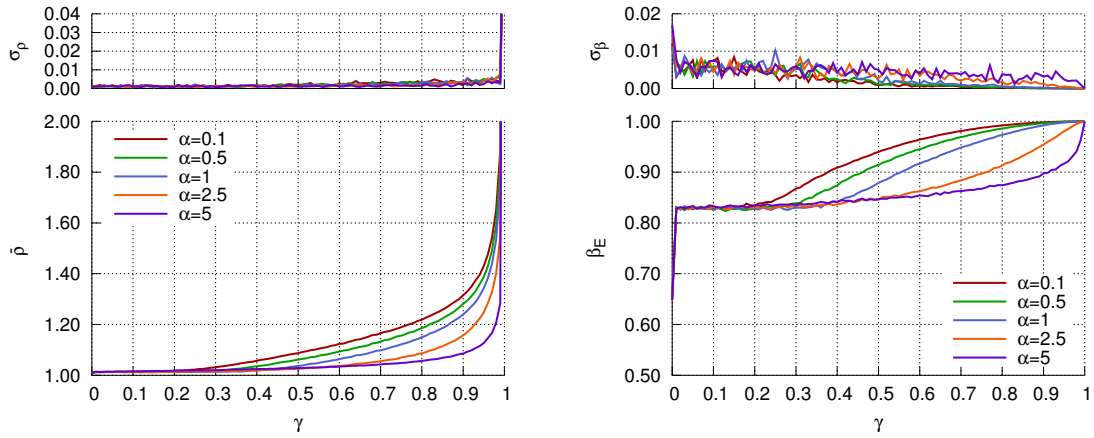
When investigating Figure 6.13, the β_E- and λ_E-values indicating the load balancing behaviour for links approach near perfect load balancing for $\gamma \to 1$. This is consistent with the system approaching LBFR as $\gamma \to 1$. The β_V- and λ_V-values converge to the values of LBFR. When the system has a parameter $\gamma = 0$ the HFR exacerbates the same load balancing behaviour as the GFR system. Something interesting happens when looking to the right of $\gamma = 0$. For $\gamma > 0$ a step can be noticed such that β suddenly increases and λ suddenly drops. When looking at Figure 6.11 and Figure 6.12, there is no such a big increase in stretch. This can be explained by the fact that when a node has to make a forwarding decision, lots of potential candidates will have an equal distance to the destination. Therefore it does not matter which node to take as next node in terms of stretch, and as such a random decision will be made. However, when taking into account load balancing, a huge improvement can be made by prioritizing those links with a low current load. This holds for all types of graphs, although the step size may vary. This analysis is also applicable to the node load balancing metrics in Figure 6.14.

In the previous figures the shortest path load balancing metrics are represented as the dashed horizontal lines. In Figure 6.13 it can be observed that $\gamma = 0$ results in load balancing behaviour that is worse than the passive load balancing behaviour of shortest path routing, which is entirely consistent with the results in Section 6.1. However, as γ increases, this passive load balancing behaviour is overcome. The same holds true for the node load balancing behaviour in Figure 6.14 but there the γ-value required to cross the passive node load balancing behaviour of shortest path routing is higher. It can also be observed that the same γ-value results in worse load balancing behaviour and a higher stretch for larger graphs, when comparing them to smaller graphs. This is consistent with the results of GFR and LBFR.
To conclude it can be said that shifting γ to zero results in HFR approximating GFR, while shifting γ to one leads to HFR approximating LBFR. Between these two values a wide spectrum of stretch-load balancing combinations exists.

6.3.2 Sensitivity analysis: trade-off function α-parameter

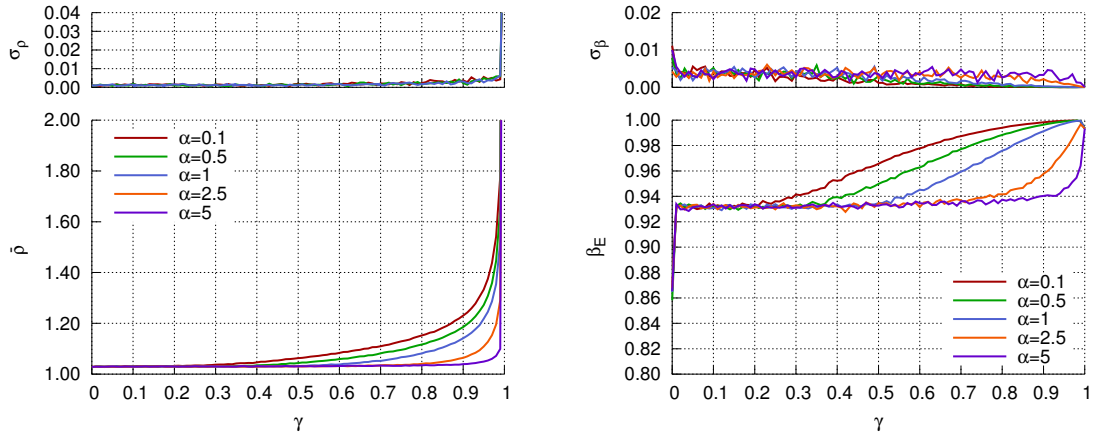
In this section the effect of changing the α-parameter in the cost function Eq. (5.16) is examined.

Experiment 4. *For different γ-values, ranging from 0 to 1 with a step of 0.01, with different values for α in $\{0.1, 0.5, 1, 2.5, 5\}$ and $k = 15$ embeddings, 10^5 random source-destination pairs were generated. Pareto distributed traffic was simulated between them. The graph embeddings were generated in RM with HKE using randomly assigned node keys. This was conducted 10 times on the graph SF500, R500 and UD500 for each k- and α-value. For the CAIDA graph each experiment was run 5 times for γ step value of 0.05.*



(a) average stretch $\bar{\rho}$ (b) link load balancing metric β_E

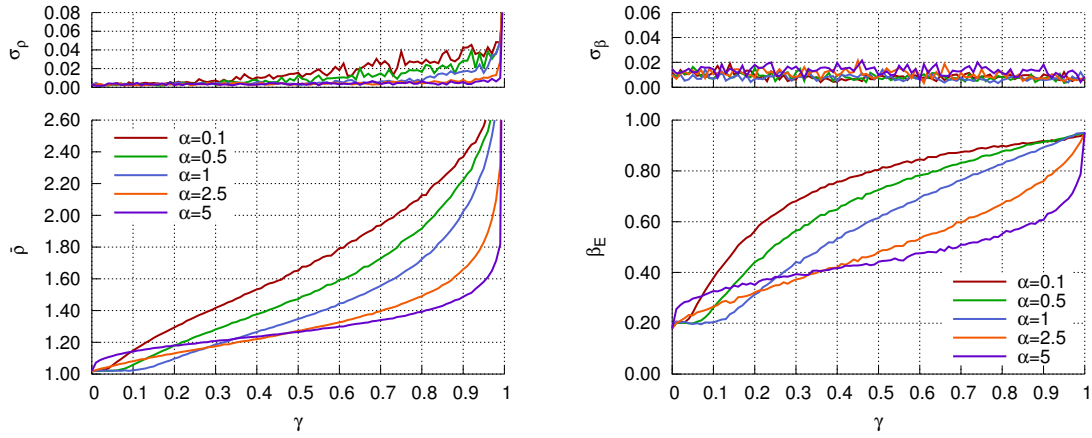
Figure 6.15: HFR: sensitivity analysis of α for an embedding into $\mathbb{T}^{15}$ for the graph SF500.



(a) average stretch $\bar{\rho}$ (b) link load balancing metric β_E

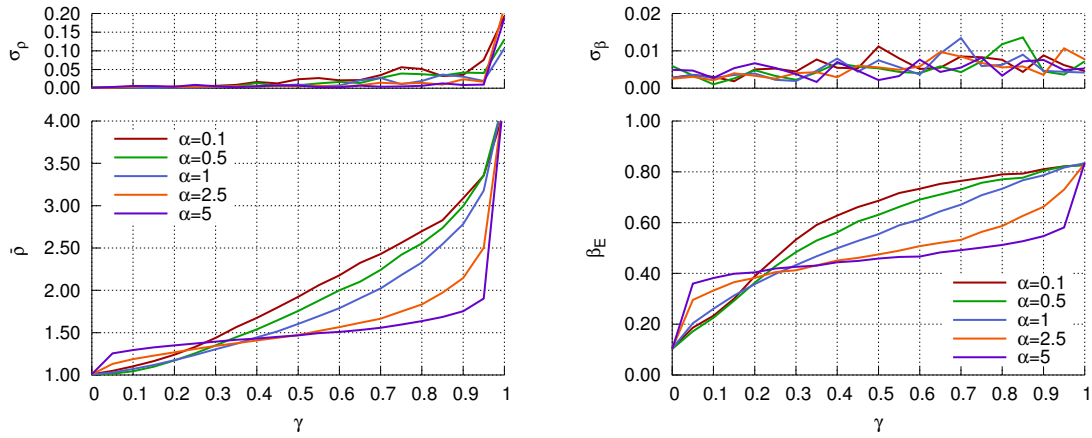
Figure 6.16: HFR: sensitivity analysis of α for an embedding into $\mathbb{T}^{15}$ for the graph R500.

The results are shown in Figure 6.15 for the scale-free graphs, in Figure 6.16 for the random graphs, in Figure 6.17 for the unit-disk graphs and in Figure 6.18 for the CAIDA graph. Raising α flattens the first part of the curve (both for $\bar{\rho}$ and β_E), thus making the cost function less sensitive to γ at low γ-values. It can also be seen that in Figures Figures 6.15(a), 6.16(a), 6.17(a) and 6.18(a)



(a) average stretch $\bar{\rho}$ (b) link load balancing metric β_E

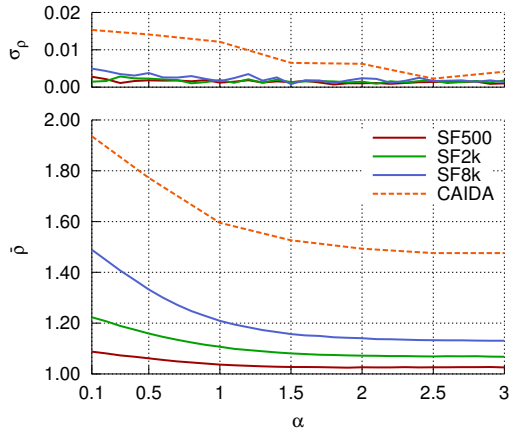
Figure 6.17: HFR: sensitivity analysis of α for an embedding into $\mathbb{T}^{15}$ for the graph UD500.



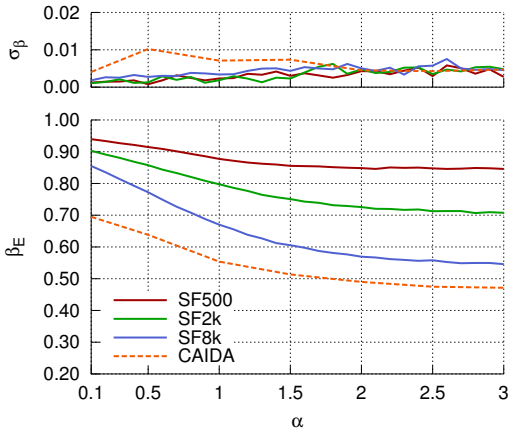
(a) average stretch $\bar{\rho}$ (b) link load balancing metric β_E

Figure 6.18: HFR: sensitivity analysis of α for an embedding into $\mathbb{T}^{15}$ for the graph CAIDA.

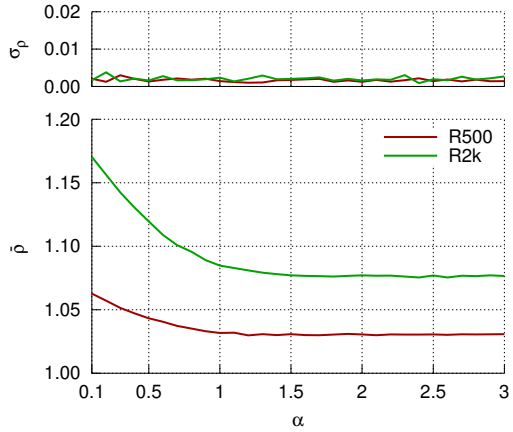
the steep increase in stretch occurs at higher γ-values compared to Figure 6.12(a). This can be useful when the stretch should be heavily prioritized by the routing scheme. By using a larger α, the γ-parameter can be more finely tuned. In Figures 6.15(b), 6.16(b), 6.17(b) and 6.18(b) it can be seen that the improved load balancing behaviour accompanying larger γ-values also occurs at higher γ-values when α increases. Figures 6.17 and 6.18, showing the previous metrics for unit-disk graphs and the CAIDA graph, highlights a slightly different trend than the other plots. Here it can be witnessed that the curve curvature alters when α varies. This could be an effect originating from the non-linear tuning (α is an exponent) of the distance and load balancing terms. Next, the effect of tuning α with a constant γ-value is examined. These results should be consistent with the previous experiment, thus increasing α should lead to a lower average stretch and worse load balancing performance.



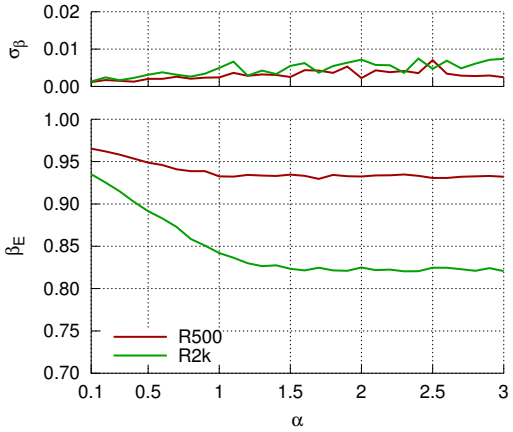
(a) scale-free graphs and CAIDA



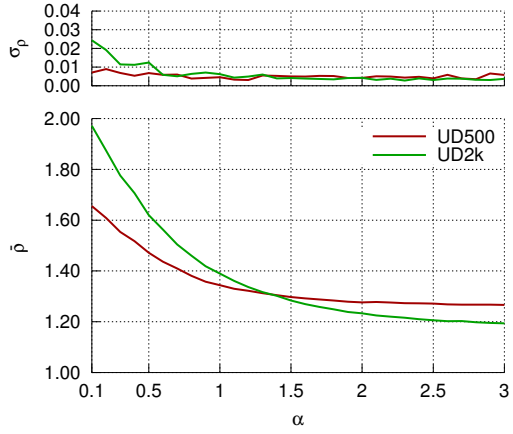
(b) scale-free graphs and CAIDA



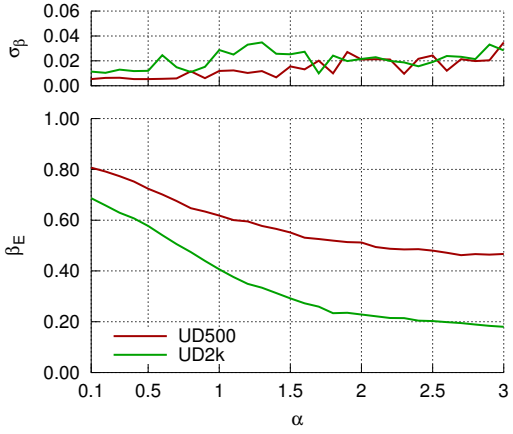
(c) random graphs



(d) random graphs



(e) unit-disk graphs



(f) unit-disk graphs

Figure 6.19: HFR: average stretch $\bar{\rho}$ and link load balancing β_E with their standard deviation (σ_ρ and σ_β) set out in function of varying α-values, tested on different networks embedded into $\mathbb{T}^{15}$.

Experiment 5. *For different α-values, ranging from* 0.1 *to* 3 *with a step of* 0.1*, with $\gamma = 0.5$ and $k = 15$ embeddings, 10^5 random source-destination pairs were generated. Pareto distributed traffic was simulated between them. The graph embeddings were generated in RM with HKE using randomly assigned node keys. This was conducted* 10 *times on the graphs SF500, SF2k, SF8k, R500, R2k, UD500 and UD2k for each α-value. For the CAIDA graph the experiment was run* 5 *times with an α step value of* 0.5*.*

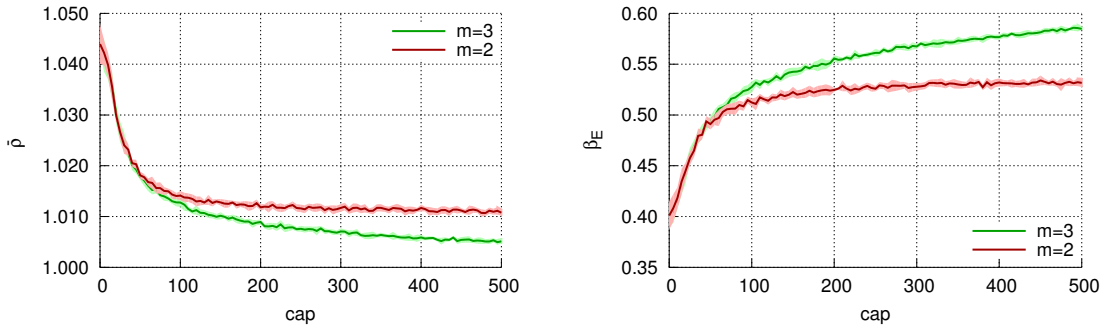
The results are shown in Figure 6.19. It can be seen that as α increases, the average stretch as well as β_E decreases. This means that increasing α has a positive effect on the average path length and a negative effect on the load balancing performance of HFR. This is consistent with the results of Experiment 4. The trend to be witnessed of $\bar{\rho}$ and β_E in function of α is similar for all types of networks. The effect of changing α seems greater, the larger the graph is. In Figure 6.19(e), much like in Figure 6.17(a), a distinct trend can be witnessed in which the two curves meet in a single point. This might be explained by the fact that the stretches of the two unit-disk graphs are very similar for varying γ-values, which can also be seen in Figure 6.11.
A general conclusion is that as α increases, HFR puts more emphasis on a lower stretch and less emphasis on load balancing behaviour for γ-values in the mid-range.

6.3.3 Effect of an m-hop neighbourhood

In this section the effect of employing different neighbourhood sizes is investigated as described in Section 5.3.2. When a node u uses an m-hop neighbourhood it not only has information (coordinate, key and traffic load information) about its first degree (1-hop) neighbours $N_1(u)$, but also information about the nodes in $N_i(u)$ for $1 \leq i \leq m$. It is expected that using a larger neighbourhood increases a node's chances of finding a lowly congested short path to the destination for a packet that has to be forwarded. To test the effects on stretch and load balancing the following experiment has been conducted.

Experiment 6. *The effects of an m-hop neighbourhood are tested by generating 10^5 random source-destination pairs between which Pareto distributed traffic is simulated. The HFR system used a graph embedding procedure in RM with HKE and randomly assigned node keys. The parameters in Eq.* (5.16) *were set to $\alpha = 1$ and $\gamma = 0.1$. The number of embeddings is $k = 15$. Both values $m = 2$ and $m = 3$ were tested with a cap size ranging from* 0 *to* 500 *with a step size of* 10*. For each cap size and m-value, the experiment was executed* 10 *times over which the average value and standard deviation was calculated for the tested metrics. The experiment was executed on the benchmark network SF8k.*

In Section 5.3.2 the effect of employing a capped neighbourhood versus an uncapped one on the computational complexity was already shown in Figure 5.6(b). Using a cap prevents the overloading of the small number of high-degree nodes (see the top of the boxplot) while still dynamically allowing low-degree nodes to reap the benefits of having a full m-hop neighbourhood at their disposal. In Figure 6.20 the effects on the load balancing metrics and stretch for different m-values can be seen, for a varying cap. Note that having a cap equal to zero is the same as $m = 1$, which is the default HFR scheme. For a 2-hop neighbourhood with a relatively small cap large gains in stretch and load balancing can be observed. At a cap of 200 the average path length only is



(a) average stretch $\bar{\rho}$

(b) link load balancing metric β_E

Figure 6.20: Effect of routing with HFR combined with an m-hop neighbourhood with varying cap-values on SF8k. The shaded background behind the curves represents $\bar{\star} \pm \sigma_\star$.

about 1% larger than the shortest path length. Employing a 3-hop neighbourhood further improves routing behaviour but the trade-off between link bandwidth sacrificing and state complexity for stretch or load balancing might not be worth it.

6.3.4 Graph embedding procedures

In this section the effect of using the different embedding procedure modes, redundant mode (RM), breadth-first mode (BFM) and breadth-first redundant mode (BFRM), in combination with the different root election procedures, highest-key node election (HKE), highest-degree node election (HDE) and anchor node election (AE), on the tree redundancy is examined. Also the correlation of different tree redundancy values, measured by the metric τ (see Definition 5.2), with the load balancing metric β_E and the average stretch $\bar{\rho}$ is investigated. It is expected that BFM results in the most tree redundancy (highest τ), RM in the least tree redundancy (lowest τ) and that BFRM lies somewhere in between.

Experiment 7. *The graph embedding procedure modes RM, BFM and BFRM were tested in combination with the different root election procedures HKE, highest-degree node election (HDE) and anchor node election (AE). For a different number of embeddings k between* 1 *and* 30 *with a step size of* 1, *τ was calculated for each mode-election combination. Whenever there was some kind of randomness present, such as electing root nodes by HKE where the node keys are randomly assigned, the test was ran* 30 *times over which the average and the standard deviation were calculated. The tests were executed on the scale-free benchmark network SF500.*

In Figure 6.21 it can be noticed that employing AE yields improvements over HKE when generating the embedding in BFM, while HDE generates leads to the highest τ-value. When generating the embeddings in BFRM the difference between the election variants becomes very small. Using HDE seems to lead to lower τ-values than HKE in BFRM. Generating the embeddings in RM, there is no additional benefit of using any specific root election mechanism. When heavily focusing on attaining minimal tree redundancy τ, the effect of the root election mechanism is overshadowed by the embedding generation mode.

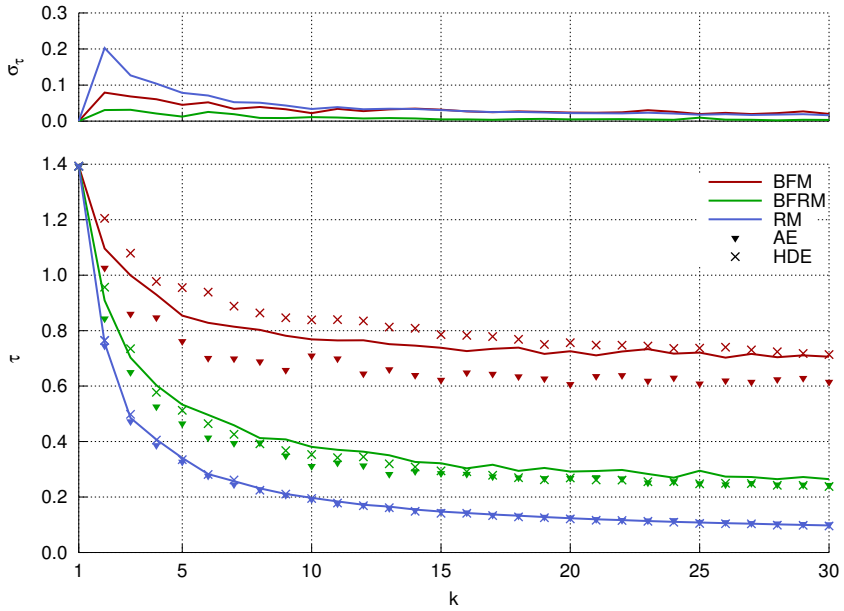


Figure 6.21: Tree redundancy τ in function of k, for the different graph embedding procedures combined with different root election mechanisms for the graph SF500. The solid lines represent HKE, the triangles AE and the crosses HDE. The different colours represent the different graph embedding modes.

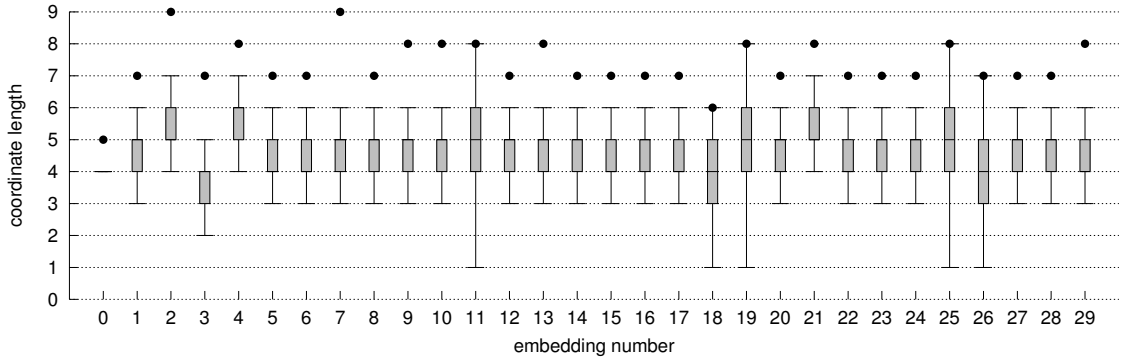
In the following experiment the effects of the different graph embedding modes on the assigned coordinate lengths are investigated. It is expected that a graph embedding mode focussing on minimizing tree redundancy (RM) would generate a deeper tree, leading to larger coordinates. The BFM and BFRM should lead to coordinates of nearly equal length as both modes generate trees of minimal depth.

Experiment 8. *The coordinate lengths produced by graph embedding procedure modes RM, BFM and BFRM with HKE are investigated. The three different modes are executed on the network SF500 and produce an embedding into* $\mathbb{T}^{30}$*, thus* $k = 30$ *embeddings into* $\mathbb{T}$*. For each distinct embedding, the coordinate lengths of all network nodes are gathered and used to calculated boxplot values.*

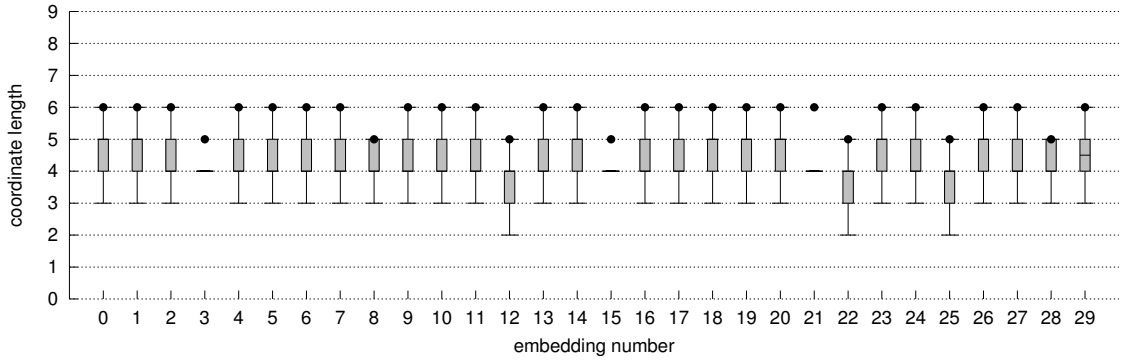
In Figure 6.22 the coordinate lengths are depicted for every embedding, for the different embedding modes, by means of a boxplot. When comparing BFM with RM it can be noticed that RM indeed generates larger coordinates in favour of less tree redundancy (lower τ). The trade-off made is thus larger storage requirements in the nodes and packet headers[1] versus potentially better load balancing. Also the variance of the coordinate lengths is much larger than when employing BFM or BFRM. It is remarkable that though the BFRM tree generation model has a τ-value that is close to the τ-value of RM, it still is a breadth-first algorithm leading coordinate lengths nearly equal to those obtained by using BFM.

The effect of a lower τ-value on the stretch and load balancing metrics is depicted in Figure 6.23 for GFR. This plot was generated by using the same set-up as Experiment 1 for all embedding

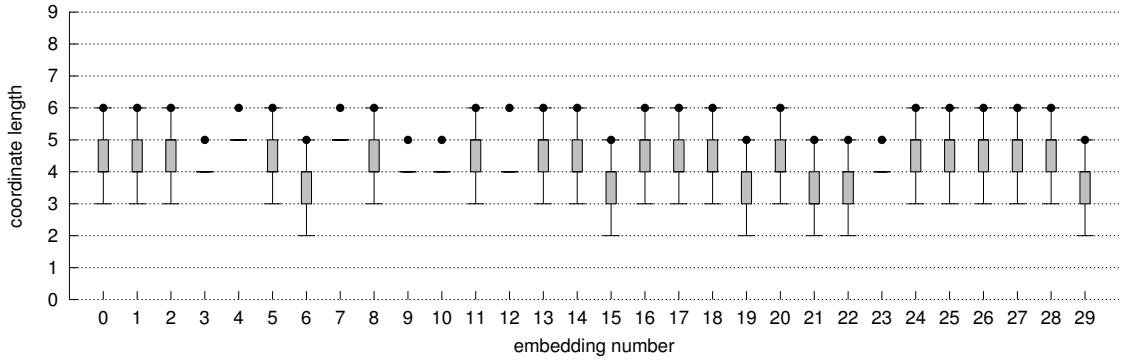
[1]Remind that the destination coordinates have to be stored in the packet being forwarded.



(a) RM embedding

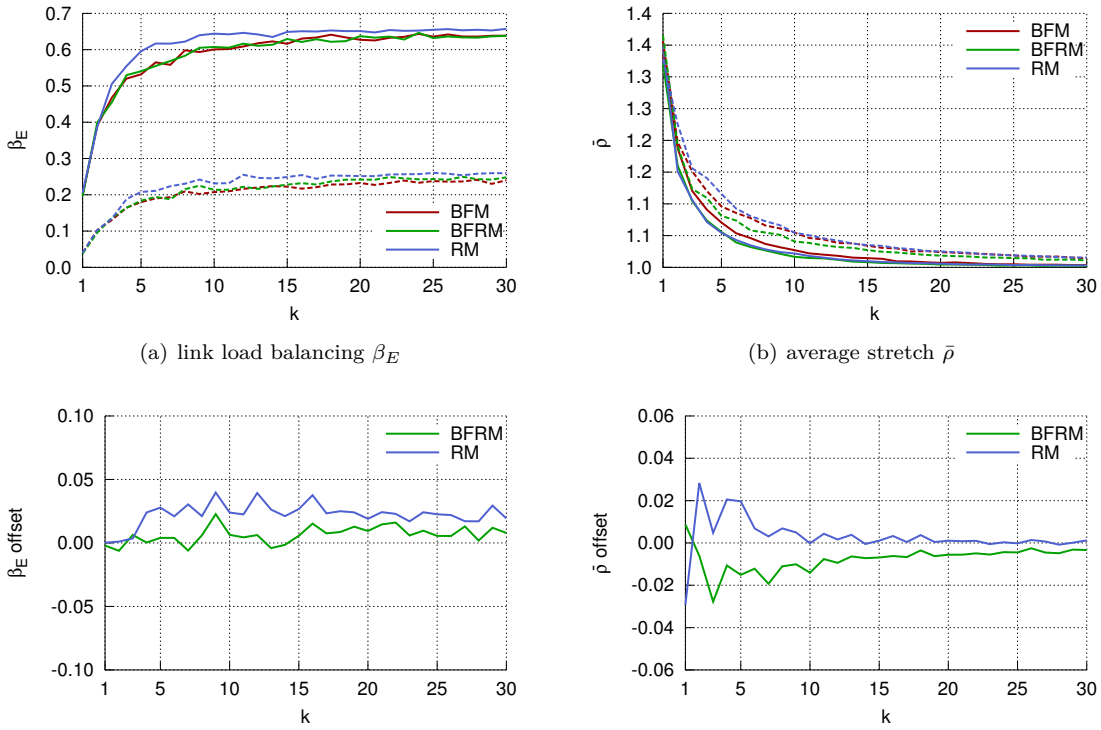


(b) BFRM embedding



(c) BFM embedding

Figure 6.22: Coordinate lengths in function of the embedding number for different embedding generation modes with HKE root election for the graph SF500. The dots represent the highest value attained.



(a) link load balancing β_E

(b) average stretch $\bar{\rho}$

(c) offset in β_E between non-BFM and BFM for SF8k

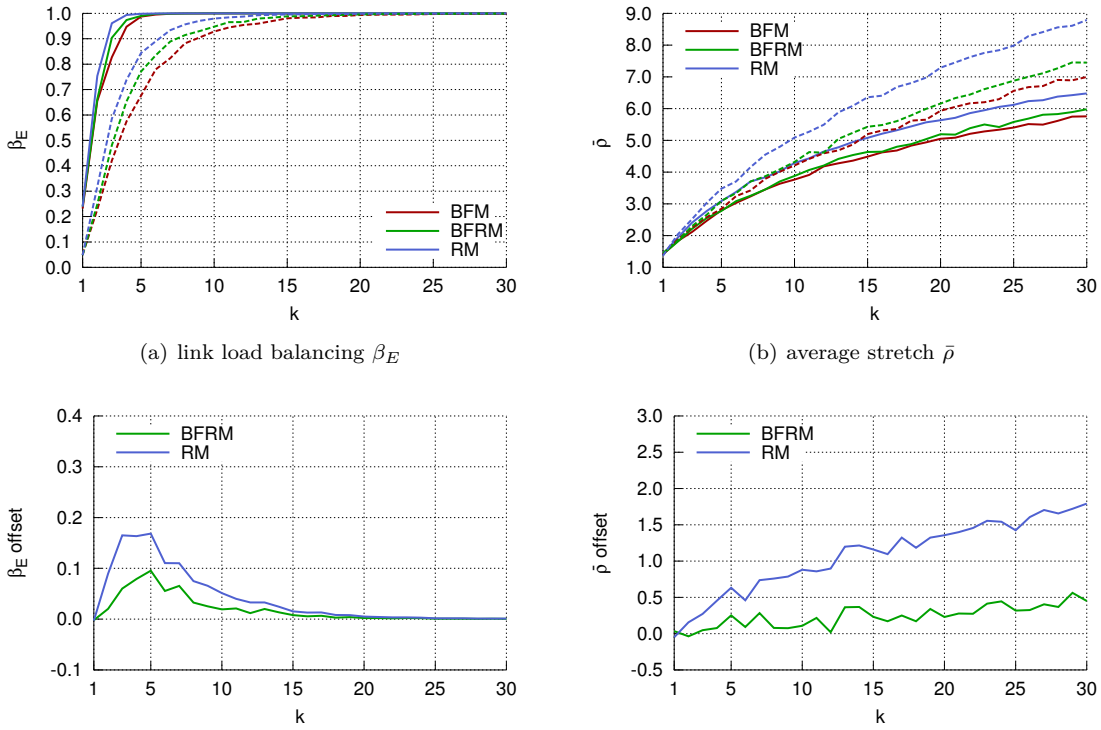
(d) offset in $\bar{\rho}$ between non-BFM and BFM for SF8k

Figure 6.23: GFR: the effect of the different tree growing modes on stretch and load balancing metrics. The solid line represents the data for the graph SF500, the dashed line SF8k in the top figures.

generation modes. In Figure 6.23(a) can be observed that the load balancing metric for links β_E rises slightly for large and small scale-free graphs when using RM compared to BFRM or BFM. When looking at Figure 6.23(b), it seems that BFRM achieves the lowest average stretch. However, the gain approaches zero as the number of embeddings increases. In Figures 6.23(c) and 6.23(d) the difference between the non-BFM and BFM is depicted for SF8k. From this it is clear that BFRM is an improvement over BFM as the same β_E value is reached with a decreased average stretch $\bar{\rho}$.

Figure 6.24 shows the effect of the different embedding modes on the stretch and load balancing behaviour for the LBFR system. For lower k values there is a large increase in β_E visible when using BFRM or RM over BFM. Due to the spreading of the different spanning trees, the curves converge faster to their asymptote. This means that using a non-BFM tree assignment mode leads to more possible routing paths. Because of the increased load balancing behaviour, the stretch will suffer as can be seen in Figures 6.24(b) and 6.24(d). However, the stretch increase of BFRM over BFM is only marginal at best, while some improvement in load balancing is visible.

When employing GFR, and thus focussing on stretch, the BFRM is able to decrease the stretch over BFM. When focussing on load balancing with LBFR, the BFM is able to achieve better load balancing performance. RM is able to perform better in both cases, but requires a complex



(a) link load balancing β_E

(b) average stretch $\bar{\rho}$

(c) offset in β_E between non-BFM and BFM for SF8k

(d) offset in $\bar{\rho}$ between non-BFM and BFM for SF8k

Figure 6.24: LBFR: the effect of the different tree growing modes on stretch and load balancing metrics. The solid line represents the data for the graph SF500, the dashed line SF8k in the top figures.

cycle avoidance and resolution mechanism. Furthermore it generates larger coordinates with more variance. Therefore the question whether to use RM or BFRM depends on whether stretch and load balancing performance are more important than low complexity and consistent coordinates of low length.

6.3.5 Employing non-uniform link capacities

In this section the effects of using link bandwidth capacities is explored, as explained in Section 5.5.1. When link capacities are used, the cost function in Eq. (5.16) is slightly altered to Eq. (5.22). In the following experiment, the capacities of each link follow a normalized Gaussian distribution to make sure that there is some form of controlled variance regarding the capacity values. It is expected that using no congestion awareness scheme leads to severe congestion of low-capacity links. This should manifest itself in lower success rates as more traffic is sent through the network. The congestion-aware load balancing (CALB) scheme should be able to spread traffic over the network regardless of the use of link capacities.

Experiment 9. *Every link of the network SF500 is assigned a capacity. The routing mechanism was assigned a cut-off ω-value of* 0.9 *for Eq.* (5.22)*. When trying to route traffic such that it would exceed its capacity, it is dropped. The capacities c are taken randomly from the range* $[c_{min}, c_{max}]$

and are accepted with a probability

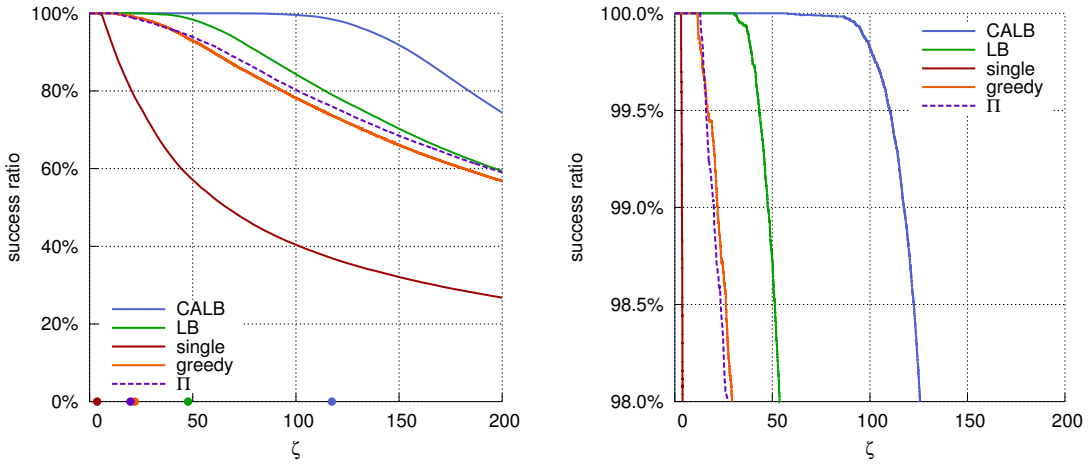
$$p(c) = \exp\left(-\frac{(c-\mu)^2}{2\sigma^2}\right) \tag{6.1}$$

which is a normalized Gaussian distribution with a standard deviation σ *and a mean* μ *defined by*

$$\sigma = \frac{1}{4}(c_{max} - c_{min}) \tag{6.2}$$

$$\mu = \frac{1}{2}(c_{max} + c_{min}) \tag{6.3}$$

with $c_{min} = 10$ *and* $c_{max} = 100$ *for this experiment. Routing success was tested for GFR with a single embedding, default HFR without congestion awareness and HFR with congestion-aware load balancing (CALB) both with the settings* $\alpha = 1$, $\gamma = 0.1$ *with* k *(varying) embeddings generated in RM with HKE using randomly assigned node keys. With the link capacities set, random source-destination pairs were generated and Pareto distributed traffic was simulated between them. The success ratio was monitored as more paths were generated. The better the system is able to cope with link capacities, the more paths it will be able to generate before breaking down (severe decrease in success ratio).*



(a) Average success ratio in function of ζ. The dots shown on the x-axis represent the point where the success ratio drops below 99%.

(b) zoomed in on y-axis of Figure 6.25(a)

Figure 6.25: Average success ratio in function of ζ for different routing schemes on the scale-free graph SF500.

In this experiment traffic was generated such that the minimum link capacity that may be assigned is ten times greater than the largest traffic that can be generated by Eq. (3.1) between two nodes. Thus the routing algorithms have plenty of room to spread traffic such that it fits the within the link capacities. The path ratio ζ is the number of paths generated divided by the number of vertices for that graph. ζ_Π is this value for a shortest path routing algorithm. Shortest path routing is used as a baseline for comparing the different algorithms.

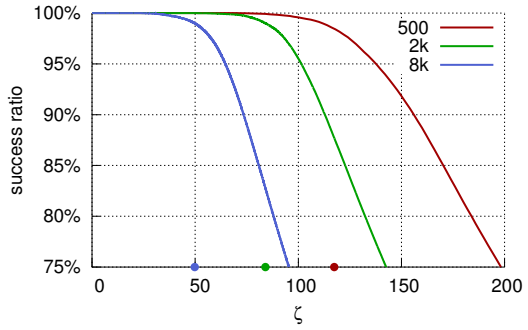
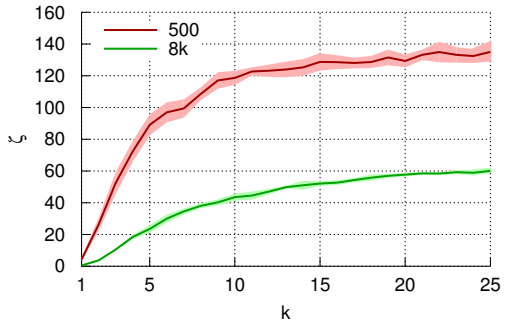
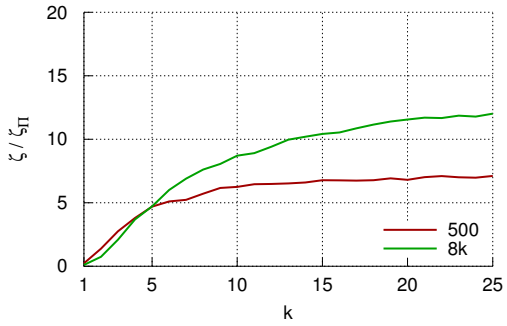


Figure 6.26: Average success ratio for HFR with CALB active for different scale-free graphs in function of ζ. The dots shown on the x-axis represent the point where the success ratio drops below 99%.



(a) 99% success rate cut-off in function of number of embeddings for HFR with CALB. The shaded background behind the curves represents $\bar{\star} \pm \sigma_\star$.



(b) ζ divided by the shortest path ζ-value, $\frac{\zeta}{\zeta_\Pi}$, in function of k.

Figure 6.27: Comparison between the scale-free graphs SF500 and SF8k when investigating the ζ-value at 99% success ratio.

In Figure 6.25 it can be seen that as more traffic is send through the network each routing mechanism deteriorates when link capacities are set. This is caused by the fact that every link has a weight which is initially set to zero and increases as traffic is routed through it. Therefore when link capacities are set the question is not whether traffic will be dropped, but when. It can be seen that using a single-embedding GFR routing scheme, congestion is not avoided at all. Even at relatively low ζ-values, the curve drops rapidly. This can be explained by the fact that a traffic hotspot is created around the root node, leading to heavy congestion at this point in the network which causes traffic to be dropped. Using multiple embeddings ($k = 10$) in combination with GFR heavily increases routing performance, even though no congestion avoidance mechanism is used. This is consistent with the fact that there now exist multiple roots to distribute the root hotspot behaviour and that more alternative paths are available. When looking at shortest path routing, it performs the same as GFR for low ζ-values and converges to the success ratio of HFR without CALB. The fact that HFR performs better than GFR leads us to believe that load balancing also inherently avoids congestion. When CALB is active for the HFR mechanism, a huge improvement can be noticed. This is logical as the routing scheme will now actively employ link capacity in-

formation to steer traffic streams. In Figure 6.26 the HFR with CALB is evaluated for different scale-free graphs. The different positions of the curves could be due to a certain bias in the ζ function. The most important result is that the curve follows the same trend for all graph sizes.

The ζ-values are set out in function of the number of embeddings k in Figure 6.27. In Figure 6.27(a) the 99% success rate cut-off point is shown for the graphs SF500 and SF8k. This also means that the ζ-value seen on the this plot for $k = 10$ corresponds with the dots in Figure 6.26. The same trend can also be witnessed: smaller graphs are capable of generating relatively more paths before breaking down. To further investigate this Figure 6.27(b) shows the ζ value divided by the shortest path ζ-value (ζ_Π). From this plot is clear that although the absolute ζ-value at which 99% success rate is crossed is lower for larger graphs, the improvement over shortest path routing is larger.
A general conclusion is that using CALB, which is an extension of the default load balancing scheme, allows HFR to cope with link bandwidth restrictions. This leads to an overall higher routing success ratio.

6.3.6 Fault-tolerance

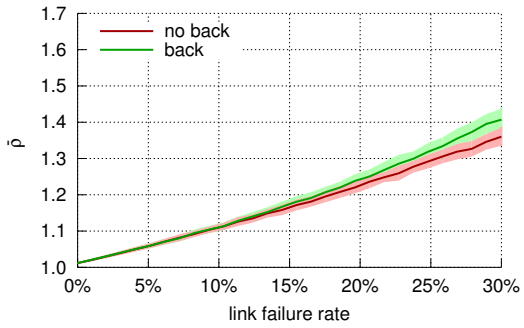
In this section the fault-tolerance of the HFR system is evaluated, its resilience towards link and node failures. The routing success rate and stretch for HFR with and without the backup routing mechanism (see Section 5.5.2) is evaluated under diverse network failure scenarios.

Experiment 10. *The experiment tests HFR ($\alpha = 1$, $\gamma = 0.1$, $k = 15$) in RM with HKE and randomly assigned node keys, under a network failure scenario by removing a number of links or nodes without disconnecting the graph. The number of links or nodes removed starts at* 0 *and goes up by* 20 *at each step. When representing the network by a graph $G = (V, E)$, the highest number of links that can be removed without disconnecting G is $(|E| - |V| + 1)$. For every step in the number of links or nodes failed, the experiment is run* 100 *times and every time* 1000 *random source-destination pairs are generated between which Pareto traffic is simulated. For each of these* 1000 *pairs the average stretch $\bar{\rho}$ is calculated as well as its standard deviation σ_ρ. Also the average success ratio is examined, which is the percentage of generated source-destination pairs having a void on the routed path between them. To test link fault-tolerance, links are removed with a probability*

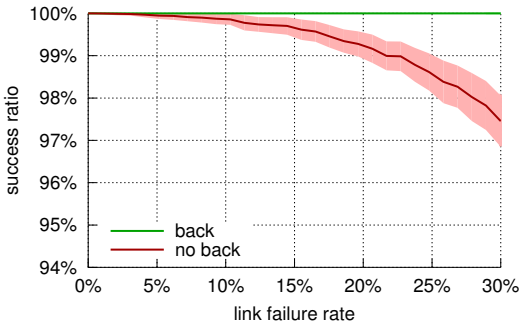
$$p(l) = \exp\left(3\left(1 - \frac{d_G(v)_o}{(d_G(v) - 1)}\right)\right) \tag{6.4}$$

with $v \in V$ the vertex that has the lowest $p(l)$ probability of both vertices incident to $l \in E$; $d_G(v)_o$ the degree of vertex v before removal of any links. This probability makes sure that the link failures are evenly spread across the network. When $d_G(v) = d_G(v)_o$ (no incident links failed), the failure probability $p(l)$ goes to one, while it goes to zero for $d_G(v) = 0$ (nearly all incident links failed). Node fault-tolerance is tested by randomly removing nodes, as well as their incident links, from the network without disconnecting the graph.

A general trend to be witnessed in Figure 6.28 is that the stretch goes up and the success ratio decreases (without backup mechanism) as the number of failed links rises. This is explainable by the fact that the spanning trees T_i used to build the embeddings $\mathcal{T}_i$ lose their resemblance to the underlying network topology due to the link failures. In Figure 6.28(a) can be seen that the average

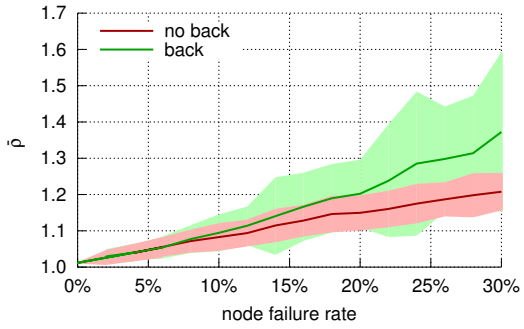


(a) Average stretch $\bar{\rho}$ with σ_ρ. Note that in case of route failures, the stretch $+\infty$ is excluded from the data.

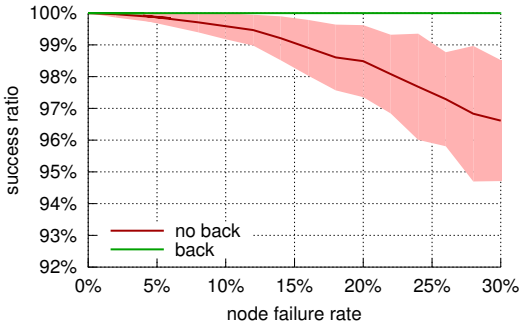


(b) average success ratio and its standard deviation

Figure 6.28: Average stretch $\bar{\rho}$ and routing success ratio on SF500 in function of the percentage of deleted links, with and without the backup routing procedure active. The shaded background behind the curves represents $\bar{\star} \pm \sigma_\star$.



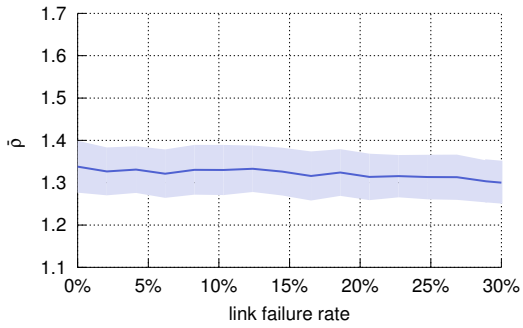
(a) Average stretch $\bar{\rho}$ with σ_ρ. Note that in case of route failures, the stretch $+\infty$ is excluded from the data.



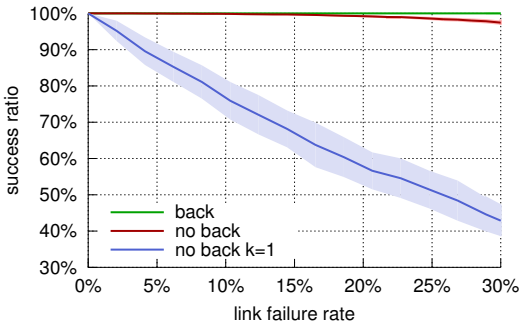
(b) average success ratio and its standard deviation

Figure 6.29: Average stretch $\bar{\rho}$ and routing success ratio on SF500 in function of the percentage of deleted nodes, without the backup routing procedure active. The shaded background behind the curves represents $\bar{\star} \pm \sigma_\star$.

stretch goes up faster when the backup mechanism is active. This can be explained by the fact that the stretch of paths with voids (which is actually $+\infty$) is not present in the data. What also follows from this picture is that the paths taken by the backup mechanism are generally longer than the paths taken by the default HFR system. In Figure 6.28(b) it is clear that HFR with the backup mechanism attains 100% routing success rate. Although this is a huge improvement over the default HFR system, the default system is still able to attain a very high routing success ratio, even at high link failure rates. In Figure 6.29 the HFR without backup mechanism is shown but for a varying number of failed nodes. Here it can be seen that the trend is nearly equal in terms of stretch and success rate, although the slope of the curve is less steep for higher failure rates. The success rate for a percentage of failed nodes seems to be slightly lower than for the same percentage of failed links. Also the standard deviation is larger, especially when the backup routing mechanism is active. This can be explained by the fact that there is a huge difference

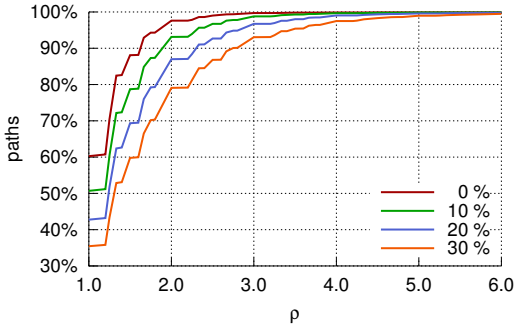


(a) Average stretch $\bar{\rho}$ with σ_ρ. Note that in case of route failures, the stretch $+\infty$ is excluded from the data.

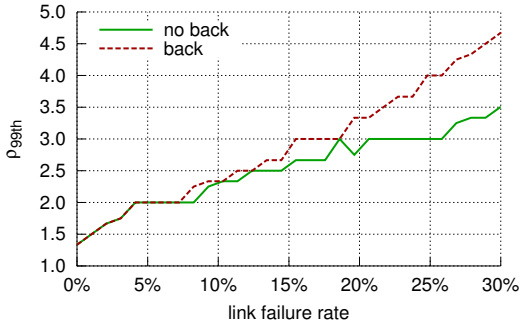


(b) average success ratio and its standard deviation

Figure 6.30: HFR with and without backup mechanism for 15 embeddings compared to routing on a single embedding with GFR for a varying number of failed links. The shaded background behind the curves represents $\bar{\star} \pm \sigma_\star$.



(a) cumulative stretch distribution for various link failure rates with backup routing system active



(b) stretch at 99th percentile ρ_{99th}

Figure 6.31: Cumulative number of paths with a certain stretch ρ for different failure ratios and the stretch ρ_{99th} of the 99th percentile with and without backup system for network SF500.

between a node failing that has a high degree versus a low-degree node failing.

In Figure 6.30 HFR is compared with a single-embedding GFR system without a backup mechanism for a varying number of failed links. In Figure 6.30(a) it can be noticed that the stretch remains nearly constant, which is in line with the fact that the infinite stretch of voids is not incorporated in the data and that the success ratio is low. The GFR system seems unable to take alternative routing paths, which is logical as there is only one embedding available. Only source-destination pairs for which there exists no void on their path in this single embedding will be able to communicate. Therefore the stretch has to remain constant. In Figure 6.30(b) the success rate is compared with the HFR system for a graph embedding in $\mathbb{T}^{15}$. Here it becomes clear that the HFR system possesses inherent fault-tolerance, which can be noticed by the huge improvement over the single-embedding GFR scheme. This can only be explained by the presence of multiple embeddings in combination with the load balancing scheme which allow traffic to be rerouted around voids along alternative paths. These rerouted traffic streams come at a cost of an increased stretch.

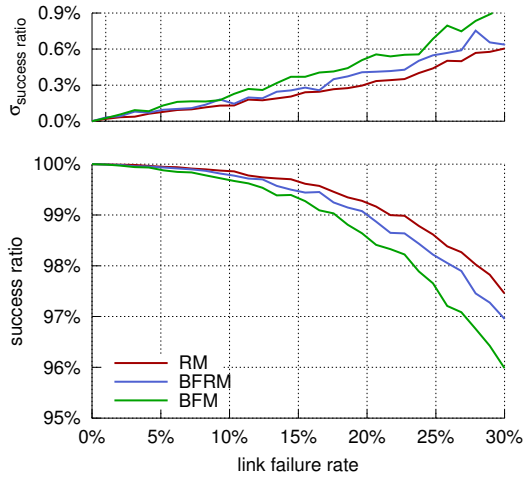


Figure 6.32: Fault-tolerance of different coordinate assignment procedures for HFR

Figure 6.31(a) shows the cumulative stretch distribution for different link failure rates with the backup system active. It shows that as the backup mechanism has to be used more often (which happens when more links are defect) more paths have a higher stretch. The distribution seems to shift to the right. This supports that the backup routing mechanism reroutes traffic along paths that are longer than the paths generated by HFR in case of no failures. In Figure 6.31(b) the average stretch of the paths with the 1% highest stretch values is depicted in function of a varying number of failed links. Here can also be noticed that the backup mechanism increases the stretch.

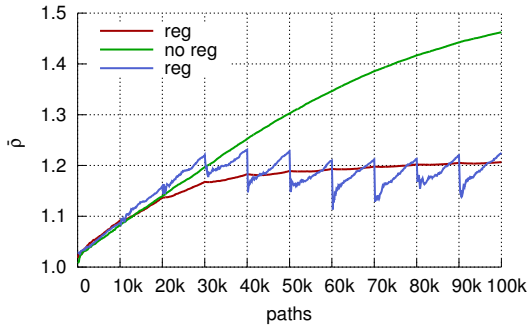
Lastly, Figure 6.32 shows the effect of the different embedding procedure modes on the routing fault-tolerance of the HFR system. Remind that the RM attained the lowest τ-value, the BFM the highest τ-value while BFRM lies somewhere in the middle. After having examined the effect of these different τ-values on the routing stretch and load balancing metrics (see Section 6.3.4), the effect on the fault-tolerance of the system is investigated. As can be seen, the τ-value correlates with the routing success ratio. Generating the embedding in RM attains the highest average success ratio, BFM the lowest while BFRM lies between the two. As a result of a lower tree redundancy, a failed link is less likely to disconnect a large fraction of the spanning trees.
It can be concluded that using multiple embeddings greatly improves routing fault-tolerance towards link and node failures. A backup routing system can be useful when the network is too severely deteriorated for the original routing scheme to work. Also it has been shown that having less tree redundancy, which can be obtained by using a specific graph embedding procedure, leads to more routing fault-tolerance.

6.3.7 Embedding regeneration procedure

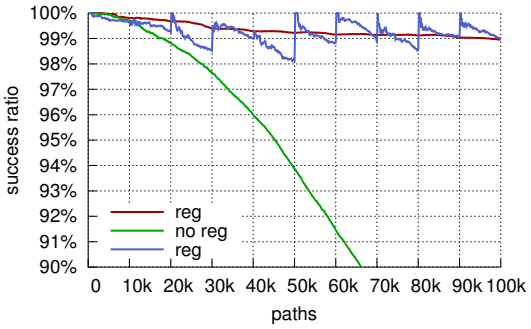
The default HFR system has difficulties coping with a dynamically changing network topology. Therefore an embedding regeneration model has been added to the baseline algorithm as presented in Section 5.5.3. This regeneration mechanism will create new embeddings at runtime which

will replace older deteriorated embeddings. The reason for this is that the more the network is altered, the less the embeddings resemble the current topology. When the embedding resembles the topology less, the stretch and success ratio will deteriorate. Therefore those embeddings that resemble the altered topology the least are swapped out for newer ones to maintain steady routing performance over time. The following experiment has been conducted.

Experiment 11. *For the network R500, random source-destination pairs were generated. Every* 100 *pairs one link was swapped, which means its incident vertices were changed randomly. Every* 10^4 *generated paths* 30% *of the* 10 *embeddings were regenerated. The mechanism used for routing is HFR with an embedding generation procedure in RM with HKE and randomly assigned node keys.*



(a) average stretch $\bar{\rho}$ in function of the number of paths generated



(b) average success ratio in function of the number of paths generated

Figure 6.33: Tree regeneration for graph R500 for 10^5 paths generated, with every 100 paths a link being randomly swapped. Every 10^4 paths, 30% of the embeddings are regenerated. This can be noticed by the saw-like trend in blue. Using no embedding regeneration procedure (green line), makes the routing system vulnerable towards dynamic network behaviour. Contrary to the red line, which depicts the total average value (stretch and success ratio), the blue line resets each value at every embedding regeneration.

When swapping links it is important that the characteristics of the network remain the same. When employing a scale-free network, it would be required that the link swapping should result in another scale-free network with the same properties, this is however non-trivial. As this experiment's only goal is to show the effects of embedding-regeneration, the used benchmark network does not matter. Therefore a random network (R500) was used because the network properties will not be altered by random link swapping. In Figure 6.33 the results are shown. It can be seen that not employing any regeneration scheme leads to a quick incline in average stretch as well as a quick decline in success ratio. When using a regeneration model that regenerates 30% of the embeddings every 100 link deletions/additions it can be noticed that the stretch converges towards a stable point. This allows the network to change indefinitely without the routing mechanism deteriorating. Contrary to the red and green line, the blue line shows the stretch and success ratio after they have been reset at each re-embedding. This better illustrates the true behaviour of the routing scheme in combination with an embedding regeneration procedure.

To conclude, an embedding regeneration procedure may help in withstanding highly dynamic networks with continuously changing topologies. What is left is obtaining a more accurate model of Internet link deletion/addition frequencies, which is however left as future work.

Chapter 7

Conclusions and future work

7.1 Conclusions

In this thesis the application of geometric routing to inter-AS topologies was investigated. After having conducted a literature study, a two-dimensional Euclidean geometric routing system was constructed named Simulated Annealing Label Trees (SALT). Although this routing system achieved good results on unit-disk networks, it performed poorly on scale-free graphs, which are a model for the Internet backbone. After having compared several geometric routing mechanisms found in literature, routing schemes based on graph embeddings that are induced by spanning trees yielded the best results. Therefore these types of systems served as a basis for the development and investigation of more advanced algorithms.

A theoretical geometric routing framework was built to serve as a foundation of the developed family of routing systems called Forest Routing (FR). First a greedy variant was developed named Greedy Forest Routing (GFR) which is based on a graph embedding induced by multiple spanning trees. This variant focuses on attaining a low stretch. As load balancing in geometric routing is still an open research question, a second variant was developed called Load Balanced Forest Routing (LBFR). Packet delivery is guaranteed by the postulation and proof of three theorems. Lastly the most advantageous parts of both variants were united in a scheme called Hybrid Forest Routing (HFR).

A problem in the Border Gateway Protocol (BGP), the current Internet backbone routing mechanism, is load balancing. To be able to balance traffic over network links, routers need to store multiple routes. This causes severe router memory scalability problems. FR avoids this problem as each node is able to naturally distribute traffic on its outgoing links. FR truly matches the large-scale distributed property of the Internet by making sure all routing decision making is done in a localised manner. As a result, the size of the network barely influences its computational forwarding complexity. Furthermore FR is able to attain a low stretch, which means that the average number of hops required for traffic to reach a destination node is nearly equal to that of shortest path routing algorithms such as BGP. Although geometric routing was not perceived to be compatible with load balancing strategies in scientific literature, FR proves that low stretch and load balancing behaviour can go hand in hand. Even more, there is a wide spectrum of possibilities

in combining stretch with load balancing. Moreover, FR can be applied with great success to large scale-free networks, whereas many current geometric routing algorithms focus on smaller unit-disk networks.

As a result of the traffic distribution over a node's outgoing links, fault-tolerance towards node and link failures emerges naturally. Several experiments have shown that FR does not require the storage of additional routing paths, as is the case with BGP, because forwarding decisions happen in an ad-hoc manner. Even at link failure rates as high as 30%, FR is able to attain success rates over 95%. To be able to cope with severe network disconnection scenarios, FR has been equipped with a routing backup system, making it possible to achieve a 100% routing success rate even in a highly deteriorated network.

Furthermore, it has been shown that geometric routing can deal with several types of network dynamics. FR has a routing decision making algorithm capable of coping with varying link bandwidth capacities. Also a routing regeneration procedure has been added to deal with a changing network where links and nodes are continuously removed and added. Though FR is designed to excel on scale-free networks, it has been shown to perform well on other types of networks too.

To conclude, FR focuses on different aspects that a future routing scheme for the Internet backbone requires: (i) scalability: the Internet is vastly growing, and with the surge of Internet of things applications it will continue to increase in size; (ii) low average path length; (iii) natural load balancing behaviour; (iv) high routing success rate, even in case of a severely deteriorated network; (v) flexibility: being able to cope with a wide array of network types and network dynamics. Forest Routing thus advances the state of the art regarding geometric routing.

7.2 Future work

This thesis work lends itself to further investigation. Some interesting future research directions are:

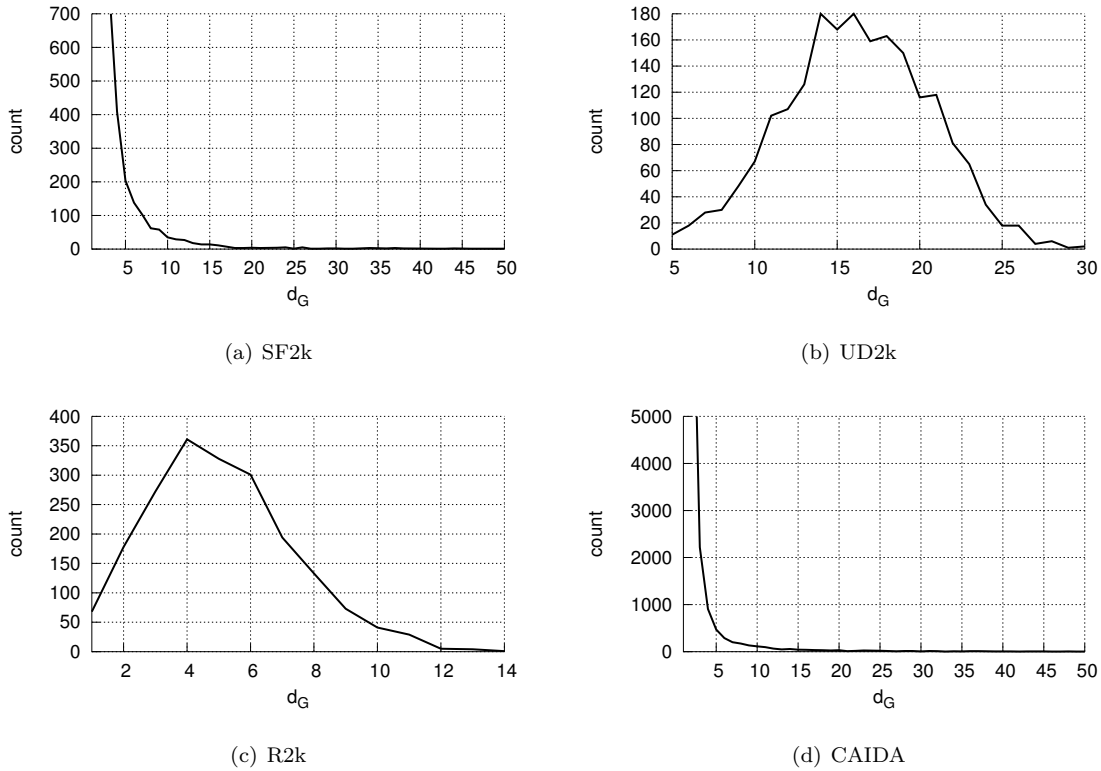
- Making the tree regeneration procedure more adaptive to the network at hand. A system could be developed capable of indicating whether all source-destination host pairs are able to communicate in a dynamic network. Nodes should then dynamically decide whether or not to regenerate a fraction of the embeddings. Also the decision of which embeddings to swap out is a possible research direction. Moreover, algorithms can be developed that are able to restore the greediness of the embedding in case of failures with minimal coordinate reassignment.
- Investigating how to improve FR performance in terms of packet processing when little computational power is available. A possible approach is to only analyse the first packet of a traffic stream, then calculate a hash label and route subsequent packets based on this computationally inexpensive label. Also the impact of using coordinate compression schemes on the computational complexity and storage complexity could be investigated.

- FR could be deployed in a true distributed setting with emulated traffic in order to test how well the simulated behaviour approximates emulation. The stability of the traffic streams generated by the FR forwarding engine with active load balancing could be investigated. This could be combined with examining how telecom providers could implement FR without having to abruptly switch to a novel routing system. A possible way of making this happen is investigating how BGP and FR may coexist in current routers. Furthermore the application of policies to geometric routing can be examined.

Appendix A

Benchmark networks

This appendix shows the properties of the different synthetic and real networks used as a benchmark for testing the routing algorithms. In Figure A.1 the node degree distributions of the different kinds of networks are shown. The scale-free networks are generated by using the algorithm of Barabási & Albert (1999), the random networks are generated with a certain random connection probability and the UDG networks are generated with a certain node density. The CAIDA graph of Hyun *et al.* (2003) is based on real Internet AS-level data.



(a) SF2k

(b) UD2k

(c) R2k

(d) CAIDA

Figure A.1: Node degree d_G distribution for the different types of benchmark networks

Table A.1: Properties of various benchmark graphs: scale-free graphs

	SF500	SF1k	SF2k	SF8k
vertex count	500	1000	2000	8000
edge count	1467	2962	5962	23950
avg. path length	3.267	3.45	3.756	4.168
diameter	6	6	6	7
avg. degree	5.868	5.924	5.962	5.988
avg. clustering coefficient	0.030	0.038	0.017	0.007

Table A.2: Properties of various benchmark graphs: random graphs

	R500	R1k	R2k
vertex count	500	1000	2000
edge count	1291	2427	5048
avg. path length	3.966	4.541	4.859
diameter	7	9	10
avg. degree	2.582	2.427	2.524
avg. clustering coefficient	0.010	0.005	0.002

Table A.3: Properties of various benchmark graphs: unit-disk graphs

	UD500	UD1k	UD2k
vertex count	500	1000	2000
edge count	3889	7804	16163
avg. path length	6.651	9.672	13.480
diameter	17	25	35
avg. degree	15.556	15.608	16.63
avg. clustering coefficient	0.620	0.603	0.605

Table A.4: Properties of the CAIDA graph.

	CAIDA
vertex count	22963
edge count	48436
avg. path length	3.842
diameter	11
avg. degree	4.219
avg. clustering coefficient	0.35

Bibliography

L. A. N. Amaral, A. Scala, M. Barthélémy & H. E. Stanley (2000). Classes of small-world networks. *Proceedings of the National Academy of Sciences*, 97(21):11149–11152.

A.-L. Barabási & R. Albert (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

M. Bastian, S. Heymann & M. Jacomy (2009). Gephi: An open source software for exploring and manipulating networks.

T. Bates, P. Smith & G. Huston (2013). CIDR report for 2 jun 13. CIDR report. URL `http://www.cidr-report.org/as2.0/`.

M. Boguñá, F. Papadopoulos & D. Krioukov (2010). Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1(62).

G. Bouabene, C. Tschudin & G. Leduc (2012). *Greedy routing and virtual coordinates for future networks*.

J. Bruck, J. Gao & A. A. Jiang (2005). MAP: medial axis based geometric routing in sensor networks. In *Proceedings of the 11th annual international conference on Mobile computing and networking*, MobiCom '05, pp. 88–102. ACM, New York, NY, USA. ISBN 1-59593-020-5.

T. Bu, L. Gao & D. Towsley (2004). On characterizing BGP routing table growth. *Comput. Netw.*, 45(1):45–54. ISSN 1389-1286.

T. Bu & D. Towsley (2002). On distinguishing between internet power law topology generators. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pp. 638–647 vol.2. ISSN 0743-166X.

S. Bürkle (2003). *BGP convergence analysis*. Ph.D. thesis, Saarland University.

M. Caesar & J. Rexford (2005). BGP routing policies in ISP networks. *Network, IEEE*, 19(6):5–11. ISSN 0890-8044.

N. Carlsson & D. L. Eager (2007). Non-euclidian geographic routing in wireless networks. *Ad Hoc Netw.*, 5(7):1173–1193. ISSN 1570-8705.

A. Caruso, S. Chessa, S. De & R. Urpi (2005). GPS free coordinate assignment and routing in wireless sensor networks. In *IEEE INFOCOM*, pp. 150–160.

M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn & S. Moon (2007). I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pp. 1–14. ACM, New York, NY, USA. ISBN 978-1-59593-908-1.

H. Chang, R. Govindan, S. Jamin, S. J. Shenker & W. Willinger (2004). Towards capturing representative AS-level internet topologies. *Computer Networks*, 44(6):737 – 755. ISSN 1389-1286.

E. Chávez, N. Mitton & H. Tejeda (2007). Routing in wireless networks with position trees. In E. Kranakis & J. Opatrny, editors, *Ad-Hoc, Mobile, and Wireless Networks*, volume 4686 of *Lecture Notes in Computer Science*, pp. 32–45. Springer Berlin Heidelberg. ISBN 978-3-540-74822-9.

Y.-J. Chi, R. Oliveira & L. Zhang (2008). Cyclops: the AS-level connectivity observatory. *SIGCOMM Comput. Commun. Rev.*, 38(5):5–16. ISSN 0146-4833.

R. Cohen & S. Havlin (2003). Scale-free networks are ultrasmall. *Phys. Rev. Lett.*, 90:058701.

M. Costa, M. Castro, A. Rowstron & P. Key (2004). PIC: practical internet coordinates for distance estimation. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pp. 178–187. ISSN 1063-6927.

D. S. J. D. Couto & R. Morris (2001). Location proxies and intermediate node forwarding for practical geographic forwarding. Technical report, MIT Laboratory for Computer Science.

D. M. Cvetković, M. Doob & H. Sachs (1982). *Spectra of graphs*. VEB Deutscher Verlag der Wissenschaften, Berlin, second edition. Theory and application.

A. Cvetkovski & M. Crovella (2009). Hyperbolic embedding and routing for dynamic graphs. In *INFOCOM 2009, IEEE*, pp. 1647–1655. ISSN 0743-166X.

F. Dabek, R. Cox, F. Kaashoek & R. Morris (2004). Vivaldi: a decentralized network coordinate system. *SIGCOMM Comput. Commun. Rev.*, 34(4):15–26. ISSN 0146-4833.

A. Dhamdhere & C. Dovrolis (2011). Twelve years in the evolution of the internet ecosystem. *Networking, IEEE/ACM Transactions on*, 19(5):1420–1433. ISSN 1063-6692.

A. B. Downey (2001). Evidence for long-tailed distributions in the internet. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW '01, pp. 229–241. ACM, New York, NY, USA. ISBN 1-58113-435-5.

D. Eppstein & M. T. Goodrich (2008). Succinct greedy graph drawing in the hyperbolic plane. *Computing Research Repository (CoRR)*, abs/0806.0341.

M. Faloutsos, P. Faloutsos & C. Faloutsos (1999). On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29(4):251–262. ISSN 0146-4833.

Q. Fang, J. Gao, L. Guibas, V. de Silva & L. Zhang (2005). GLIDER: gradient landmark-based distributed routing for sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pp. 339–350 vol. 1. ISSN 0743-166X.

G. G. Finn (1987). Routing and addressing problems in large metropolitan-scale internetworks. Technical report, Information Sciences Institute, Marina del Rey, California.

R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker & I. Stoica (2005). Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pp. 329–342. USENIX Association, Berkeley, CA, USA.

S. Funke & N. Milosavljevic (2007). Guaranteed-delivery geographic routing under uncertain node locations. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 1244–1252. ISSN 0743-166X.

K. Gadkari, D. Massey & C. Papadopoulos (2011). Dynamics of prefix usage at an edge router. In *Proceedings of the 12th international conference on Passive and active measurement*, PAM'11, pp. 11–20. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-642-19259-3.

J. Gao & L. Zhang (2004). Tradeoffs between stretch factor and load balancing ratio in routing on growth restricted graphs. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, pp. 189–196. ACM, New York, NY, USA. ISBN 1-58113-802-4.

J. Gao & L. Zhang (2009). Trade-offs between stretch factor and load-balancing ratio in routing on growth-restricted graphs. *Parallel and Distributed Systems, IEEE Transactions on*, 20(2):171–179. ISSN 1045-9219.

K.-I. Goh, E. Oh, H. Jeong, B. Kahng & D. Kim (2002). Classification of scale-free networks. *Proceedings of the National Academy of Sciences*, 99(20):12583–12588.

M. T. Goodrich & D. Strash (2009). Succinct greedy geometric routing in the euclidean plane. In Y. Dong, D.-Z. Du & O. Ibarra, editors, *Algorithms and Computation*, volume 5878 of *Lecture Notes in Computer Science*, pp. 781–791. Springer Berlin Heidelberg. ISBN 978-3-642-10630-9.

B. M. Halpern, J. & P. Jakma (2006). Advertising equal cost multipath routes in bgp. RFC page. URL `http://tools.ietf.org/html/draft-bhatia-ecmp-routes-in-bgp-02`.

X. He & H. Zhang (2011). On succinct convex greedy drawing of 3-connected plane graphs. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pp. 1477–1486. SIAM.

Y. Hyun, A. Broido & K. Claffy (2003). Traceroute and BGP AS path incongruities. Technical report, Cooperative Association for Internet Data Analysis (CAIDA).

B. Karp & H. T. Kung (2000). GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pp. 243–254. ACM, New York, NY, USA. ISBN 1-58113-197-6.

B. N. Karp (2000). Geographic routing for wireless networks. Technical report, Harvard University.

J. Kennedy & R. Eberhart (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pp. 1942–1948 vol.4.

S. Kirkpatrick, C. D. Gelatt & M. P. Vecchi (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

J. Kleinberg, A. Slivkins & T. Wexler (2009). Triangulation and embedding using small sets of beacons. *J. ACM*, 56(6):32:1–32:37. ISSN 0004-5411.

R. Kleinberg (2007). Geographic routing using hyperbolic space. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 1902–1909. ISSN 0743-166X.

A. Korman, D. Peleg & Y. Rodeh (2002). Labeling schemes for dynamic tree networks. In H. Alt & A. Ferreira, editors, *STACS 2002*, volume 2285 of *Lecture Notes in Computer Science*, pp. 76–87. Springer Berlin Heidelberg. ISBN 978-3-540-43283-8.

E. Kranakis, H. Singh & J. Urrutia (1999). Compass routing on geometric networks. In *Proc. 11 th Canadian Conference on Computational Geometry*, pp. 51–54. Vancouver.

F. Kuhn, T. Moscibroda & R. Wattenhofer (2004). Unit disk graph approximation. In *Proceedings of the 2004 Joint Workshop on Foundations of Mobile Computing*, DIALM-POMC '04, pp. 17–23. ACM, New York, NY, USA. ISBN 1-58113-921-7.

F. Kuhn, R. Wattenhofer, Y. Zhang & A. Zollinger (2003). Geometric ad-hoc routing: of theory and practice. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, PODC '03, pp. 63–72. ACM, New York, NY, USA. ISBN 1-58113-708-7.

T. Leighton & A. Moitra (2010). Some results on greedy embeddings in metric spaces. *Discrete & Computational Geometry*, 44(3):686–705. ISSN 0179-5376.

B. Leong, B. Liskov & R. Morris (2007). Greedy virtual coordinates for geographic routing. In *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*, pp. 71–80.

F. Li, S. Chen & Y. Wang (2010). Load balancing routing with bounded stretch. *EURASIP J. Wirel. Commun. Netw.*, 2010:10:1–10:16. ISSN 1687-1472.

Z. Li & N. Xiao (2010). Weak greedy routing over graph embedding for wireless sensor networks. *Wireless Sensor Network*, 2(9):683–688.

P. Mahadevan, D. Krioukov, M. Fomenkov, X. Dimitropoulos, k. c. claffy & A. Vahdat (2006). The internet AS-level topology: three data sources and one definitive metric. *SIGCOMM Comput. Commun. Rev.*, 36(1):17–26. ISSN 0146-4833.

Y. Mao, F. Wang, L. Qiu, S. S. Lam & J. M. Smith (2007). S4: Small state and small stretch routing protocol for large wireless sensor networks. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, pp. 8–8. USENIX Association, Berkeley, CA, USA.

J. Mikians, A. Dhamdhere, C. Dovrolis, P. Barlet-Ros & J. Solé-Pareta (2012). Towards a statistical characterization of the interdomain traffic matrix. In R. Bestak, L. Kencl, L. Li, J. Widmer & H. Yin, editors, *NETWORKING 2012*, volume 7290 of *Lecture Notes in Computer Science*, pp. 111–123. Springer Berlin Heidelberg. ISBN 978-3-642-30053-0.

A. Narayanan (2009). A survey on BGP issues and solutions. *CoRR*, abs/0907.4815.

M. E. J. Newman (2003). The structure and function of complex networks. *SIAM Review*, 45(2):167–256.

J. Newsome & D. Song (2003). Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information. pp. 76–88. ACM Press.

R. Oliveira, B. Zhang, D. Pei & L. Zhang (2009). Quantifying path exploration in the internet. *Networking, IEEE/ACM Transactions on*, 17(2):445–458. ISSN 1063-6692.

J. Pan, S. Paul & R. Jain (2011). A survey of the research on future internet architectures. *Communications Magazine, IEEE*, 49(7):26–36. ISSN 0163-6804.

C. H. Papadimitriou & D. Ratajczak (2005). On a conjecture related to geometric routing. *Theoretical Computer Science*, 344(1):3 – 14. ISSN 0304-3975.

F. Papadopoulos, D. Krioukov, M. Bogua & A. Vahdat (2010). Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9. ISSN 0743-166X.

L. Popa, A. Rostamizadeh, R. Karp, C. Papadimitriou & I. Stoica (2007). Balancing traffic load in wireless networks with curveball routing. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '07, pp. 170–179. ACM, New York, NY, USA. ISBN 978-1-59593-684-4.

A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker & I. Stoica (2003). Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, MobiCom '03, pp. 96–108. ACM, New York, NY, USA. ISBN 1-58113-753-2.

Y. Rekhter, T. Li & S. Hares (2006). A border gateway protocol 4 (BGP-4). RFC page. URL `http://www6.ietf.org/rfc/rfc4271`.

S. Sahhaf, W. Tavernier, D. Colle, M. Pickavet & P. Demeester (2013). Link failure recovery technique for greedy routing in the hyperbolic plane. *Comput. Commun.*, 36(6):698–707. ISSN 0140-3664.

R. Sarkar, F. Luo, X. Yin, X. D. Gu & J. Gao (2009). Greedy routing with guaranteed delivery using ricci flows. In *In Proc. of the 8th International Symposium on Information Processing in Sensor Networks (IPSN'09)*.

Y. Shavitt & T. Tankel (2004). Big-bang simulation for embedding network distances in euclidean space. *IEEE/ACM Trans. Netw.*, 12(6):993–1006. ISSN 1063-6692.

G. Siganos, S. L. Tauro & M. Faloutsos (2006). Jellyfish: A conceptual model for the AS internet topology. *Communications and Networks, Journal of*, 8(3):339–350. ISSN 1229-2370.

L. Subramanian, S. Agarwal, J. Rexford & R. Katz (2002). Characterizing the internet hierarchy from multiple vantage points. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pp. 618–627 vol.2. ISSN 0743-166X.

H. Takagi & L. Kleinrock (1984). Optimal transmission ranges for randomly distributed packet radio terminals. *Communications, IEEE Transactions on*, 32(3):246–257. ISSN 0090-6778.

E.-G. Talbi (2009). *Metaheuristics : from design to implementation*, volume 10 of *The Sciences Po series in international relations and political economy*. John Wiley & Sons. ISBN 9780470278581.

M. Tang, H. Chen, G. Zhang & J. Yang (2010). Tree cover based geographic routing with guaranteed delivery. In *Communications (ICC), 2010 IEEE International Conference on*, pp. 1–5. ISSN 1550-3607.

W. Tavernier (2012). *Resilient Future Internet Architectures*. Ph.D. thesis, Ghent University.

M. van Steen (2010). *Graph Theory and Complex Networks: An Introduction*. Maarten van Steen. ISBN 9081540610.

H. Velayos, V. Aleo & G. Karlsson (2004). Load balancing in overlapping wireless LAN cells. In *Communications, 2004 IEEE International Conference on*, volume 7, pp. 3833–3836 Vol.7.

Y. Wang, G. Xie & M.-A. Kaafar (2012). FPC: A self-organized greedy routing in scale-free networks. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pp. 000102–000107. ISSN 1530-1346.

B. Yang, J. Xu, J. Yang & M. Li (2010). Localization algorithm in wireless sensor networks based on semi-supervised manifold learning and its application. *Cluster Computing*, 13(4):435–446. ISSN 1386-7857.

Y. Yu, R. Govindan & D. Estrin (2001). Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks. *Energy*, 463.

T. Zahariadis, D. Papadimitriou, H. Tschofenig, S. Haller, P. Daras, G. D. Stamoulis & M. Hauswirth (2011). The future internet. chapter Towards a future internet architecture, pp. 7–18. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-642-20897-3.

K. Zeng, K. Ren, W. Lou & P. J. Moran (2009). Energy aware efficient geographic routing in lossy wireless sensor networks with environmental energy supply. *Wirel. Netw.*, 15(1):39–51. ISSN 1022-0038.

J. Zhang, Y.-p. Lin, M. Lin, P. Li & S.-w. Zhou (2005). Curve-based greedy routing algorithm for sensor networks. In *Proceedings of the Third international conference on Networking and Mobile Computing*, ICCNMC'05, pp. 1125–1133. Springer-Verlag, Berlin, Heidelberg. ISBN 3-540-28102-9, 978-3-540-28102-3.

J. Zhou, Y. Chen, B. Leong & B. Feng (2010). Practical virtual coordinates for large wireless sensor networks. In *Proceedings of the The 18th IEEE International Conference on Network Protocols*, ICNP '10, pp. 41–51. IEEE Computer Society, Washington, DC, USA. ISBN 978-1-4244-8644-1.

S. Zhou & R. Mondragón (2004a). The rich-club phenomenon in the internet topology. *Communications Letters, IEEE*, 8(3):180–182. ISSN 1089-7798.

S. Zhou & R. J. Mondragón (2003). Towards modelling the internet topology - the interactive growth model. *Computing Research Repository (CoRR)*, cs.NI/0303029.

S. Zhou & R. J. Mondragón (2004b). Accurately modeling the internet topology. *Computing Research Repository (CoRR)*, cs.NI/0402011.

List of Figures

List of Tables

List of Algorithms