Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3150

Underlay aware approach to support quality of service in publish-subscribe systems

Christian Schieberle

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Dr. h. c. Kurt Rothermel
Supervisor:	M. Sc. M. Adnan Tariq
Commenced:	March 1, 2011
Completed:	August 31, 2011
CR-Classification:	C.2.4, C.2.2

Contents

List of Figures

List of Tables

List of Algorithms

1 Abstract

Providing delay-reduced routing is important in publish-subscribe systems where timely delivery of event notifications is a critical factor affecting system operation or user experience. However, common research focused primarily on alleviating false-positives. More recent efforts aim towards quality related issues through adapting the overlay according to subscriber requirements but leaving underlying network characteristics aside.

It is commonly accepted that efficient routing can only be achieved when underlying network characteristics are respected. Even so, incorporating underlay-aware strategies to build low-stretch overlays is not considered in many distributed environments.

This work focuses on solving the problem of establishing an efficient underlay-aware routing mechanism in a content-based publish-subscribe system. In particular, we strive to reduce end-to-end delay among communication partners. Thereby, our contributions are twofold: We will develop a topology inference scheme for unstructured peer-to-peer networks and introduce a routing mechanism reducing overall end-to-end delay among peers. Experimental evaluations will be given for different Internet-like router topologies showing that the approach is capable of modeling an underlay network in an efficient and accurate manner. Furthermore, we will show the positive impact on the stretch of the overlay to outline the concept as a source for efficient event notification delivery in a publish-subscribe environment.

2 Introduction

Operation of distributed applications in dynamic environments is influenced by many factors. In general systems, poor performance may only cause impaired user experience, while in more sophisticated systems like stock exchange applications or environmental monitoring systems, providing timely delivery of event notifications is an essential requirement. Du to the tight coupling of the latter applications to real-world incidents where squandering time may directly result in loss of economic wealth or even physical inviolability.

A crucial factor to performance in that context is reducing end-to-end delay between hosts. It is therefore critical to relate overlay connections to network links exhibiting low latency. Necessary information to implement such a behavior is not at hand without prior efforts. Being able to take network characteristics into account requires inferring of topology information first. Approaches that gain and incorporate such knowledge to a certain degree are said to be *underlay-aware*.

The ratio between the end-to-end latency of the overlay and the network-level delay can be measured and is commonly referred to as *stretch*. It has significant impact on routing cost as it defines the lower bound of the achievable performance.

Dealing with environments where communication partners are dynamically changing and decoupled adds to the complexity of the problem. The publish-subscribe paradigm is a typical representative of such environments as message consumers are addressed indirectly by the content of a message rather than by name or identifier. Participants may express their interests in specific events by issuing subscriptions. Events are emitted by publishers without knowledge of receivers, therefore notifications have to be propagated throughout the network deliberately to reach subscribers.

The expressiveness varies in the different types of subscriptions. Topic-based subscriptions for instance are limited to predefined subjects, while content-based subscriptions permit the definition of attributes and allow filtering on content making them serve a more expressive purpose.

One can categorize the different architectures providing the upper mentioned capabilities of publish-subscribe systems as follows: The common approach implements message brokers. Each participant connects to a broker using a dedicated overlay where event notifications are disseminated solely among brokers. Most recent research however is focusing on techniques that implement peer-to-peer overlays where participants have equal abilities.

In this thesis, we will study the problem of establishing an efficient, underlay-aware routing mechanism for unstructured peer-to-peer environments. The objective is to contribute to quality of service in content-based publish-subscribe systems. We are proposing a distributed topology inference scheme that maintains a peer-to-peer overlay based upon overlapping paths on the router-level. Our second contribution will be the development of a mechanism reducing overall routing cost by decreasing end-to-end delay between peers. Therefore, we will introduce a distributed algorithm building and maintaining an approximate minimum routing cost spanning tree. The routing overlay is used to establish a content-based publish-subscribe system that uses a filter-based approach to propagate event notifications and to reduce false-positives.

We will show experimentally that the inference scheme is able to draw an accurate low-stretch representation of the underlying network. Simulations reveal that achievable stretch is not exclusively related to inference overhead, but also to the amount of dedicated local space. Values close to the optimum can be reached calling for significantly less communication overhead compared to the naive n-by-n approach. Furthermore, simulations show that the routing mechanism is capable of achieving improved results compared to other approaches.

This thesis is structured as follows: After discussing related work in the following chapter, we will define the system model and formulate the problem statement. An approach overview is given in chapter 5, while the process of topology discovery and routing are detailed in chapters 6 and 7. The last chapter concludes the work and offers a prospect of possible future work.

3 Related work

In the following, we will briefly introduce related work that aims towards providing quality of service in terms of routing cost reduction in publish-subscribe systems and general overlays.

3.1 QoS in publish-subscribe systems

Tariq et al. [25] recently proposed an approach to satisfy subscriber-defined delay requirements in a publish-subscribe environment: Subscribers maintain the overlay by establishing connections in a peer-to-peer system. To balance the trade-off among reducing false-positives and improving scalability. The authors distinguish between two types of subscriptions, namely *user-level* and *peer-level* subscriptions. User-level subscriptions represent the original interest, while the peer-level subscriptions define the notifications a peer receives due to forwarding. The peer-level subscription is generated by *spatial indexing* within a decomposed event space. Subscribers satisfy their delay requirements by connecting to peers having tighter requirements and covering subscriptions. They rely on the conjuncture that those peers in turn will connect to proper peers.

While the authors show that their proposed system is robust and scales well, the given approach is not considering underlay characteristics in detail. Since we are focusing on underlay-awareness, the proposed concepts are not implicitly conferrable.

Majumder et al. [18] developed a routing framework that groups subscriptions based on similarity to reduce communication overhead: Each group maintains a dissemination tree with minimized cost. Computing such a tree is a generalization of the *Steiner tree* problem. The authors designed approximation algorithms that use low-stretch spanning trees and prove that the cost is within a poly-logarithmic factor of the optimum.

The dissemination tree is built using a centralized approach and deployed in a broker-based publish/subscribe environment. Our work is paying special attention on decentralized peer-to-peer systems postulating that every participant is capable of fulfilling all tasks involved.

The approach of Jaeger et al. [12] considers broker overlays and seeks a *delivery tree* that spans only brokers that are interested in a specific notification: The authors define the cost for the distribution of a single notification as the sum of processing costs induced by involved brokers and the communication cost of using links between them. The overall cost to distribute all notifications shall be minimized which gives the *Pub/Sub Overlay Optimization Problem (PSOOP)*. The authors show that the corresponding decision problem is NP-complete and develop a *Cost and Interest heuristic* that aims to respect cost and reduce the distance between brokers that consume many identical notifications. Therefore, brokers cache their notifications and compare them to other caches by using *Bloom filters*. Since brokers only have local knowledge, they run a *evaluation and consensus phase* before reconfigurations may occur.

While the approach certainly contributes to quality of service, a broker-based approach again serves different prerequisites.

3.2 QoS in overlays

Beyond the scope of publish-subscribe systems more effort is spent on considering network characteristics in overlay construction. The approaches summarized in this section were contributing to our work by gaining a wider knowledge of established techniques to support quality of service.

Zhu et al. [34, 33] modeled link correlations as linear constraints and proposed a distributed algorithm constructing flow-rate optimized overlays through finding hidden bottlenecks. Overlay links are considered to be correlated if they correspond to paths in the underlay, sharing at least one physical link. The distributed algorithm uses multiple steps. First, it utilizes a probing tool to detect shared bottlenecks based on inter-arrival times of packets among a group of hosts. The groups are partitioned to derive *linear capacity constraints (LCC)* from smaller subgroups while maintaining a global set of constraints. The step of dividing is repeated until all constraints are found. The authors further study two network flow problems, *maximum flow* and *widest path* with the addition of LCC and show that the latter is NP-complete.

In group multicasting every member may multicast to others belonging to the same group. The problem of group multicasting routing with delay constraints (DCGMRP) is addressed by Low et al. [17]: Since the problem is NP-complete, a heuristic algorithm is developed. Each member of the group maintains its own delay-bounded multicast tree. The algorithm constructs a set of *minimum delay multicast trees* using Dijkstra's algorithm as an initial solution. Then the overall solution is iteratively improved by finding the *busiest link* (which is the one with the largest link usage), locating the multicast trees that contain that link, and adjusting the trees to reduce the usage of that link while at the same time satisfying delay constraints.

Parmer et al. [21] show several multicast tree construction algorithms to meet subscriber-defined QoS constraints. The authors define functions that detect nodes that perform best and worst in several latency and route related metrics. They propose algorithms that swap those nodes within the multicast tree to support different subscription policies. They conclude that none of the algorithms can be considered as the best, instead they vary in terms of delay penalty, link stress and cost.

3.3 Topology inference

Kwon et al. [16] addressed the construction of a multicast overlay by exploiting underlying router information: Overlaps among routes from a single source to other group members are used to reduce delay and duplicate packets. They define a process of (shortest) *path matching*, where the overlay tree is partially traversed to determine parents that forward packets originating at the source to its children. This is done when the group of participants changes. The goal is to balance the trade-off among delay and bandwidth. The paths have a significant overlap with the paths determined by routing algorithms, and sharing route prefixes and establishing forwarding lessens the number of identical packet copies sent along underlying links.

The nature of this single-source concept is not targeting towards peer-to-peer overlays. But, we will show that the process of path matching can be extended from trees to general graphs.

A distributed router-level topology inference approach was developed by Jin et al. [13]. It is based on a previously proposed centralized approach, called *Max-Delta*: A server collects traceroute results from a group of hosts. It selects a set of representative paths for the hosts to discover. The goal is to reveal undiscovered topology information while at the same time tracing less routes from each host. In the distributed version each host sends gained traceroute information to all other hosts. Doing that, every host maintains a partially discovered topology. Information is exchanged via an overlay tree that tries to minimize the tree diameter via a node-degree based heuristic. Also, the authors integrate the *Doubletree* approach which is aimed to reduce measurement redundancies using a modified version of traceroute.

The proposed concept of recommending targets for traceroute executions differs from our approach. Even though the paper substantiates our assumption that giving such suggestions during the path matching process will reduce message overhead and will scale even for large environments.

3.4 Approximation algorithms for Minimum Routing Cost Spanning Trees

Among all possible spanning trees of a graph, the *Minimum Routing Cost Spanning Tree* shows the lowest routing cost possible. Finding such a tree is proven to be an NP-hard problem (see the network design problem called *shortest total path length spanning tree* in [14, 11]). Therefore, heuristics have to be applied.

Important related work in that particular context of this thesis is proposed by Wu et al. [29, 28]: The authors develop several algorithms that achieve different approximation ratios of the optimal solution. They utilize a special approximation solution based on a structure referred to as *general stars*. Ratios between 2 and $\frac{4}{3} + \epsilon$ are achieved by seeking specific subtrees called *separators* breaking the overall tree in smaller components. Finding good separators yields lower approximation ratios.

The algorithms are centralized and runtime is increasing rapidly with tighter approximation bounds. This originates from the fact that the main intention of the authors is to prove the existence of several approximation ratios, rather than claiming to derive efficient approaches.

The same authors develop algorithms for generalized problems called the *Optimal Communication Spanning Tree problems* in [28]. Additionally, to an undirected and positively-weighted graph, a requirement is given for every pair of vertices. The cost between two vertices is expressed by the requirement multiplied by the path length between them; the special case where all requirements are set to 1 is the MRCT problem.

Extending our approach by concepts of this paper is proposed in chapter 8.

A recent paper by Campos et al. [6] proposed an MRCT approximation that we will use among other approaches to compare our achieved stretch. The authors claim to provide a solution exhibiting the same routing cost as an MRCT in practice. It is a centralized approach which modifies Prim's well-known MST algorithm: Degree of a node and its adjacent nodes is taken into account additionally to distance information. By composing those factors a *spanning potential* is derived which is used to select a parent node in the spanning tree.

However, to the best of my knowledge there has not been published any distributed algorithm that approximates an MRCT till to the time of the writing of this thesis.

4 System model and problem statement

In the following, we will give a formal description of the system model and formulate the problem statement. An approach overview is given in chapter 5.

4.1 System model

We strive to provide an underlay-aware approach, therefore we will make certain assumptions about the properties of the underlying network. While the approach can easily be extended to asymmetric routes, we will assume that paths on the underlay are stable and symmetric. This is necessary for the path matching routine in the join process of peers.

While it is not problematic to presume that peers have distinguishable identifiers, we will also require that property for routers. This may not reflect reality properly due to the existence of anonymous routers, router aliases and routers with multiple interfaces having different network addresses. We will detail in chapter 6.6.3 how such undesired behavior can be addressed using existing mechanisms.

The underlay network model of routers and peers exhibits a rather natural graph-theoretic formulation. It is given as follows:

Definition 4.1 (Underlay network) *We model the underlay network as an undirected, connected graph with non-negative edge weights. It is given by* $N = (R_N \cup P_N, E_N, \omega_N)$*. The set of vertices is a union of the disjoint finite sets of routers* R_N *and peers* P_N*. The set of links between two arbitrary routers and between routers and peers is defined by* $E_N \subseteq (P_N \times R_N) \cup (R_N \times R_N) \cup (R_N \times P_N)$*. All links are mapped to their corresponding latency value by the weight function* $\omega_N \colon E_N \to \mathbb{R}^{\geq 0}$.

We require that each peer $p \in P_N$ is connected to a single router $r_p \in R_N$ and that the edge (p, r_p) is contained in E_N. We will assume that the last mile delay exhibits low latency and that processing time on a peer to forward messages is negligible. The intuition is that forwarding between peers should not be distinctly more expensive than a direct connection as long as the same underlying routes are involved. The process of forwarding is detailed in chapter 6.

Definition 4.2 (Route) *A route $\rho_N \colon P_N \times P_N \rightarrow \mathcal{P}(E_N)$ is a set of edges that connect a peer s to another peer t via a shortest path on N. It is given by $\rho_N(s,t) = \{(s,r_1),(r_1,r_2),...,(r_{k-1},r_k),(r_k,t)\}$ if it traverses the routers r_i, $i \in \{1,...,k\}$ using only edges from that set. For $s \neq t$, $\rho_N(s,t)$ contains no cycles.*

Definition 4.3 (Delay of a route) *The function $d_N \colon P_N \times P_N \rightarrow \mathbb{R}^{\geq 0}$ returns the end-to-end latency of a route between two given peers. It is defined as $d_N(s,t) = \sum_{e \in \rho_N(s,t)} \omega_N(e)$*

Each peer $s \in P_D$ is able to infer the route to another $t \in P_D$ via executing *traceroute*. The result is the actual path on the underlay network that a packet uses by traveling from a router to another. It is given by $\rho_N(s,t)$. Each peer s may also *ping* a target peer t and receives $d_N(s,t)$ as a result which is given by:

Both functions ρ_N and d_N are used during the topology discovery process. We will now model the discovery overlay D which aims towards giving an accurate representation of the underlay network N based on comparing routes and distances between peers. It emerges from peers inferring routes to other peers and seeking overlaps. The process is discussed in detail in chapter 6. We will only state the preliminaries here.

Definition 4.4 (Topology discovery overlay, neighbor sets) *The topology discovery overlay is modeled by an undirected, positively weighted and connected graph $D = (P_D, E_D, \omega_D)$. The set $P_D \subseteq P_N$ is a subset of peers of the network model. It holds all peers that have joined the discovery overlay.*

Each $p \in P_D$ is aware of a neighbor set $\mathcal{N}_p \subset P_D$ that includes all peers to that it maintains a direct connection. The edge set E_D is the union of all those connections of all peers. It is defined as $E_D = \bigcup_{p \in P_D} \{(p,q) | q \in \mathcal{N}_p\}$. The weight function $\omega_D \colon E_D \rightarrow \mathbb{R}^{\geq 0}$ returns the distance on the underlying network. So for $e = (p,q)$ we set $\omega_D(e) = d_N(p,q)$.

Neighbor sets are common in peer-to-peer overlays and represent in our case the fact that a peer should only have limited knowledge about the whole topology.

Definition 4.5 (Path on the discovery overlay) *Let $u, v \in P_D$ be peers on the discovery overlay. A path on D is given by a set of edges $\{(u,h_1),(h_1,h_2),...,(h_k,v)\}$ if it uses the peers $h_i \in P_D, i \in \{1,...,k\}$ as intermediate hops from u to v in the given order. For every edge (f,g) in a path $g \in \mathcal{N}_f$ holds. We will denote the shortest path from u to v by $\rho_D(u,v)$.*

Definition 4.6 (Distance on the discovery overlay) *The distance $d_D(u,v)$ is given by the sum of edge weights of the shortest path that connects $u, v \in P_D$, so $d_D(u,v) = \sum_{e \in \rho_D(u,v)} \omega_D(e)$*

It is common in overlay networks, especially in peer-to-peer networks, to have one or more designated bootstrapping nodes, sometimes called rendezvous hosts. They provide information to newly joining nodes concerning the mechanism of opting in. We will not go into detail about how such nodes can be identified, but assume that at least one such host exists and that new participants are able to contact it.

The routing overlay R is modeled as an acyclic subgraph of D. All peers of D are spanned in R but with a reduced set of edges.

Definition 4.7 (Routing overlay) *The routing overlay R is an acyclic, connected subgraph of D. We define $R = (P_D, E_R \subseteq E_D, \omega_D)$.*

Definition 4.8 (Path on the routing overlay) *Let u, v be peers in P_D. Then the path $\rho_R(u, v)$ is the unique set of edges that connects that peers on the routing overlay R.*

The distance of paths on the routing overlay R is given by the sum of weights of links between forwarding peers. It is defined as:

Definition 4.9 (Distance on the routing overlay) *Let u, v be peers in P_D. The distance on the routing overlay R is given by the corresponding edge weights in D, so*

$$d_R(u, v) = \sum_{e \in \rho_R(u,v)} \omega_D(e)$$

The stretch on R compared to N is related to the overall routing cost of R.

Definition 4.10 (Overall routing cost) *The overall cost of the routing overlay R is given by*

$$c(R) = \sum_{u,v \in P_D} d_R(u, v)$$

Definition 4.11 (Stretch) *Given a network N and a routing overlay R, we define stretch to be the ratio of the overall routing cost of R over the sum of minimal achievable end-to-end latency on N for all peers in P_D.*

$$stretch_N(R) = \frac{\sum_{u,v \in P_D} d_R(u, v)}{\sum_{u,v \in P_D} d_N(u, v)}$$

We will now define preliminaries for the process of event notification propagation in the content-based publish-subscribe system. We define attributes to have a type, a name and a domain.

Definition 4.12 (Attribute) *An attribute $attr$ is given by a tuple $(type_{attr}, name_{attr}, domain_{attr})$.*

We define *notifications* to be a set of typed attributes having a specific value for each attribute.

Definition 4.13 (Constraints) *A constraint ϕ is a tuple $(type_{\phi}, name_{\phi}, operator_{\phi}, domain_{\phi})$. If an attribute $attr$ matches a constraint ϕ we denote that by $a \prec \phi$.*

A *filter* selects notifications by specifying a set of attributes and a set of constraints on their domain. A subscriber can express its interest using a filter as subscription. If a filter contains multiple constraints for the same attribute, then they are interpreted as a conjunction which means that all constraints must be matched.

So, a notification n matches a filter f if for every constraint ϕ of that filter, there is an attribute a in the notification such that a is matched by ϕ, formally:

$$n \prec f \Leftrightarrow \forall \phi \in f \colon \exists a \in n \colon a \prec \phi$$

4.2 Problem statement

Our aim is to solve the problem of establishing an efficient underlay-aware routing mechanism in an unstructured peer-to-peer environment. In particular, we strive to reduce end-to-end delay. In the context of content-based publish-subscribe systems that implies having to propagate event notifications cost-efficiently through a routing overlay exhibiting low-stretch with respect to the underlying network.

Since we will take network characteristics into account explicitly, we have to derive a discovery overlay first. Driving objective is an accurate representation of the underlying network. The overhead in terms of both local space and communication shall be as small as possible. Furthermore, we need to maintain a routing overlay that exhibits low overall routing cost to achieve a low stretch.

5 Approach overview

The following chapter presents an overview of our solution. We will divide the problem and explain how the resulting sub-problems are conquered separately (see chapter 4.2 for the problem statement).

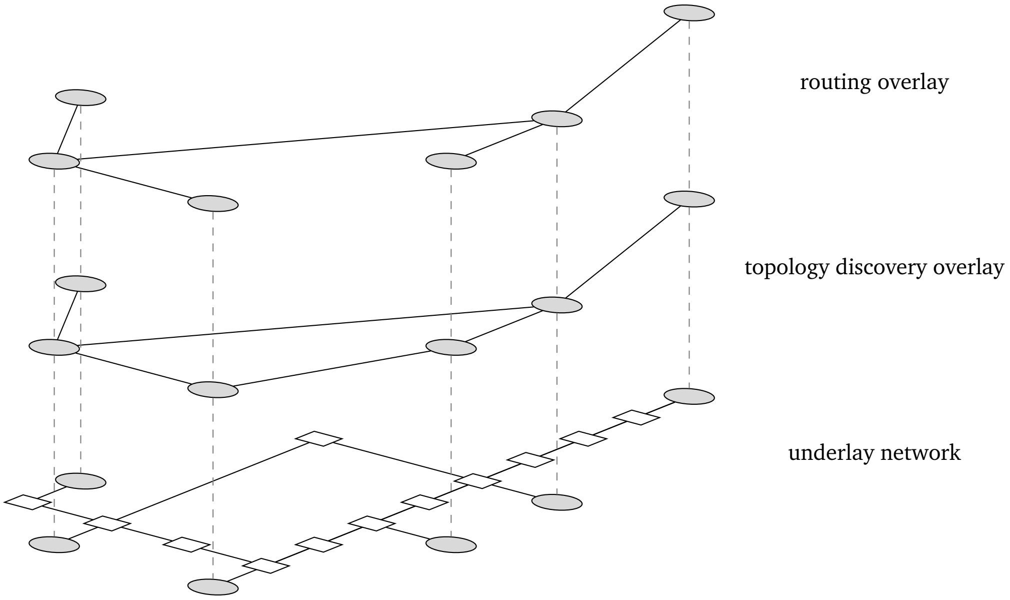


Figure 5.1: Different layers of abstraction having different optimization objectives.

The problem decomposition is depicted in Figure 5.1 and follows along the separation into underlay network, topology discovery overlay and routing overlay, as stated in the system model. We will first focus on the process of topology inference and discovery overlay construction. Subsequently, we will detail the distributed algorithm for the minimum routing cost tree approximation forming the routing overlay. Finally, we will explain the adoption of the publish-subscribe model into this approach.

Topology inference by tracing routes between hosts is costly regarding network operations. An important objective of the discovery process is to keep the amount of the operations small. Nevertheless, we have to derive an accurate representation of the underlay network to build of a low-stretch overlay. Our distributed inferring scheme maintains an unstructured peer-to-peer environment. Neighbor sets of peers are rearranged by applying a path matching process based on router-level information.

The path matching routine was inspired by the *Topology Aware Grouping (TAG)* approach by Kwon et al.[16]. The authors build an acyclic multicast overlay by applying a process that exploits overlaps among underlay routes originated from a single source. Following this idea, we will adapt the process to be used in a peer-to-peer overlay. As a result, neighbor sets are managed to form a general graph, instead of parent-child relations in a tree. We will permit the formation of cycles in the discovery overlay to reach a more accurate representation by considering routes from all peers.

Consider the exemplary situation of a message producer p and the two consumers c_1 and c_2 in Figure 5.2. The two paths (p, c_1) and (p, c_2) have a significant overlap of underlying routers. If p sends a message to c_1 and to c_2 subsequently, the overlapping path will be used twice. Incorporating the path matching routine, p is able to force a change neighborhood sets. After completion of a rearrangement phase, p can reach c_2 by sending a message to c_1 that is delegated to c_2. A positive side-effect may arise from link stress reduction. In the decoupled environment of publish-subscribe systems, every participant may act as a producer (publisher) as well as a consumer (subscriber).

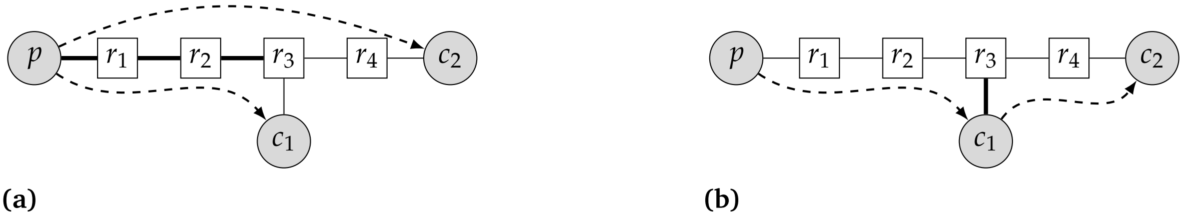


Figure 5.2: Unicast-like message delivery from producer p to consumers c_1 and c_2 **(a)** compared to underlay-aware message forwarding based on router-level path matching **(b)**. Multiple link usage is depicted by bold lines.

When introducing the routing overlay, the main objective is reducing overall end-to-end delay between peers. When considering structures that provide efficient propagation and avoid multiple delivery, minimum routing cost trees (MRCTs) are the optimal solution, if the distance on the discovery overlay is used as the cost function.

Since finding such a structure is proven to be an NP-hard problem, we will develop a heuristic that approximates the MRCT in a distributed manner. We will first derive an alternative way to calculate the overall routing cost. Subsequently, we show by the resulting formula that the contribution of an edge to the overall routing cost is not only determined by its weight. The number of times an edge is used on unique shortest paths on the tree highly influences its contribution. This means in our context that stretch of the routing overlay is influenced by end-to-end delay between peers, and furthermore by the forwarding load of peers.

Our algorithm aims to determine those links having high load forming the future basis of the routing overlay. The first of two phases is called the *voting phase*, where peers are motivated to vote for their neighbors. Lacking deeper knowledge of the peer-to-peer topology, the choice has to be made solely based on latency information. The peers obtaining the most votes are considered to be important for efficient forwarding and therefore incorporated as *relay nodes*. Furthermore, these peers are connected to each other in order to form a subgraph of the routing overlay that we will call the *core*. In a second phase, all other peers are connecting to the core through shortest paths on the discovery overlay.

Considering the publish-subscribe model, we will establish an event routing approach that benefits from the shortest path connections in the routing overlay. As a consequence event notifications are only propagated in the direction of subscribers having claimed interest before. The routes to be taken are determined by a reverse path mechanism. A reverse path is established once a subscription is received, by installing a filter on the receiving peer.

6 Topology discovery

In the following, we will describe in detail how the process of network topology inference is designed. The overall aim is to build a peer-to-peer overlay that represents an underlying router-level topology in an accurate manner.

We presume the utilization of traceroute-like network diagnostic tools to infer topology characteristics. While this enables us to build a precise model of the underlay, the process of gaining this information is costly. It involves sending multiple ICMP packets with increasing time-to-live to determine intermediate routers on the path to a target host. The routers decrease the time-to-live of the packet and will respond with an error message once the value reaches zero. The source host is able to construct the list of intermediate routers based on that responses. An obvious objective for the inference scheme is to avoid as many expensive network operations as possible.

The driving mechanism of the discovery process is a distributed underlay path matching routine.

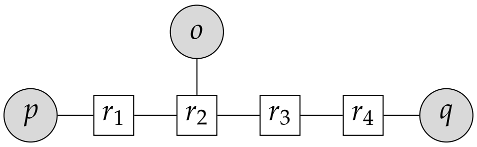


Figure 6.1: Simple underlay network. Which connections need to be maintained in a peer-to-peer overlay to accurately represent this situation?

Assume the simplified situation in Figure 6.1: The three peers o, p and q connected through the routers r_1 to r_4. The peers could simply form a mesh where each peer accepts all other peers as neighbors. While this might result in a representation exhibiting low-stretch, such a solution is obviously not scalable. The objective question is: Which overlay edges can be abandoned compared to a mesh overlay, while still maintaining an accurate representation?

Our system model presumes last-mile delay being low and processing time on a peer being negligible. Under these assumptions, the question is answered for the example given in Figure 6.1 as follows: If p wants to send a message to q, it will first send a message to o using the routers r_1 and r_2. The message is subsequently forwarded from o to q via the routers r_2, r_3 and r_4.

In a situation where all peers can be consumers and producers of messages at the same time, the neighborhood must be set as follows, to keep the number of overlay edges as low as possible: $\mathcal{N}_p = \{o\}$, $\mathcal{N}_o = \{p, q\}$ and $\mathcal{N}_q = \{o\}$. This guarantees not only a space-efficient representation, but also maintains a low end-to-end delay for all peers.

As a nice side-effect the links stress in the topology discovery overlay is reduced. If for instance, q intends to send the same message to o and p, the forwarding mechanism via o guarantees that all links except of o's connection to its own router are used only once.

Obviously, there is more overhead involved in the management of a peer-to-peer overlay than maintaining a set of neighbors, for instance, maintaining socket connections to neighbors. But in the following chapters, we will allow ourselves a simplification and refer to this problem only in the context of local space reduction, since the additional overhead is directly related to the size of the neighbor sets.

In the following, we will give a formal description of the underlay path matching process and the implementation for building and maintaining the discovery overlay. Following that, complexity analysis and evaluations are given below.

6.1 Prefixes

We previously presumed that a peer p stores the resulting route of $\rho_N(p, q)$ to all its neighbors $q \in \mathcal{N}_p$. That information enables p to rearrange its neighbor set once it receives a join request from another peer. As mentioned before, the size of the set is a crucial factor to accuracy and space efficiency of the overall process.

Before the detailed maintenance of the neighbor sets is discussed, we will define the formal foundation behind the rationale of the process. Therefore, we define *prefixes* that represent the shared routes.

Definition 6.1 (Prefix) *Let p, o, q be peers in P_D ($p \neq o \neq q$). Let $\rho_N(p, o) = \{(p, r_{o,1}), ..., (r_{o,k}, o)\}$ and $\rho_N(p, q) = \{(p, r_{q,1}), ..., (r_{q,k+l}, q)\}$ be shortest paths on the network N ($k \geq 1$, $l \geq 0$). We call o a prefix of q with respect to p if $r_{o,i} = r_{q,i}$, $\forall i \in \{1, ..., k\}$. We will denote such a situation by $o \prec^p q$.*

Consider again the situation in Figure 6.1. In that situation, $\rho_N(p, o)$ is given by $\{(p, r_1), (r_1, r_2), (r_2, o)\}$, and $\rho_N(p, q)$ is $\{(p, r_1), (r_1, r_2), (r_2, r_3), (r_3, r_4), (r_4, q)\}$. Obviously, the paths overlap on the links from p via the routers r_1 and r_2. Since additionally, r_2 is the router that connects o to the network, we say o is a prefix of q with respect to p, denoted by $o \prec^p q$.

Furthermore, if $o \prec^p q$ holds, we want to change the neighbor sets such that o forwards messages from p to q. Formally expressed we strive to have $\{(p, o), (o, q)\} \subseteq E_D$, and $(p, q) \notin E_D$. We will now generalize the process to achieve that.

6.2 Joining the overlay

Assume the non-trivial case where there are already peers in the discovery overlay (i.e. $P_D \neq \emptyset$). A peer n ($n \in P_N$, but $n \notin P_D$) wants to opt in, so it contacts a rendezvous node as described in the system model. We presume that the rendezvous node responds with the name of a randomly chosen peer $j \in P_D$. The peer n will now contact j with a join request.

To perform the path matching process, j need to trace the route to n. Knowing the route to n enables j to perform the path matching process and compare the newly discovered route with routes to its neighbors. There are three mutually exclusive conditions leading to different actions. For a better understanding, an exemplary join process for a simple unit-weight topology is given in Figure 6.2.

The first condition handles the case, where j knows a neighbor a that is connected to a router that lies on the underlay path from j to the newly joining peer n:

(6.1) : $\exists a \in \mathcal{N}_j \colon (a \prec^j n) \wedge |\rho_N(j,n)| > |\rho_N(j,a)|$

In that case, n and j will not become neighbors, since they want to benefit from the shared route, so j suggests n to join with a instead. Consequently, n will send a join request to a. An example for that situation is depicted in Figure 6.2(c).

We will later prove that there is at most one $a \in \mathcal{N}_j$ that fulfills that condition. In other words, j can't know any node $a' \in \mathcal{N}_j$ ($a' \neq a$) that had a shorter prefix of n, having less hops.

If $a \prec^j n$ holds, but the second term of the condition does not, the situation is covered by the following condition.

(6.2) : $\exists a \in \mathcal{N}_j \colon n \prec^j a$

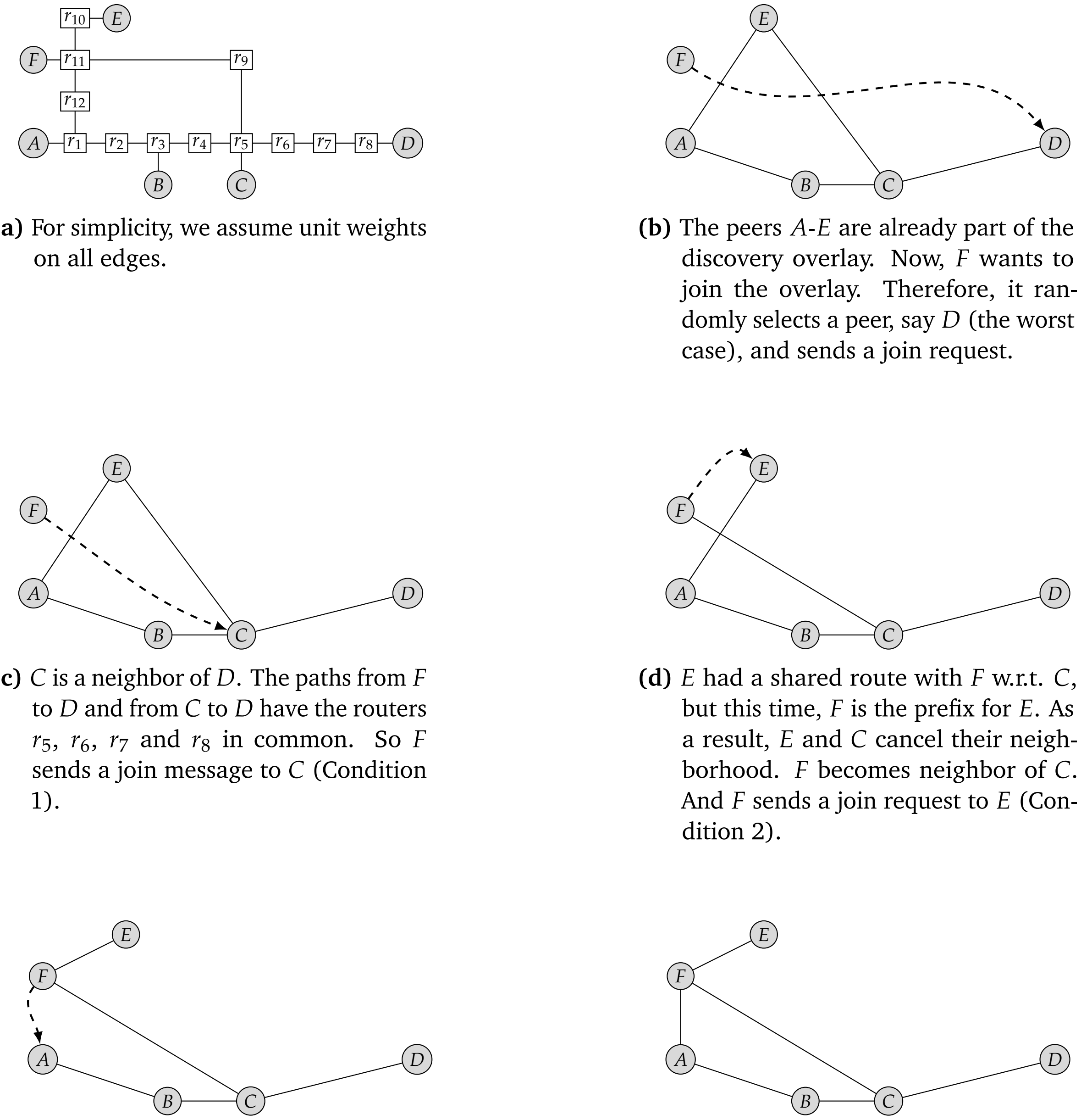
The peers n and j become neighbors while a and j cancel their current neighborhood relation. The peer n additionally sends a join request to a. Metaphorically speaking, n somehow *adopts* a from j.

An example for that situation is depicted in Figure 6.2(d) and 6.2(e).

In the emerging situation, n acts as forwarding peer between j and a. We achieve the desired result where (a,j) is removed from E_D, while the edge (n,j) is added. The connection between a and n is established in the separate join process.

If the above conditions did not hold, n and j become neighbors.

(6.3) : neither (6.1) nor (6.2) evaluate to true



(a) For simplicity, we assume unit weights on all edges.

(b) The peers A-E are already part of the discovery overlay. Now, F wants to join the overlay. Therefore, it randomly selects a peer, say D (the worst case), and sends a join request.

(c) C is a neighbor of D. The paths from F to D and from C to D have the routers r_5, r_6, r_7 and r_8 in common. So F sends a join message to C (Condition 1).

(d) E had a shared route with F w.r.t. C, but this time, F is the prefix for E. As a result, E and C cancel their neighborhood. F becomes neighbor of C. And F sends a join request to E (Condition 2).

(e) F shares a prefix with E's former neighbor A. So, E and A cancelled their neighborhood. F accepts E as a neighbor. F sends a join request to A (Condition 2).

(f) F just joined A which has only one other neighbor, B. Since neither F is a prefix of B, nor B is a prefix of A, A accepts F as neighbor and vice versa (Condition 3). At the end, the overall link stress is minimized successfully.

Figure 6.2: An exemplary join process of a peer F joining an established discovery overlay.

That last condition evaluates to true, if there could not be any path matched on the underlay between n and j, or any of j's neighbors. This is not necessarily an undesired behavior, but may also be an indicator that j was not a good choice for n. We will cover that situation later.

The last situation is depicted in Figure 6.2(f).

In the simulations, we will additionally restrict the maximum size of the neighbor sets. In that case, a peer will only be added to the neighbor set if it is closer than the currently farthest neighbor in the set which it will replace. If adding n to $\mathcal{N}_j$ failed because the maximum size of the set is reached and j is not accepted, n is allowed to request a new peer using bootstrapping mechanism.

We will now prove that if there is an a in $\mathcal{N}_j$ that is a prefix of n then j cannot know any better prefix $a' \in \mathcal{N}_j$ $(a' \neq a)$ such that a' is a possibly shorter prefix of n.

Lemma 6.1 *If n joins j and there is an $a \in \mathcal{N}_j \colon a \prec^j n$ then there cannot be any $a' \in \mathcal{N}_j \colon (a' \prec^j n) \wedge |\rho_N(j, a')| < |\rho_N(j, a)|$.*

Proof 6.1 *Assume, that not only $a \in \mathcal{N}_j$ but there is also an $a' \in \mathcal{N}_j \colon (a' \prec^j n) \wedge |\rho_N(j, a')| < |\rho_N(j, a)|$. In that case, a' would be in the set $\mathcal{N}_j$. Also, a must have joined j to be put in $\mathcal{N}_j$. But, then a path matching between a' and a would have happened before. During that path matching, condition 6.2 would have evaluated to true, and a would have joined a' instead, and implies that $a \notin \mathcal{N}_j$. We reached a contradiction.*

We will introduce in the following two strategies designed to cover the situation where no path matching was possible on condition 6.3. The strategies are called *limited flooding* and *random walk* and will be evaluated in combination with the overall join process subsequently.

6.3 Limited flooding strategy

Assume the situation, where a peer n opted in the overlay but unluckily ends up at a position that not accurately represents the underlay network condition. As described, this happens if no successful path matching was possible during the join process.

We propose a strategy covering that situation as follows: Currently, n is connected to a node j. Since path matching failed, n will now perform a less expensive underlay-aware matching routine by starting to measure its distance to peers in the neighborhood of j.

The distance measurement is a valuable indicator whether there are better nearby choices than j. Even if the neighbors of j did not have overlapping routes, other peers that are multiple hops away on the overlay, may perform better. Since, n doesn't know these peers, it sends a designated message to j: That message contains the tuple (n, d, ttl), where d is the underlay distance of n to its nearest neighbor. That message can afterwards be forwarded by j to its neighbors. Subsequently, the message can be forward again while decreasing the time-to-live. Forwarding is stopped, once a value of 0 is reached.

We introduce the system parameter λ, which we call the *locality factor*, as it determines the degree of locality for improvements to n. Peers that receive the previously described tuple are able to guess, whether they would be an improvement for n. They simply evaluate whether the following inequality holds:

$$(6.4)\quad d_N(a,n) < \lambda * d$$

If that evaluates to true, the peer a knows that it would be an improvement to n, so it acts as if it would have received a join request from n. Using a higher λ, increases the probability that even distant peers are sending a join request to n. During that process the path matching is executed. So we implemented a lighter version of flooding the neighborhood by using only distance measurements, and still get the benefit of possibly overlapping paths of physically close peers after execution of a path matching.

Even if no overlaps exist, once again the condition 6.3 may hold and a and n become neighbors anyway. This is at least an improvement in delay, although paths must not necessarily overlap. The flooding can be regarded as a mechanism that pushes the peer into the right direction based on distance measurements.

One drawback of that strategy is that the mechanism may still causes many path matching processes. Especially if λ is set to a high value, many peers may consider themselves an improvement for n and will cause traceroute executions. We are able to overcome that situation by being more selective when evaluating the inequality 6.4. This can either be done by a tighter setting for λ, or by limiting the number of neighbors that are flooded. This is the key factor in the second strategy called *random walk*.

6.4 Random walk strategy

As described, *limited flooding* may cause a considerable amount of distance measurements, join messages and traceroute executions in situations, where many hops are necessary on the overlay. It might cost some hops on the overlay until a peer reaches a decent position, representing the underlay accurately.

The idea behind *random walk* originates in the assumption, that it might not be necessary to flood the whole neighborhood on every hop. Instead, only a subset of neighbors is selected. Therefore, we introduce the *selectivity factor* σ representing the chance of each neighborhood peer to be chosen for flooding ($0 < \sigma < 1$).

A tight setting for σ lessens the amount of flooding per hop, but raises the probability that a peer is missed, although might have meant an improvement. While discussing the evaluations later on, we will see that a setting lower as 0.375 performs poorly. Interestingly, taking every second neighbor into account, performs reasonable, while reducing the amount of join processes and the number of expensive network operations involved.

6.5 Leaving the overlay and node failure

In the following we will describe, how different situations arising are handled, when a peer p is dropped from the discover overlay through node failure, or intentionally leaves it.

We will consider graceful leave first: Before p disconnects, it can easily determine which of its neighbors q should send a join message to another neighbor q'. For every situation in which $p \prec^q q'$ holds, it motivates q to join q'. This restores the underlay-awareness of the overlay.

When p is disconnected caused by node failure, there are two cases that can easily be managed: If a neighbor $q \in \mathcal{N}_p$ is disconnected from the overlay because p fails, it has to rejoin the network by contacting a rendezvous node. If q is not disconnected, it will restore underlay-awareness by running one of the improvement strategies described before.

6.6 Evaluations

We will give a brief complexity analysis of the path matching process and will subsequently provide evaluations.

6.6.1 Complexity analysis

Space complexity is crucial to accuracy of the derived discovery overlay and depends on the system parameter $max_{\mathcal{N}}$: A peer p has to store the routes $\rho_N(p,q)$ and their length $d_N(p,q)$ to its neighbors $q \in \mathcal{N}_p$. This is necessary to process the underlay path matching process when receiving a join request. The number of stored routes is limited by that parameter, so each peer stores at most $max_{\mathcal{N}}$ routes that consist of a number of router identifiers.

It is difficult to give bounds on message complexity, as a join process may lead to other join processes. The number of traceroute executions is proportional to the number of join messages, since at most two of them are executed during that process. We will therefore examine the number of join messages and traceroute execution via simulations.

6.6.2 Simulations

Experimental setup

In topology generation there exist two major model categories, namely *domain-level* and *router-level* topologies. Both of them are represented as graphs, but with different meanings of the components: In the domain-level model, nodes depict domains and edges represent inter-domain connections, while on the router-level, nodes refer to routers and edges represent a one hop connection between them.

Topology generation is an ongoing research topic, so there naturally exist multiple concepts. Caused by the sheer size, lack of persistence and system administrators' efforts to obscure routing behavior within a domain, it is very difficult, if not impossible, to verify those models. We can at least try to judge their eligibility. There's little doubt that the Internet has a significant degree of hierarchy on both levels: at the router level this is mainly induced by backbones and at the domain level service providers are broken into tiers [24]. That is one reason, why AS-level topologies mainly focus on degree distribution (for which a power-law was detected [8, 23]), while imitating hierarchy and locality is the focus of router-level topologies.

For evaluations of our approach, only the latter models are aplicable. To acknowledge diversity, we will generate two models that follow completely different concepts. We aim to determine whether hierarchy influences the overall accuracy and efficiency of our approach, as well if different locality affect the quality of the overall process.

We will use BRITE [19] to generate a non-hierarchical Waxman topology [26], and we will utilize GT-ITM [32] to generate a hierarchical transit-stub topology [31] with different domain-roles.

Waxman topologies are commonly used models for generation of random networks. The nodes are placed at random positions in a two-dimensional grid. All possible pairs of nodes are considered and the decision whether a link should exist between two nodes is made according to a probability function that models locality. One of the parameters, α, increases the general probability of edges between any two nodes. The second parameter, β, yields the ratio of long edges to short edges in the overall topology. The number of links added per node, m, determines the overall number of edges of the topology. We will use the standard settings $\alpha = 0.15$ and $\beta = 0.2$, but increase the number of links per node ($m = 4$).

The transit-stub model is used to represent a hierarchical topology. It also places routers in a two-dimenstional space. One or more transit-domains of routers are inter-connected and a number of stub-domains is connected to the transit domains. Both domain types are populated with a number of routers that are inter-connected based on given probabilities for the domain type [31, 30]. Our topology contains 4 transit-domains, each consisting of 12 routers and a chance of 60% that a link between any two routers exists in a transit domain. The 12 stub domains are fully linked, each containing 21 routers.

With that setting, both topologies have exactly 1008 routers. The characteristics of the generated topologies are given in table 6.1.

Characteristic	Transit-stub model	Waxman model
Number of routers	1008	1008
Number of links	9436	4032
Average node degree	9.36	4

Table 6.1: Characteristics of generated router-level topologies

We will run experiments using the *PeerSim* [20] peer-to-peer simulator. If not otherwise stated, we will randomly connect 150 peers to the router-level topology models. In transit-stub topologies, peers are only connected to stub domains. Consistent to our assumptions in the system model, we will set the last-mile latency between a peer and a router to 0. This will also give us the possibility to better judge achieved stretch values, since it is at least possible in theory to reach a stretch of 1.

We already introduced three system parameters that are designed to control certain trade-offs among accuracy and efficiency. We will run simulations with different values for those parameters, to determine reasonable settings. The parameters are:

- maximum number of neighbors $max_{\mathcal{N}}$ - used in both strategies
- locality factor λ - used in both strategies
- selectivity factor σ - used in the random walk strategy only

When discussing the results for the *random walk* strategy, we will reduce the number of variables. Therefore, the locality factor is set to a fixed value, namely $\lambda = 1.5$. The intuition is results showing reasonable results in the limited flooding strategy while exhibiting a fair trade-off among overhead and locality.

Metrics

To observe accuracy, we are interested in the coverage of routers and links. The metrics are given by:

Definition 6.2 (Router coverage, link coverage) *Given a network N and a topology discovery overlay D, the router coverage of N by D is the ratio* $\frac{|R_D|}{|R_N|} * 100$ *where* $R_D \subseteq R_N$ *is the set of routers that were discovered and used when constructing D. Respectively, the link coverage of N by D is given by* $\frac{|E_D|}{|E_N|} * 100$.

The most important metric for accuracy is stretch with respect to the underlay network, it is given by:

Definition 6.3 (Stretch of the discovery overlay) *Given a network N and a topology discovery overlay D we define stretch to be the ratio of the shortest path length on D and the network delay on N between any two peers, which is given by* $\frac{\sum_{u,v \in P_D} d_D(u,v)}{\sum_{u,v \in P_D} d_N(u,v)}$.

To determine the efficiency of the process, we will also observe the average number of neighbors that are connected to a peer, as well as the number of join messages that were sent per peer. Furthermore, the overall amount of traceroute executions compared to a naive n-by-n approach.

As described before, we expect reduction in link stress as a side-effect of message forwarding. We define link stress as the number of times an underlay link is used when a peer is sending to a set of targets. To derive sample values for link stress, we assume for the experiments that peer identifiers are globally known. Each peer p selects k randomly chosen other peers, where $k \in [1, |P_D|[$. We will plot the ratio of average usage of a link when using unicast-like, but shortest path based routing would be used in N compared to a forwarding mechanism that uses shortest paths in the overlay D.

Result discussion

As mentioned before, there is a trade-off between the achievable accuracy and the overhead regarding local space and messages. We have introduced the underlay path matching process and two strategies to cover the situation where the matching process fails.

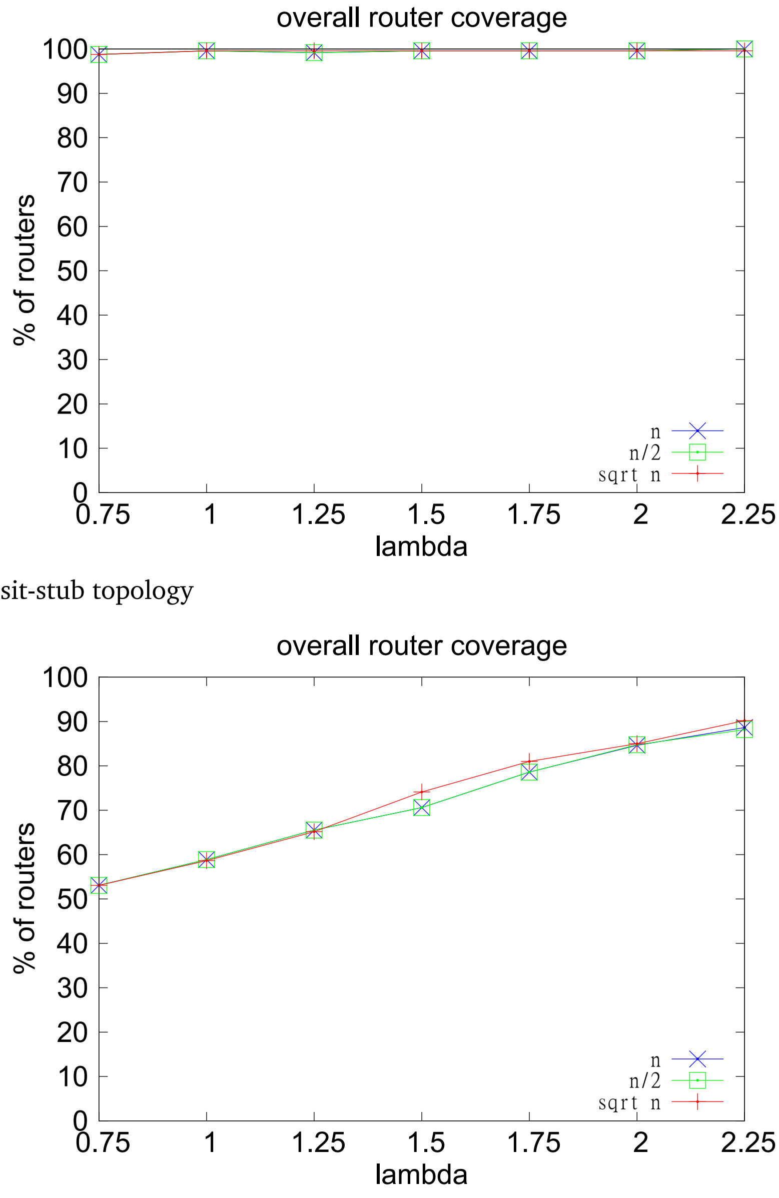
We will first focus on the accuracy of the *limited flooding* strategy.

Accuracy of limited flooding

It is evident from Figure 6.3(a) that overall router coverage for hierarchical transit-stub topologies is very high for all values the locality factor λ takes. In the Waxman topology the router coverage starts low at barely over 50% and needs an increase of λ to grow close to 90% for different settings of maximum allowed neighbors (Figure 6.3(b)).

Nevertheless, the overall router coverage does not seem to be heavily influenced by $max_{\mathcal{N}}$. That might stem from the fact that the coverage is increasing as new routers are discovered by using traceroute during the join process, and is not determined by the current values of the dynamically changing neighbor sets. Increasing coverage is an indicator that the choices made to gain more knowledge are reasonable, asnew routers are discovered.

Link coverage for the Transit-stub topology needs increasing λ to reach values over 90%. Similar to the router coverage, the link coverage for the Waxman topology (figure 6.4b) is improving with higher λ-values. An explanation for that behavior lies in the building process of that type of topology. There is basically a random nature opposed to the hierarchical structure



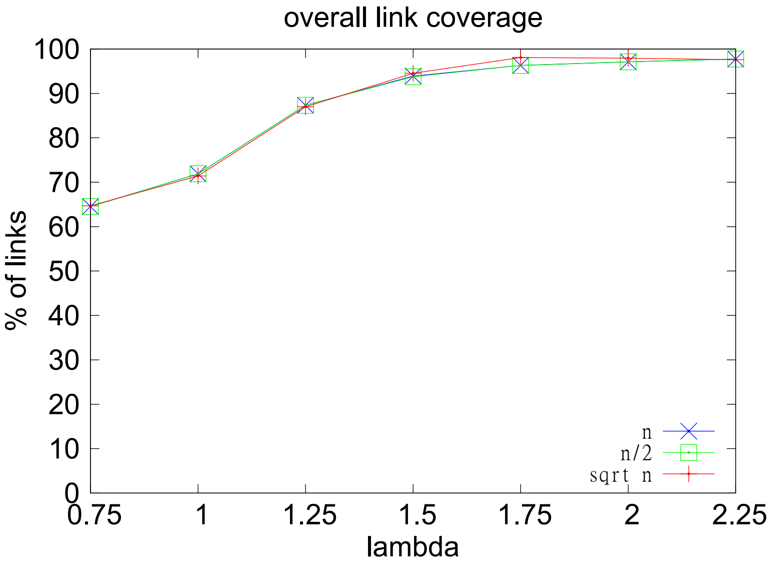
(a) Transit-stub topology

(b) Waxman topology

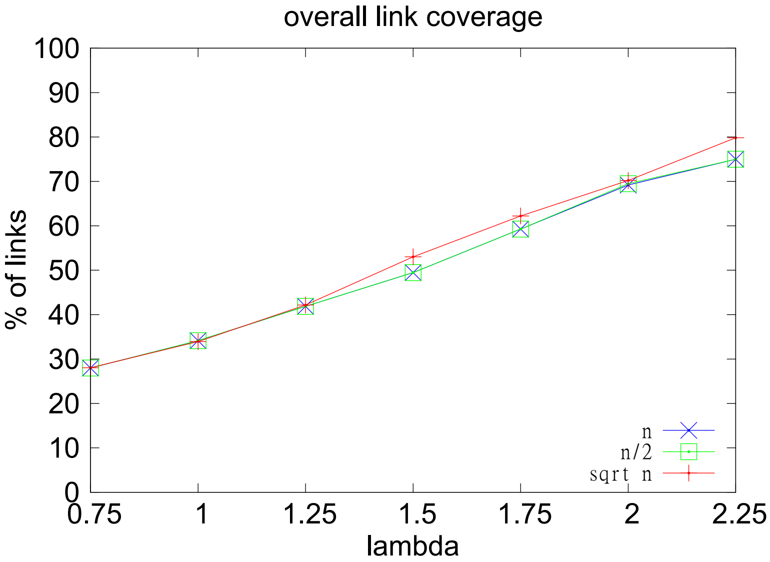
Figure 6.3: Router coverage for the limited flooding strategy for different λ-values. The maximum number of neighbors is limited to n, $\frac{n}{2}$ and $\sqrt{n}$.

of the transit-stub topologies. Since Waxman topologies are not channeled through a transit domain, many paths between distant peers may exist.

However, while coverage of links and routers determines the effect of greediness of the process at a glance, it is apparent from Figures 6.5a and 6.5b that coverage is not a good indicator for



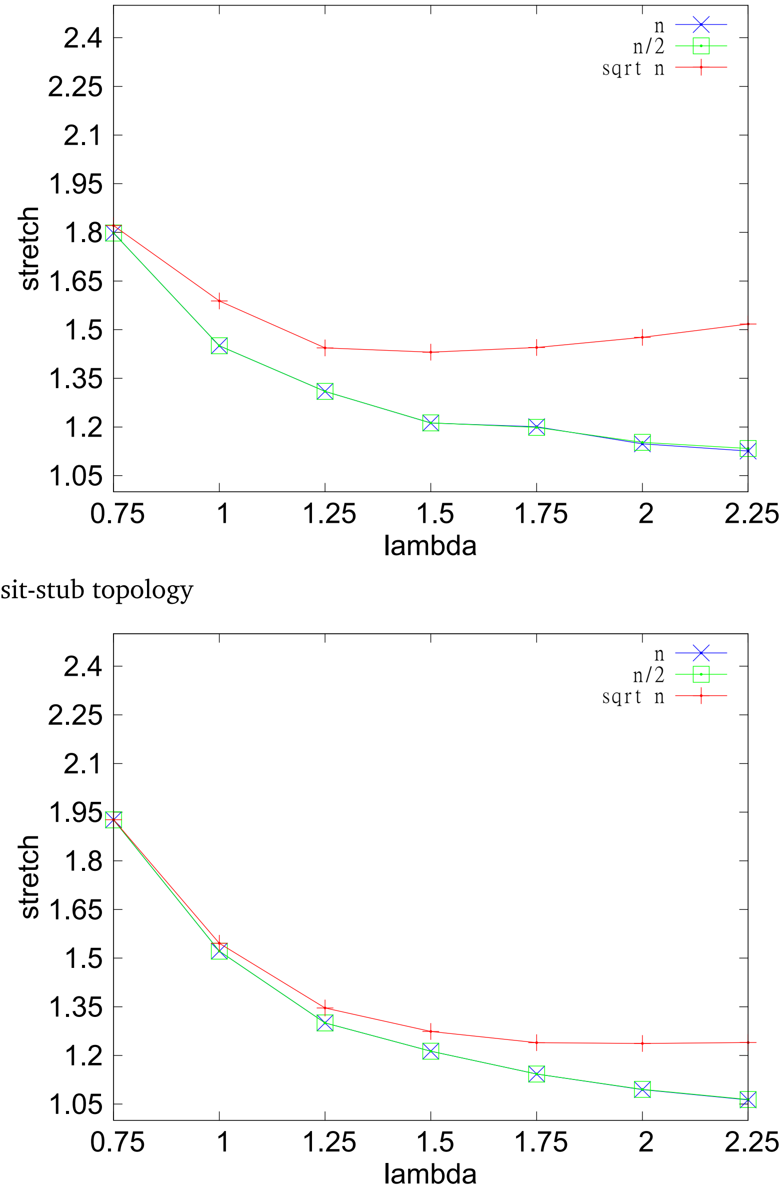
(a) Transit-stub topology



(b) Waxman topology

Figure 6.4: Link coverage for the limited flooding strategy for different λ-values. The maximum number of neighbors is limited to n, $\frac{n}{2}$ and $\sqrt{n}$.

accuracy. For example when at most $\sqrt{n}$ neighbors are allowed, the router coverage increases for $\lambda \geq 1.75$ while stretch stalls or even slightly increases. In the transit-stub topology the stretch underlies the same effect when the size of neighbor set is limited. Accordingly, the



(a) Transit-stub topology

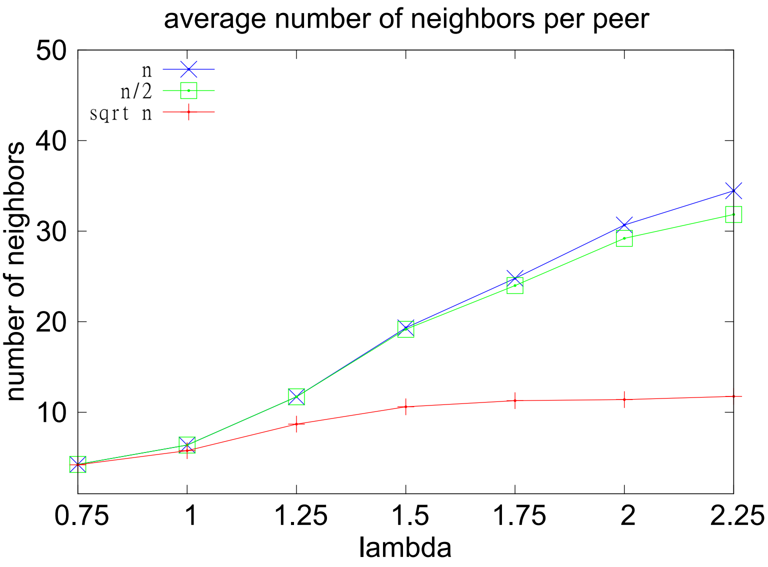
(b) Waxman topology

Figure 6.5: Stretch for the limited flooding strategy for different λ-values. The maximum number of neighbors is limited to n, $\frac{n}{2}$ and $\sqrt{n}$.

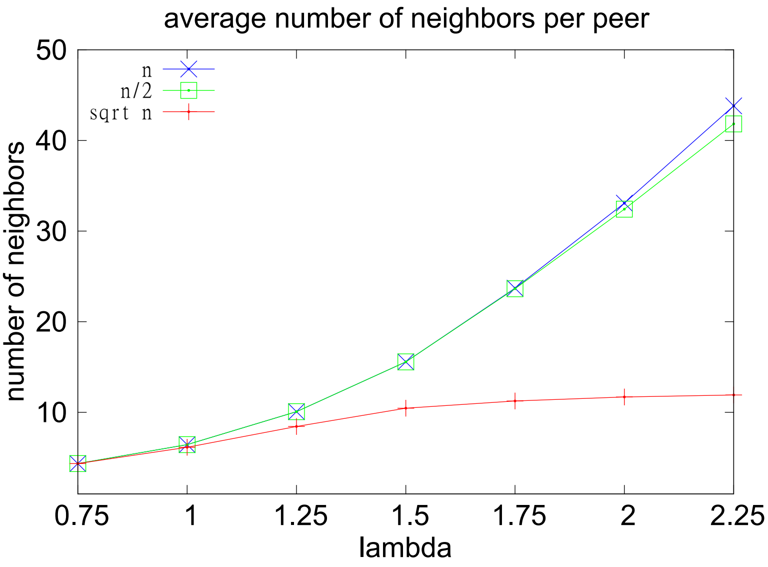
achievable stretch highly depends on $max_{\mathcal{N}}$ and λ. We will now observe, how these values influence the efficiency of the approach.

Efficiency of limited flooding

When investigating the induced overhead, we are interested in average space used by neighbor sets and message overhead caused by join messages on the one hand and traceroute messages on the other hand. We will discuss the results for the different system parameters.

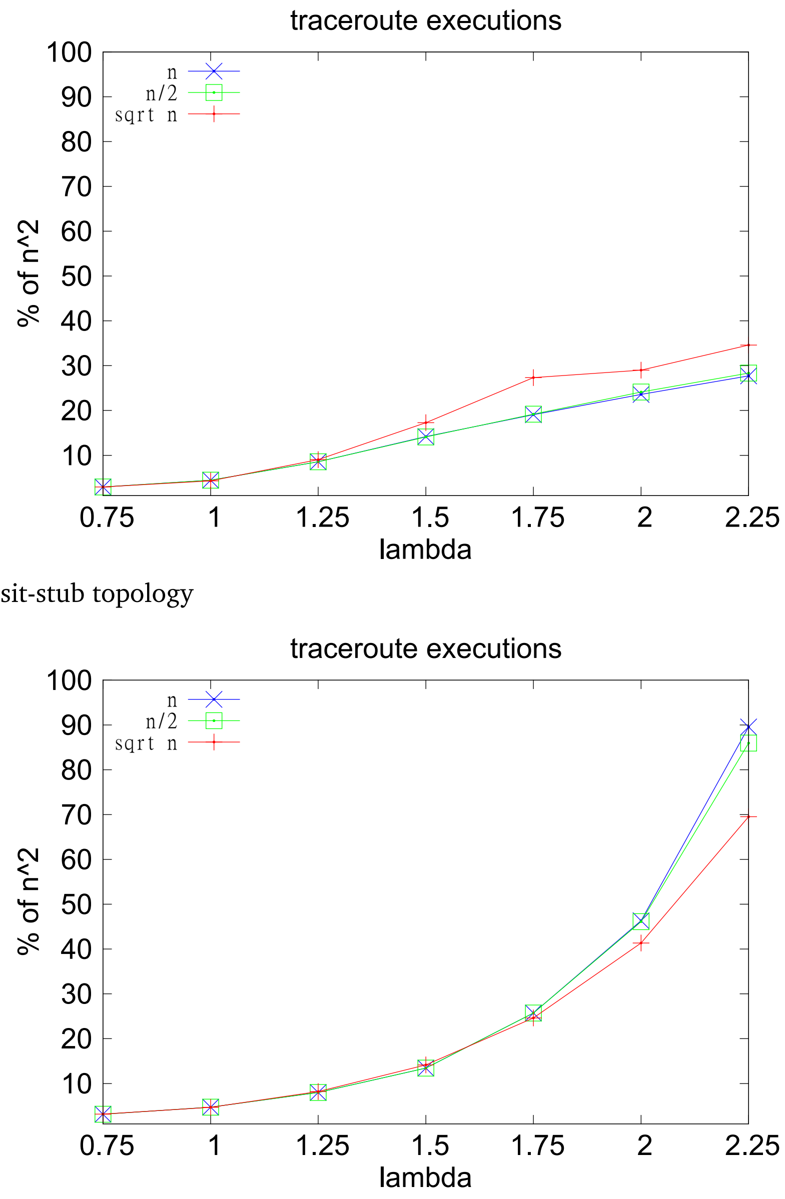


(a) Transit-stub topology



(b) Waxman topology

Figure 6.6: Average number of neighbors on the limited flooding strategy for different λ-values. The maximum number of neighbors is limited to n, $\frac{n}{2}$ and $\sqrt{n}$.

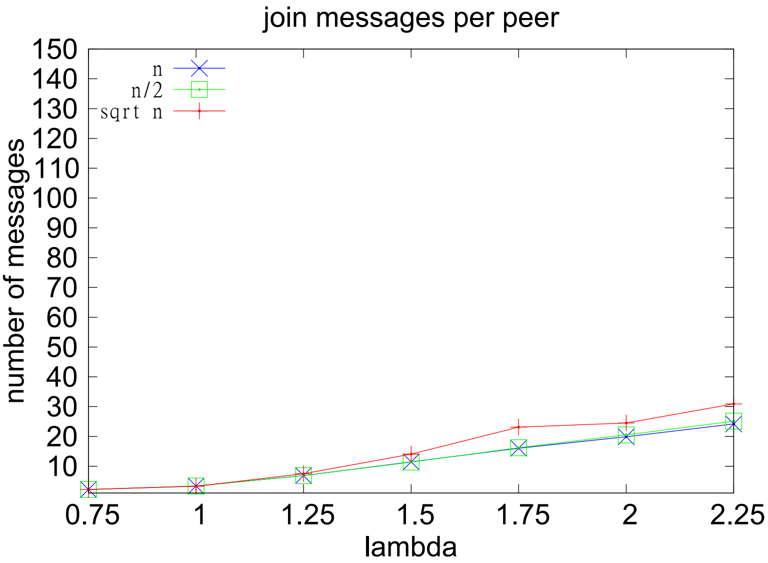


(a) Transit-stub topology

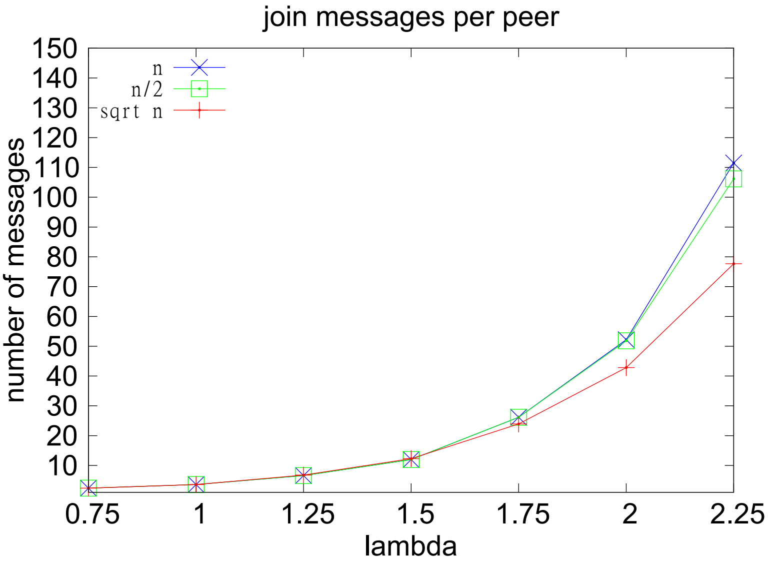
(b) Waxman topology

Figure 6.7: Number of traceroutes with the limited flooding strategy for different λ-values. The maximum number of neighbors is limited to n, $\frac{n}{2}$ and $\sqrt{n}$.

It is not surprising, that higher values for λ yield a higher amount of communication overhead. Since more neighbors are accepted by peers, more path matching processes are executed. In a similar way a higher value for $max_{\mathcal{N}}$ influences that behavior: The number of routes stored on a peer is determined by the number of peers in the neighbor set. We observe for an unbounded



(a) Transit-stub topology



(b) Waxman topology

Figure 6.8: Number of join messages with the limited flooding strategy for different λ-values. The maximum number of neighbors is limited to n, $\frac{n}{2}$ and $\sqrt{n}$.

number of neighbors (i.e. $max_{\mathcal{N}} = |P_D|$) that these values rise rapidly. While having more underlay information locally available at each peer, consequently yields better results in terms of accuracy, but also affects efficiency negatively. A higher number of neighbors increases the chance of a successful path matching, but in turn may cause a split of an existing overlay connection, resulting in new join messages being issued. As stated before, the number of traceroute messages is directly related to the number of join messages. The coupling stems from the behavior in path matching, where peers are motivated to to join others (condition 6.1) or are adopting neighbors from other peers (condition 6.2).

We see that in Waxman topologies the amount of join and traceroute messages rises up to 90% for higher values of λ. While transit-stub topologies reach a sort of upper limit as soon as about 40% of traceroute messages are executed. As described, the same holds for join messages. While this might be a strange observation on the first sight, we will do another experiment that will help to explain that behavior.

We observe that link stress can be reduced by a factor of 2 to 3 for transit-stub topologies, while in Waxman topologies, it is at least possible to achieve ratios from 0.5 to 0.75 compared to unicast. This is caused by the same reason, that increases join and traceroute messages. The difference in stress reduction derives from messages being channeled through one or more transit domains. The chance of finding matching prefixes is therefore much higher than in Waxman topologies, where no such separation is enforced, leading to more different paths being available. The path matching process is less likely to find overlapping underlay routes in such topologies.

Accuracy of random walk

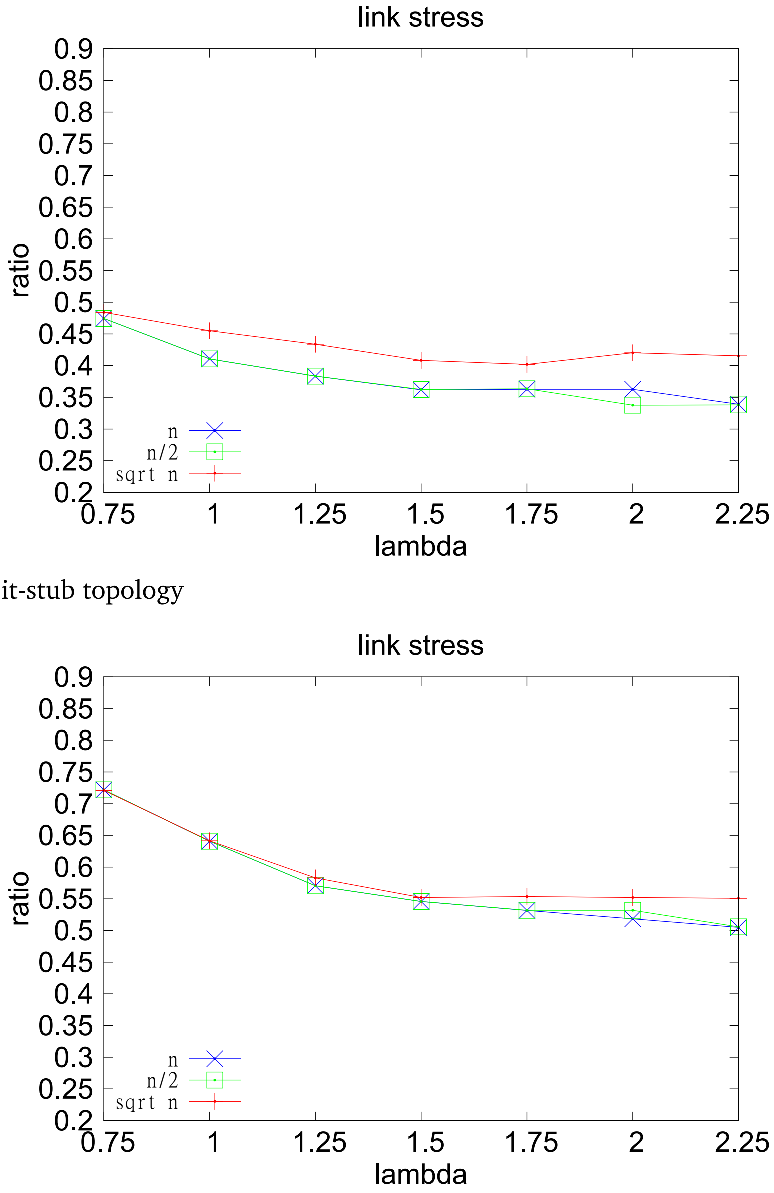
The main focus lies on the different values of σ which determine the amount of probed peers in the neighborhood. The intention is to randomly choose only a subset of neighboring peers for distance measurements, which should reduce communication overhead and we will investigate to what extent results are comparable.

We will use the same metrics as in the first strategy and will again examine efficiency of the process after having considered accuracy.

Investigating on less peers also lowers link coverage, while router coverage remains stable. It is still reasonable for the transit-stub topology that link coverage is increasing with higher σ, reaching similar results as in limited flooding for $\sigma \geq 0.375$. We will determine that this is a critical value.

Keeping in mind that for Waxman topologies, the coverage was highly dependent on the value of the locality factor, the random walk results are quite the same for fixed λ, as long as $\sigma \geq 0.375$. High values for λ are necessary to reach 90% router coverage and 80% link coverage in limited flooding for Waxman topologies. We also observed that coverage may not be a good indication for accuracy.

Considering stretch, once again values for $\sigma < 0.375$ lead to unreasonable results. We observe stretch being bounded even when having $\sigma > 0.5$. While not evident from the stretch figure, we can find the reason for this particular behavior while investigating the figure for the average number of neighbors following.



(a) Transit-stub topology

(b) Waxman topology

Figure 6.9: Link stress ratio with the limited flooding strategy for different λ-values. The maximum number of neighbors is limited to n, $\frac{n}{2}$ and $\sqrt{n}$.

Efficiency of random walk

The values for the average number of neighbors are much lower than in the limited flooding approach, but even if $\frac{n}{2}$ or more neighbors are potentially allowed, the values are not increasing.

So, $max_{\mathcal{N}}$ can be withdrawn as an explanation for the mentioned behavior. It is evident that important peers might get missed during the random walk, which is a reasonable explanation.

When neighbors are missed, the chance of gaining benefit from the path matching in the join process is decreasing. This is also apparent from the results of the number of traceroute and join messages. They indicate how intensive the path matching mechanism is used. The number is much lower than in the limited flooding approach and is not increasing with higher σ values.

As a synopsis for the *random walk* strategy, we can derive that the setting of σ has the desired effect on reducing message overhead during the process. The reduced number of join requests and traceroute execution have a negative effect on underlay-awareness and therefore, are related to the achievable stretch. We observe that there is a critical minimum value of σ being 0.375 in the random walk strategy.

We ran several other experiments for the *random walk* strategy that showed repeated execution has influence on the overall performance. Caused by neighbor sets of peers changing over time, and joining peers are gaining benefit from new information to find overlapping paths. As in the limited flooding approach, high settings for λ have positive influence on peers gaining knowledge on distant other peers.

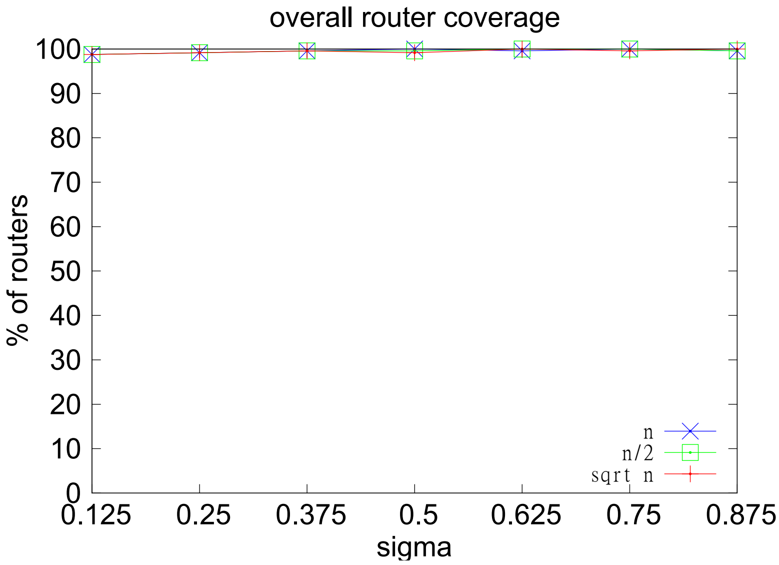
Overall, the results suggest the assumption that this might be worth further research. A hybrid strategy could be used performing as follows to benefit from both approaches:: Starting with a low $\sigma \geq 0.375$ should reduce the amount of flooding and therefore reduce overall communication overhead for both, join messages and especially traceroutes. Using a high $\lambda \geq 2$ helps gaining more information about the underlay and enables the peer to make the right hops on the overlay looking for a peer that being close in terms of distance measurements. The parameter values should change reciprocally over time, meaning increasing σ and decreasing λ.

6.6.3 Anonymous routers and router aliases

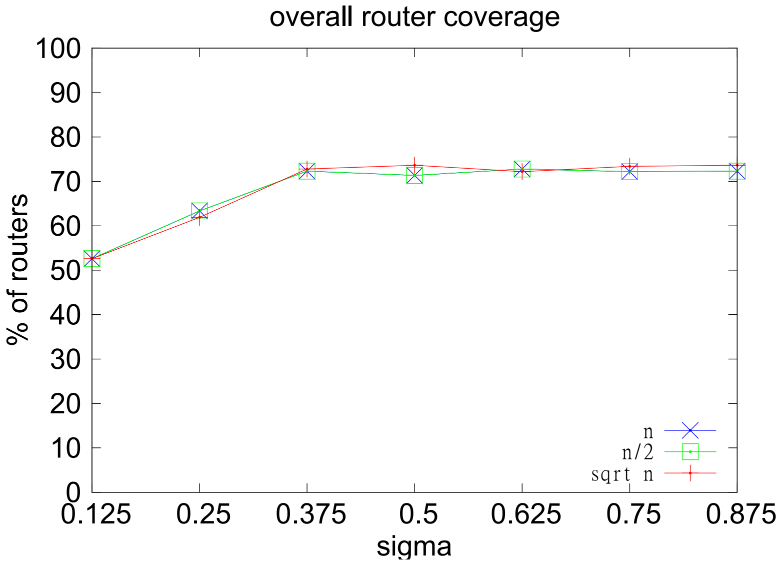
An unfortunate issue when deriving topology representations based on traced routes is the existence of routers with no names, multiple aliases or multiple interfaces with different network addresses.

However, such behavior is not represented in the system model and this work is not intended to handle the occurrence of anonymous routers, but we will briefly discuss how such routers affect the process and refer to existing strategies that can handle such behavior.

That measurement noise might influence the delay reduction during the topology discovery. It would certainly result in more edges on the discovery overlay, but the approach would still not fail. Most efforts that handle the noise simply skip non-responding routers or collapse them with adjacent routers that respond to ICMP messages.



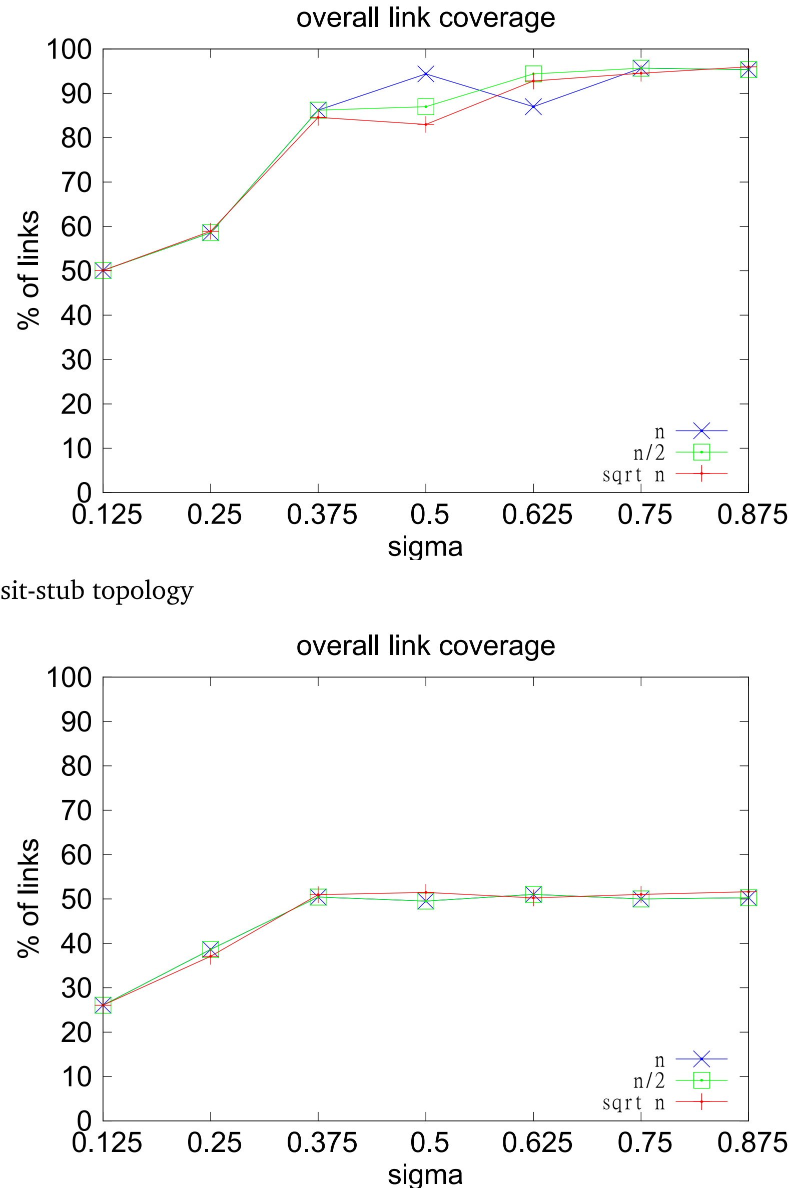
(a) Transit-stub topology



(b) Waxman topology

Figure 6.10: Router coverage for the random walk strategy for different σ-values.

For instance, Barford et al. [3] studied the use of traceroute as a tool for Internet topology discovery. They render a solution for interface disambiguation and router alias resolution that detects and collapses routers with multiple interfaces. Broido et al. [4] propose two methods to bypass anonymous routers in topology inference. They skip non-responding routers and connect valid routers located before and after the inquired router by so called *arcs*. The authors



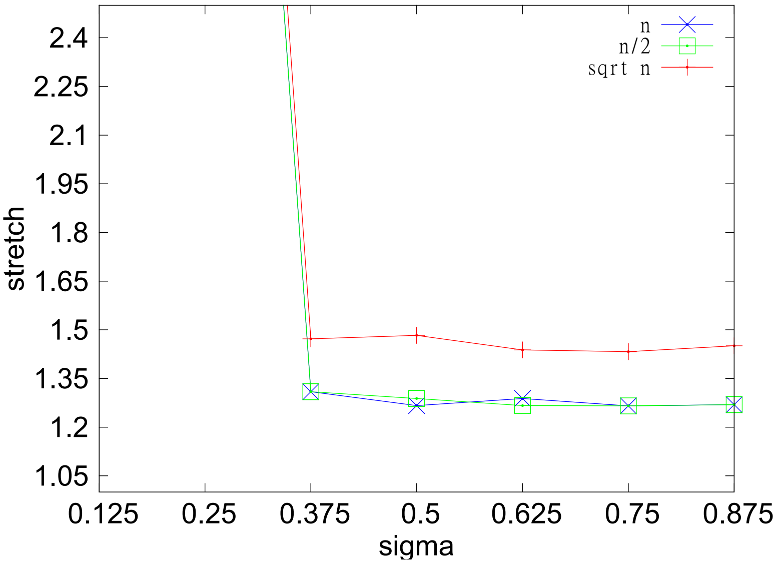
(a) Transit-stub topology

(b) Waxman topology

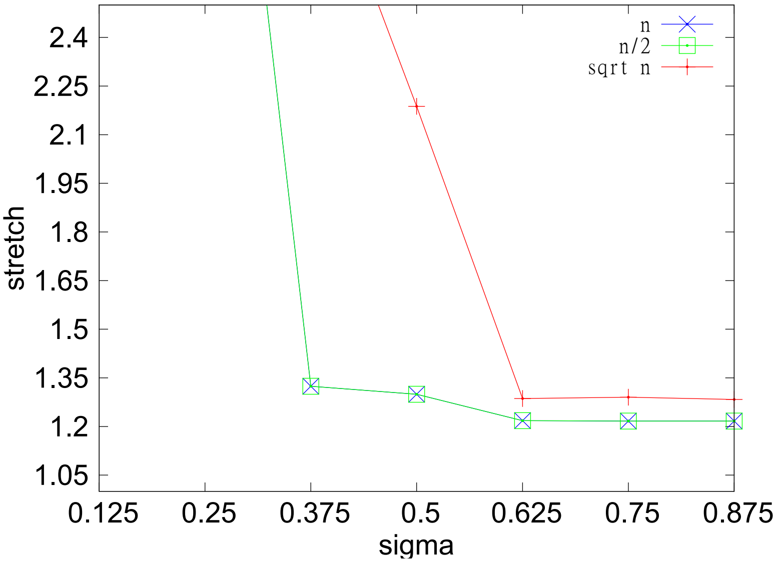
Figure 6.11: Link coverage for the random walk strategy for different σ-values.

alternatively propose the usage of so called *placeholders* that are aiming towards preserving both connectivity and hops.

It's difficult to determine the amount of routers on the Internet that do not accept and respond to traceroute messages. Barford et al. [3] report that less than 13% of router interfaces in the



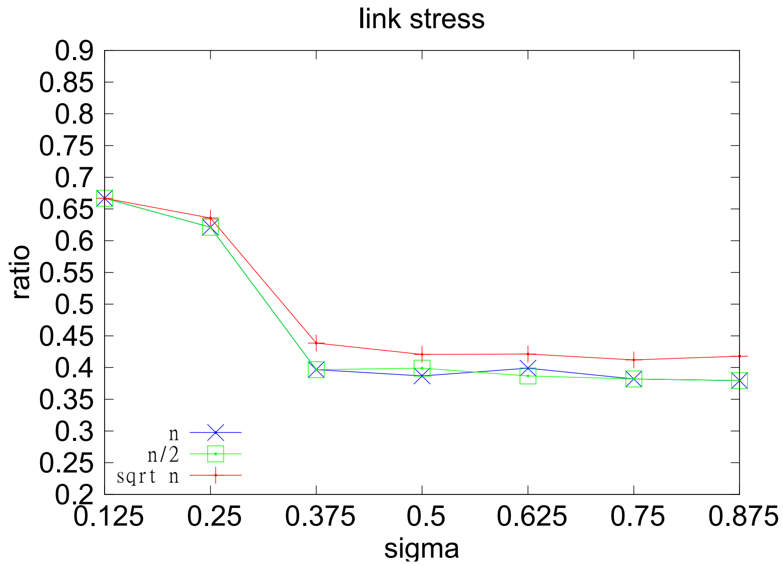
(a) Transit-stub topology



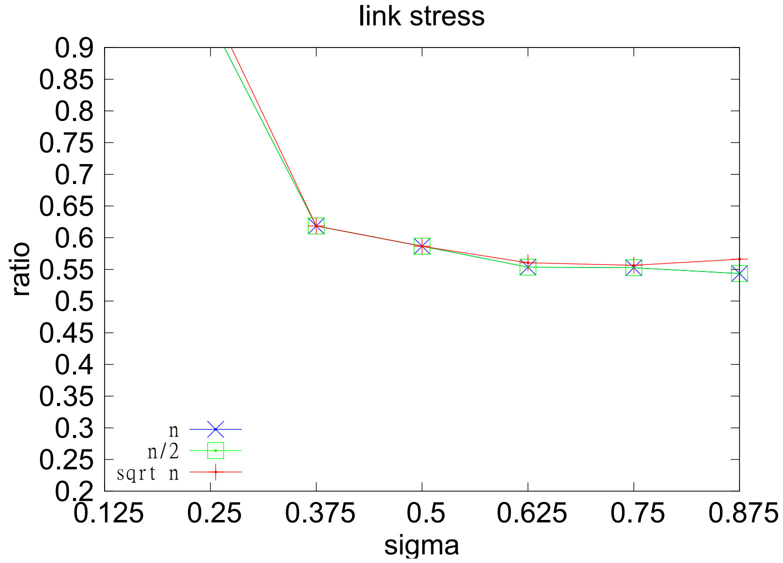
(b) Waxman topology

Figure 6.12: Stretch for the random walk strategy for different σ-values.

Internet did not respond to traceroute ICMP messages [3]. Broido et al. observe that less than 33% of the probed paths in their study contained anonymous or invalid routers.

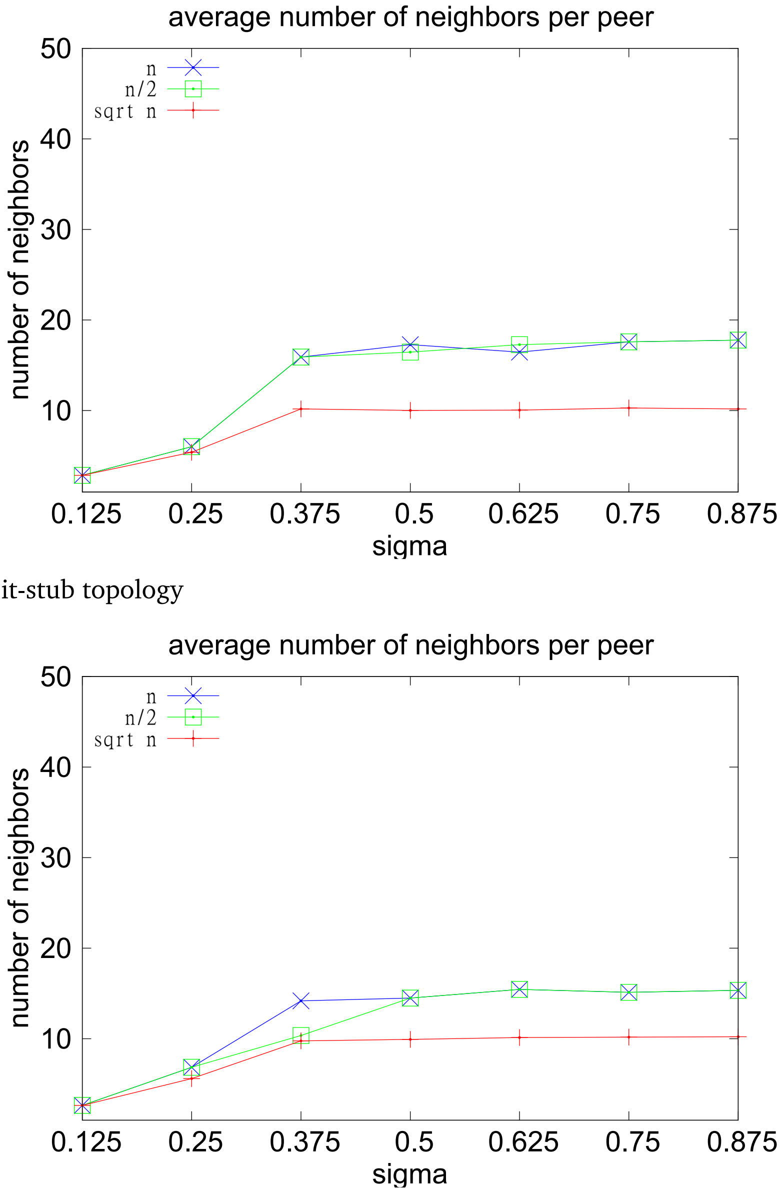


(a) Transit-stub topology



(b) Waxman topology

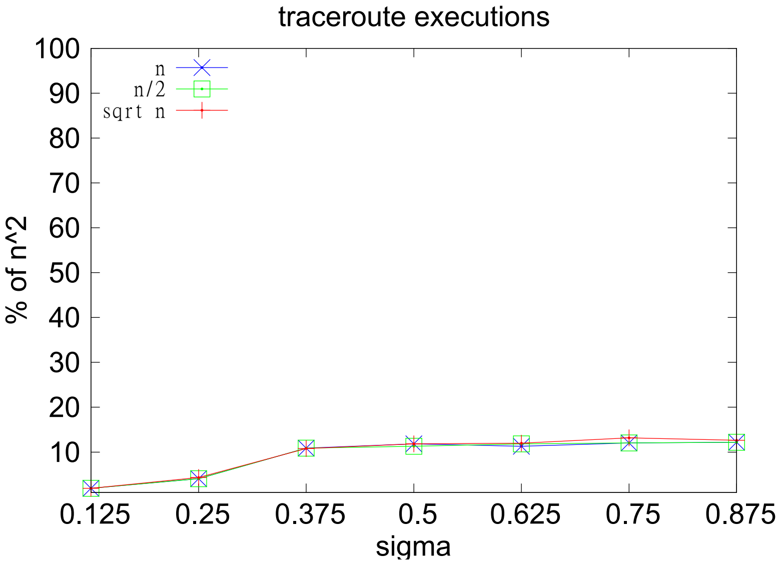
Figure 6.13: Link stress ratio for the random walk strategy for different σ-values.



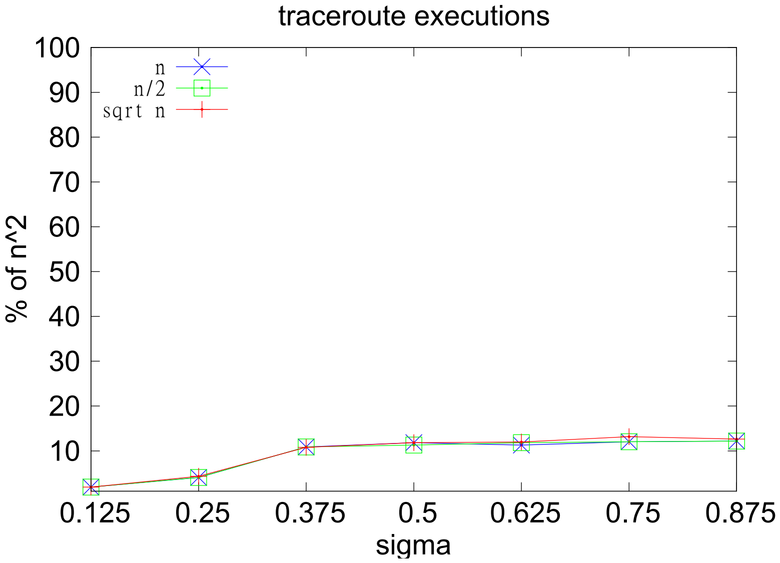
(a) Transit-stub topology

(b) Waxman topology

Figure 6.14: Average number of neighbors for the random walk strategy for different σ-values.

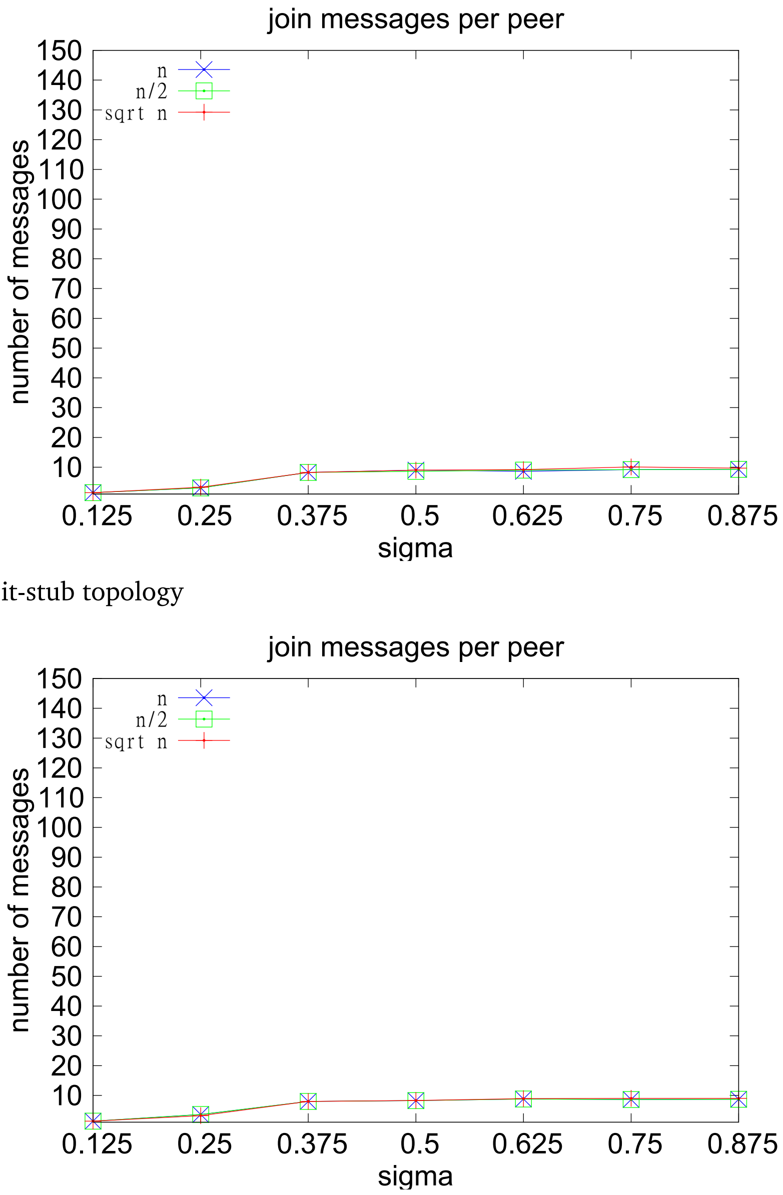


(a) Transit-stub topology



(b) Waxman topology

Figure 6.15: Number of traceroutes for the random walk strategy for different σ-values.



(a) Transit-stub topology

(b) Waxman topology

Figure 6.16: Number of join messages for the random walk strategy for different σ-values.

7 Routing overlay

As described in the system model, we are building an acyclic routing overlay on top of the discovery overlay. Since the discovery overlay is already an accurate representation of the underlay network, we will focus on the establishment of an efficient routing mechanism that reduces stretch of the routing overlay. That stretch is given by the ratio of the overall routing cost $c(R)$ over the achievable routing cost on the underlay network. Since the main goal is the achievement of a low-stretch routing overlay and we cannot influence cost on the underlay, the only available approach is to reduce the overall routing cost $c(R)$.

Definition 7.1 (Minimum Routing Cost Spanning Tree (MRCT)) *Let $\mathcal{T}(D)$ denote the set of all spanning trees of a graph D. Then, $M \in \mathcal{T}(D)$ is a MRCT of D iff $c(M) = \min_{T \in \mathcal{T}(D)} c(T)$.*

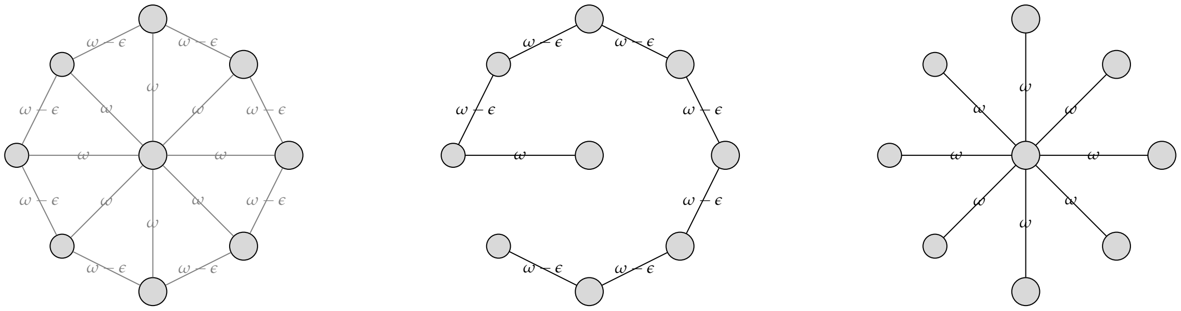
As stated before, finding the minimum routing cost spanning tree is NP-hard. In the following, we will discuss three existing centralized approaches that can be used as heuristics for the problem, and we will briefly explain their shortcomings. Subsequently, we will describe our newly developed distributed algorithm to build a maintainable structure that approximates the MRCT. But first, we will discuss the other approaches.

7.1 Approximation by a Minimum Spanning Tree (MST)

A simple heuristic is a minimum spanning tree since it would at least reduce the cost between any two adjacent peers.

Using an MST would be a handy solution, since it is well-researched and there exist accepted distributed algorithms to build the tree [9, 2]. Considering the fact that the structure focuses on minimum weight edges, the assumption could be valid, MST is approximating an MRCT accurately. But an MST is only serving a good approximation when edges have unit weights or the graph is in general homogeneous.

It is apparent from Figure 7.1 that an MST might produce bad approximations for our purpose, even if the underlying graph is only slightly in-homogeneous. Nevertheless, we will run the experiments with an MST algorithm for comparison.



(a) Topology. Weights are given by $\omega > \epsilon > 0$.

(b) Minimum Spanning Tree (others possible).

(c) Minimum Routing Cost Tree.

Figure 7.1: A Minimum Spanning tree failing to approximate a Minimum Routing Cost Tree of a given topology. A similar observation was made by Chao et al.[7]

7.2 Approximation by a Shortest Path Tree (SPT) rooted at the median

A shortest path tree rooted at the median of a graph was proposed by Wong et al. [27] as an approximation for the *shortest total path length spanning tree problem* [11]. It is possible to achieve a 2-approximation of the MRCT using that approach [28].

Obviously, this structure is highly fragile. When churn occurs, even slight change may cause a different node being the median of the graph and will lead to a different tree. That causes the shortest path tree having to be recalculated to keep approximation bounds valid.

Another reason making this structure unfeasible for our purpose, is that median information is based on summing up all distances from every node to all other nodes. This involves shortest-path knowledge for all pairs. Determining such global information is inefficient both in terms of space and messages exchange.

7.3 Approximation by Campos' algorithm

Campos et al. [6] proposed a fast and simple centralized MRCT approximation. The authors claim to be able to provide a solution having the same routing cost as an MRCT in practice. The approach modifies Prim's minimum spanning tree algorithm: Instead of taking solely edge weights into account, the algorithm calculates a *spanning potential* to select a parent node. It combines degree of a node, incident edge weights and information about adjacent nodes.

We will include that centralized algorithm in our evaluations.

7.4 Approximation by our core-based approach

Due to unstable results, a distributed minimum spanning tree algorithm is unsuitable, especially in heterogeneous topologies.

Developing a distributed algorithm for Campos' approach is thinkable as it primarily based on concept of minimum spanning trees. . Even if Campos' algorithm overcomes some of the drawbacks of MSTs, the authors admit that it is not possible to derive any propositions about the quality of the outcome [6].

As stated, a problem of using median-rooted SPT is that the properties of the tree depend on a single node that is derived by global knowledge. Small changes in the topology might influence the overall structure. There are some distributed algorithms available, detecting the median of a graph by comparing all possible SPTs, lacking the possibility of future scaling (see [15, 22]).

Therefore, we will focus to find a more sophisticated structure within the discovery overlay to derive the router overlay: An optimized subgraph of the discovery overlay is constructed, that will be called the *core* of the graph. The core will consist of nodes that are important for overall routing cost improvement. It is composed of the high-routing load peers and aims to improve distant paths between peers. Consequently, we will connect the remaining peers to the core by using shortest paths on the discovery overlay.

The core is meant to serve a similar purpose on the peer-level as transit-domains do on the router level. An important difference is allowing the core to change dynamically over time. The core is therefore elected by other peers.Choosing the right edges $E_R \subseteq E_D$ is the critical factor. We will detail that requirements in the following section by analyzing the structure of minimum routing cost trees and looking for efficient ways to build the routing overlay.

7.4.1 Structure of an MRCT

As a first step, we will analyze the general structure of minimum routing cost trees to explain the intuition of our solution: The original formula for the routing cost $c(R)$ sums up the distances between each pair of vertices on the routing overlay. As we cannot derive structural information from the edge weights, we have to observe how often an edge weight is actually summed up. The formula counts the usage of each edge $e \in \rho_R(u,v)$ for all $u, v \in P_D$. On multiplying the usage by the weight of e, we get e's contribution to overall routing cost.

Lemma 7.1 *An edge $e \in E_R$ that connects the two subtrees induced by the vertex sets $V' \subset V_R$ and $V'' \subset V_R$ that result by removing e from R is used $2|V'||V''|$ times.*

This is evident, if we regard the number of vertices in every emerged subtree. Consider the set of vertices V'. Every vertex $u \in V'$ uses e on a path to every $v \in V''$. That holds as well for the opposite direction.

Proof 7.1 *As in [28]:*

$$\sum_{u \in P_D} |\{v \in P_D | e \in \rho_R(u,v)\}| = \sum_{u \in V'} |V''| + \sum_{u \in V''} |V'| = 2|V'||V''|$$

We will call that usage of an edge the *load* of it. Using the above observation, we can define the load of an edge by:

Definition 7.2 (Load) *The load of an edge e in a tree R that connects the two subgraphs S' and S'' that result by removing e is given by*

$$ld_R(e) = 2|V_{S'}||V_{S''}|$$

Multiplying the load of an edge by its weight permits to calculate the overall routing cost of the tree. It leads to the following lemma:

Lemma 7.2 *An alternative formula of the overall routing cost $c(R)$ of a tree R is given by*

$$c(R) = \sum_{e \in E_R} ld_R(e)\omega_R(e)$$

Proof 7.2 *As in [28]: Let $\rho_R(u,v)$ denote the path from a node u to a node v on the tree R. Then,*

$$\begin{aligned} c(R) &= \sum_{u \in P_D} \left(\sum_{v \in P_D \setminus \{u\}} d_R(u,v) \right) \\ &= \sum_{u \in P_D} \left(\sum_{v \in P_D \setminus \{u\}} \left(\sum_{e \in \rho_R(u,v)} \omega_R(e) \right) \right) \\ &= \sum_{e \in E_R} \left(\sum_{u \in P_D} |\{v \in P_D | e \in \rho_R(u,v)\}| \right) \omega_R(e) \\ &= \sum_{e \in E_R} ld_R(e)\omega_R(e) \end{aligned}$$

While not solving the actual problem, we can at least observe that there are certain edges that might contribute more to the overall routing cost than others. Edges with a high routing load contribute multiple times with their weight. Therefore, finding edges exhibiting both high routing load *and* low weight, will overall yield a low routing cost.

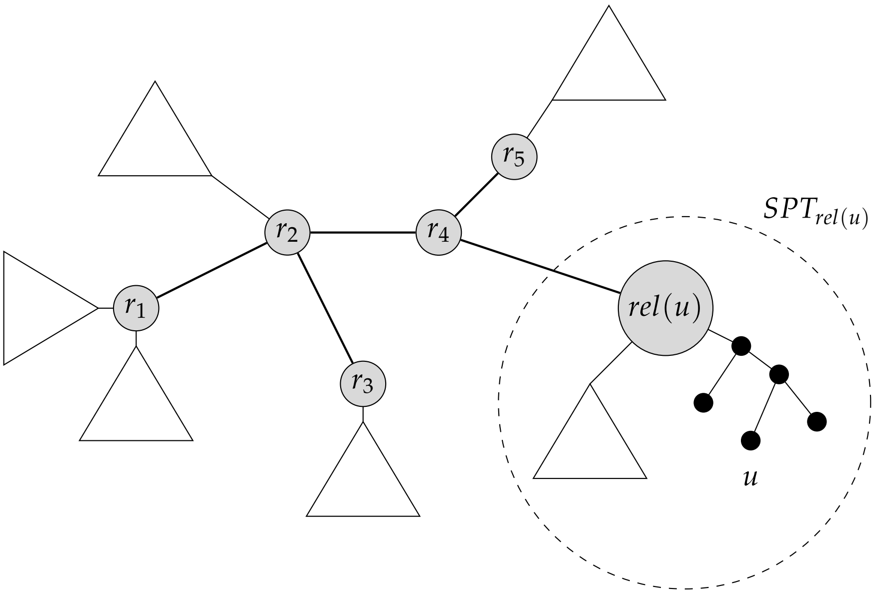


Figure 7.2: The core of the graph is a tree connecting the relay nodes.

Obviously, peers are not able to locally detect the routing load of edges in the future overlay. We will establish a mechanism that collects local guesses of peers to derive an estimated contribution. Therefore, we will generate the estimations based on peers, rather than on their links. As it is virtually impossible to constitute in advance whether links will be part of the future overlay. As peers will be naturally part of the tree, they will be used as a representation of the load on the adjacent edges.

Having exposed the intuition of the approach, we will now describe the desired structure in detail and discuss the distributed algorithm. At the end of this section the approach is evaluated and compared to existing mechanisms. Since the other approaches are centralized, message and space complexity will be evaluated analytically.

7.4.2 Desired structure of the core

The decomposition of the discovery overlay in multiple connected trees provides several benefits. The graph is split into multiple shortest path trees, where one among them is forming our core. As a consequence, trees might be maintained at least partially independent keeping topological changes mostly local.

Nevertheless, the structure needs to stay connected and the marginal core-peers will serve this purpose. They will serve as a connection for non-core peers in a shortest path tree. We will call those peers *relay nodes*:

Definition 7.3 (Core, Relay node) *We will denote the set of peers in the core by $\mathcal{C}$. We call a node $rel(u) \in \mathcal{C}$ the relay node of u if it has the minimum distance to u among all nodes in $\mathcal{C}$. So, $d_R(u, rel(u)) = min_{r \in \mathcal{C}} d_R(u, r)$. The shortest path tree rooted at $rel(u)$ is denoted by $SPT_rel(u)$.*

Paths on the routing overlay in general start at a peer u in a branch of the tree $SPT_{rel(u)}$, using a shortest path to the relay node $rel(u) \in \mathcal{C}$, allowing them to reach other overlay peers, as depicted in Figure 7.2.

Using that structure, enables us to give an upper bound on the path distance in R:

$$d_R(u,v) \leq d_{SPT_{rel(u)}}(u, rel(u)) + d_{\mathcal{C}}(rel(u), rel(v)) + d_{SPT_{rel(v)}}(rel(v), v)$$

More detailed bounds are discussed in chapter 7.8. But firstly, we will focus on the algorithm in the following section.

7.5 Distributed algorithm

Our approach forms the routing overlay by connecting multiple shortest path trees. Solving shortest path related problems is a sub-problem of many routing protocols or network optimization algorithms in general. It often turns out to be the performance bottleneck of them as well. Depending on the algorithm used, it has negative influence on either time, space or message complexity.

Most of the algorithms have in common to use the basic formulation of the Bellman inequality. It is a rather local property that expands globally. It describes the *optimal substructure property* of shortest paths stating that any shortest paths is composed of further shortest paths [28]. We will take advantage of that property in the following.

Peers are influenced by that local property to optimize routes by determining shortest paths in their neighborhood. We may assume that nodes connected to edges causing an improvement are important.

The main algorithm consists of two separate phases: The voting phase and the connection phase.

7.5.1 Voting phase

The peers form a spanning tree for temporary communication. Any distributed spanning tree algorithm is sufficient, for instance as proposed by Awerbuch [2]. The peers will attempt to influence the formation of the core to their advantage. Each peer elects among its neighboring peers while all votes are gathered by sending them along the spanning tree. Peers receiving higher votes are considered to be important due to their low-latency link. Peers that acquired more than the average amount of votes will form the core after being informed by the root of the temporary spanning tree. Each relay node $r \in \mathcal{C}$ is aware of being part of the core.

In detail, the algorithm is implemented similar to the *synchronizer β* by Awerbuch [1]: In an initialization phase a spanning tree is formed and the root is a temporary *leader* in the network. The leader sends a message along the tree, notifying other nodes that the voting process may commence. Starting at the leaves of the temporary spanning tree, the votes are summed up and propagated towards the leader. The temporary root can now distinguish non-core peers from the relay nodes by summing up the votes each peer has received from other peers. The temporary structure is then used to inform all peers about the relay nodes.

7.5.2 Connection phase

The remaining non-core peers have to choose one of the relay nodes from the core as it is not guaranteed that one of the peers they voted for eventually made it to the core. The challenge is that peers have to choose the *closest* of the core even if it is multiple hops away and they are not aware of the distance.

A peer u has to find the closest relay node $rel(u) \in \mathcal{C}$ and take a position in the shortest path tree $SPT_{rel(u)}$ rooted at that relay node. Therefore, each peer executes the **Init Peer** procedure proposed in Algorithm 7.1: In this state, no routing information to the core is known. If a peer detects that itself as being part of the core, it will immediately inform its neighbors. Non-core peers need to send a $\langle sync \rangle$-message. After the initialization exactly 2 messages have been sent along each link.

The peers keep track of an estimated distance to the relay nodes. They might receive messages from neighbors containing the neighbor's distance estimate to a relay node. A peer recomputes the value by adding the distance to the neighbor and might set the neighbor as the next hop to the relay node, in case of providing an improvement.

Each distance update would potentially cause a flood of updates to other nodes. We strive to avoid such a distance update causing a complete change on all connected descendants in the shortest path tree. Therefore, our algorithm uses a synchronization approach as introduced by Awerbuch et al. [1], reducing the amount of messages produced. Peers consider a phase to be finished once they have received a message from all of their active neighbors.

Algorithm 7.1 Init Peer (at node u)

```
var $dist : \mathcal{C} \rightarrow \mathbb{R} \cup \{\infty\}$
var $nextHop : \mathcal{C} \rightarrow \mathcal{N}_u \cup \{\perp\}$                // one hop routing table
var $\mathcal{C}_{updated}$ : set of $\mathcal{C}$         // may contain $s$ after receiving a $(r, d_{vr})$ message
var $\mathcal{N}_{rcvd}$ : set of $\mathcal{N}_u$                                   // track responses

procedure INIT
    $\mathcal{C}_{updated} \leftarrow \emptyset$ ; $\mathcal{N}_{rcvd} \leftarrow \emptyset$
    for all $r \in \mathcal{C}$ do
        $nextHop[r] \leftarrow \perp$ ; $dist[r] \leftarrow \infty$
    if $u \in \mathcal{C}$ then
        $nextHop[u] \leftarrow u$ ; $dist[u] \leftarrow 0$
        send $(u, 0)$ to all $v \in \mathcal{N}_u$         // updates tables and rcvd-set at neighbors
    else
        send $\langle sync \rangle$ to all $v \in \mathcal{N}_u$              // needed for synchronization
```

There are two types of messages that a peer may receive from a neighbor in the **OnReceive** procedure (see Algorithm 7.2): Distance updates of the form (r, d_{vr}) and status updates of the form $\langle state \rangle$.

Once a peer u receives a distance update from a neighboring peer v, it evaluates whether the new information would be an improvement in reaching the core through the neighbor. It might happen, that this is the first time that u can estimate its distance to the relay node. It updates its routing information and adds the relay node to $\mathcal{C}_{updated}$. Note that u does not instantly inform the other neighbors about improved paths to a relay node. Instead, it adds the neighbor v to $\mathcal{N}_{rcvd}$ and waits until the synchronization phase has come to an end. By maintaining the set $\mathcal{N}_{rcvd}$ a node can determine whether it has received messages from all of its neighbors.

Algorithm 7.2 OnReceive(at node u)

```
procedure OnReceive($r$, $d_{vr}$) // $d_{vr}$ is $v$'s current distance estimate to a relay node $r \in \mathcal{C}$
    if $(u \notin \mathcal{C}) \wedge (dist[r] > \omega(u,v) + d_{vr})$ then        // would $(u,v)$ help to better reach $r$?
        $nextHop[r] \leftarrow v$                                              // update routing table
        $dist[r] \leftarrow \omega(u,v) + d_{vr}$
        $\mathcal{C}_{updated} \leftarrow \mathcal{C}_{updated} \cup \{r\}$                          // because $dist[r]$ was improved
    $\mathcal{N}_{rcvd} \leftarrow \mathcal{N}_{rcvd} \cup \{v\}$                  // We will not send the improvement right now...
    CheckSync()                                 // ...but wait until received from all ('sync')

procedure OnReceive($state$)                                          // sent by neighbor $v$
    if $state = \langle sync \rangle$ then
        $\mathcal{N}_{rcvd} \leftarrow \mathcal{N}_{rcvd} \cup \{v\}$
    CheckSync()
```

Once having received a message from a neighbor, a peer runs **CheckSync** (see Algorithm 7.3). It evaluates, whether it has reached the end of a phase which is the case when $\mathcal{N}_{rcvd} = \mathcal{N}_u$. If the phase has ended, it examines whether the route information to any relay node has been updated since the last phase (i.e. $\mathcal{C}_{updated} \neq \emptyset$). In that case, then the closest relay node is chosen and a distance update is sent to the neighbors. Otherwise, if the peer has finished the phase but no route information was updated, a $\langle sync \rangle$-message is sent.

Algorithm 7.3 CheckSync(at node u)

```
procedure CheckSync
    if $\mathcal{N}_{rcvd} = \mathcal{N}_u$ then                   // $u$ received messages from all active neighbors
        if $\mathcal{C}_{updated} \neq \emptyset$ then
            $r_{opt} \leftarrow r_{min}$ such that $dist[r_{min}] \leq dist[r'], \forall r' \in \mathcal{C}_{updated}$          // closest relay
            send $(r_{opt}, dist[r_{opt}])$ to all $v \in \mathcal{N}_u$
        else
            send $\langle sync \rangle$ to all $v \in \mathcal{N}_u$
        $\mathcal{N}_{rcvd} = \emptyset$
        $\mathcal{C}_{updated} = \emptyset$
```

Termination

We will now enhance the termination behavior of the algorithm. In the current state the algorithm ends implicitly after a certain amount of messages sent, which is called *message termination*. Since the peers are not aware of the size of the overlay, they are not capable to determine whether a stable state is reached. We will improve that to explicit process termination without having the peers to know $|P_D|$. This is achieved similar as in proposed by Bui et al. [5]. Therefore, we introduce the set $\mathcal{N}_{inactive}$ containing all inactive neighbors, being set to $\emptyset$ in **Init Peer**.

We introduce a new state, namely $\langle inactive \rangle$, which needs to be handled by **OnReceive**. It causes a minor, straight-forward enhancement. Once a node receives such a state update from its neighbor v, it will add v to $\mathcal{N}_{inactive}$. That is shown in Algorithm 7.4.

Algorithm 7.4 OnReceive(at node u)

```
procedure OnReceive($state$)                                  // sent by neighbor $v$
    if $state = \langle sync \rangle$ then
        $\mathcal{N}_{rcvd} \leftarrow \mathcal{N}_{rcvd} \cup \{v\}$
    else if $state = \langle inactive \rangle$ then
        $\mathcal{N}_{inactive} \leftarrow \mathcal{N}_{inactive} \cup \{v\}$
    CheckSync()
```

The new state can now be used by **CheckSync** such that it treats all inactive neighbors as if they had sent a message. So, a phase is finished once $\mathcal{N}_{rcvd} = \mathcal{N}_u \setminus \mathcal{N}_{inactive}$ holds (see Algorithm 7.5).

A peer sends $\langle inactive \rangle$ when optimization is finished. That is the case, when all neighbors are either inactive or have sent a distance update without improvement. That happens if the neighbors do not know a better distance to the core, so the update had no effect on $\mathcal{C}_{updated}$.

Remember that the overall process starts at the core peers and distance information is sent downwards the future shortest path tree. A peer can process terminate as soon as $\mathcal{N}_{inactive} = \mathcal{N}_u$ *and* it has sent $\langle inactive \rangle$ to all neighbors. Such a message is only sent once to all neighbors. It is therefore the last message from u to its neighbor v.

Algorithm 7.5 CheckSync(at node u)

```
procedure CheckSync
    if $\mathcal{N}_{rcvd} = \mathcal{N}_u \setminus \mathcal{N}_{inactive}$ then      // $u$ received messages from all active neighbors
        if $\mathcal{C}_{updated} \neq \emptyset$ then
            $r_{opt} \leftarrow r_{min}$ such that $dist[r_{min}] \leq dist[r'], \forall r' \in \mathcal{C}_{updated} \setminus \mathcal{N}_{inactive}$
            send $(r_{opt}, dist[r_{opt}])$ to all $v \in \mathcal{N}_u$
        else
            send $\langle inactive \rangle$ to all $v \in \mathcal{N}_u$                 // inactivation message
                                                      // process termination check goes here
        $\mathcal{N}_{rcvd} = \emptyset$
        $\mathcal{C}_{updated} = \emptyset$
```

7.5.3 Routing overlay formation

The remaining steps are straight-forward. Once a peer process terminates in the previous step, it is aware of its closest relay node in the core. It also knows the next hop on the shortest path. By sending a request to the next hop peer, all non-core nodes can form the shortest path tree which is rooted at the relay node. If a peer has process terminated and receives such a request, it marks the incident link as part of the routing overlay. Using the described distributed algorithm, we managed to connect all non-core nodes to their closest relay node using only shortest paths on the discovery overlay.

For our evaluations, we will presume that all relay nodes in the core also form a shortest path tree. We will also assume that once a request for using a next hop peer is received, this will be counted and as a result the relay node knows the number of connected peers. The relay node serving the most non-core peers will be the root of the shortest path tree.

We will show in the evaluations that this structure already performs very well compared to other approaches. Further investigation is possible and may yield even better results.

7.5.4 Churn

We will now discuss the process of new peers opting in to the routing overlay, as well as removing peers that disconnect. This is caused by churn that is handled in the topology discovery overlay (see chapters 6.2 and 6.5).

If a new peer p joins the routing overlay and no tree overlay edge was affected by rearrangement in the discovery overlay, the peer can determine the distance to relay nodex through $\mathcal{N}_p$. After receiving all distance messages the peer performs **OnReceive** and **CheckSync**.

If the structure of R was changed by rearrangement in the discovery overlay, the affected shortest path trees may have to be recalculated. This can be triggered through the discovery overlay. The corresponding relay node needs to run **Init** assuming that any neighboring core-peers being inactive.

7.6 Publish/subscribe routing

Any publish/subscribe system that benefits from end-to-end delay reduction can be established. As an example, we will describe in the following how a simple, but efficient publish/subscribe system that uses subscription forwarding can be established.

The publish/subscribe system uses the routing overlay to propagate event notifications. Therefore, each peer maintains a covering filter for every adjacent edge in E_R. The routes are determined by filters on peers and notification events follow a corresponding reverse path. The filters are determined by the subscriptions sent by other peers: If a subscription is received, the filter of the corresponding edge is extended, so that the edge can act as a reverse path when event notifications are received.

When exploiting similarities among subscriptions resource usage can be improved. This is achieved by propagating subscriptions only along paths that have not been covered by previous forwarding of subscriptions. Whether subscriptions are forwarded along the other adjacent links is determined by the corresponding filter. If that filter already covers the subscription, it does not need to be forwarded, since the necessary route is already established.

Once an event notification is received or a peer wants to publish an event, it can determine by the corresponding filters whether it should forward the event notification along a neighboring link. So, we will guarantee to send notifications only towards the direction of interested subscribers.

7.7 Evaluations

7.7.1 Complexity analysis

Voting phase

Let $n = |P_D|$, $m = |E_D|$. Any off-the-self distributed algorithm is sufficient to form a temporary spanning tree. For instance, Awerbuch's MST algorithm [2] which uses $O(m + nlogn)$ messages and runs in $O(n)$ time can be considered. There exist minimum spanning tree algorithms that run in sub-linear time (see [10]). Actually, it is not necessary for the spanning tree to be minimum-weighted, but it is beneficial.

During the voting exactly $n - 1$ messages are sent. The temporary root informs all other peers with $n - 1$ messages about the relay nodes with a message of size $O(|\mathcal{C}|)$.

The algorithm starts at the core peers. So, actually only the $|\mathcal{C}|$ core nodes need to be informed. The message size is in $O(|\mathcal{C}|)$. The time needed is proportional to the height of the tree, which might be $n - 1$ in the worst case.

Connection phase

The algorithm has to store $dist$ and $nextHop$ to the core, so the space complexity is given by $O(|\mathcal{C}|)$.

The process consists of synchronized phases in which the sets $\mathcal{N}_{rcvd}$ and $\mathcal{N}_{inactive}$ are filled. At the end of each phase, a peer sends at most one message per neighbor. So up to $O(mn)$ messages are sent in the whole process for $m = |E_D|$. Each peer in turn sends n messages in the worst case to each neighbor in the whole process (in either *Init* or *CheckSync*) along each link.

This yields $O(mn)$ messages in the worst case of $O(n)$ phases. In every phase at least one peer finds the optimum relay, if the algorithm starts at the relay nodes.

The messages are either status or distance updates, so message size is constant.

7.7.2 Simulations

Experimental setup

We will use the same topologies and the number of peers as described in chapter 6.6.2. The first simulations are run allowing unlimited size in neighbor sets. A subsequent run will limit the size to $max_{\mathcal{N}} = \sqrt{n}$.

We consider minimum spanning tree, a shortest path tree rooted at the median of the discovery overlay, and Campos' minimum routing cost approximation algorithm [6]. We use a discovery overlay derived by utilizing the limited flooding strategy. Consistently, we are comparing again the two different topologies, Waxman and Transit-stub (see chapter 6.6.2 for topology characteristics).

Metrics

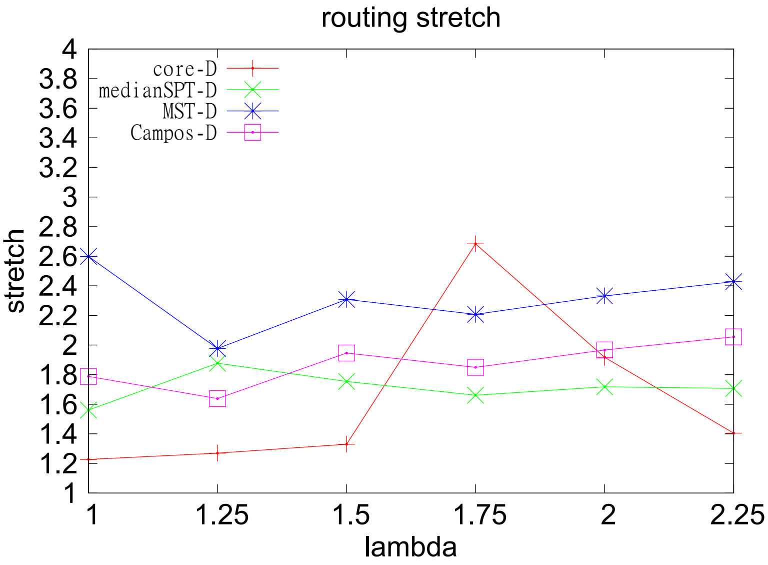
Since the other approaches are centralized, we are not able to compare metrics related to efficiency. For our approach, efficiency and complexity is previously discussed in chapter 7.7.1.

We will concentrate on two different values for stretch: The stretch of the routing overlay on top of the discovery overlay, as well as stretch of the routing overlay on top of the underlay network.

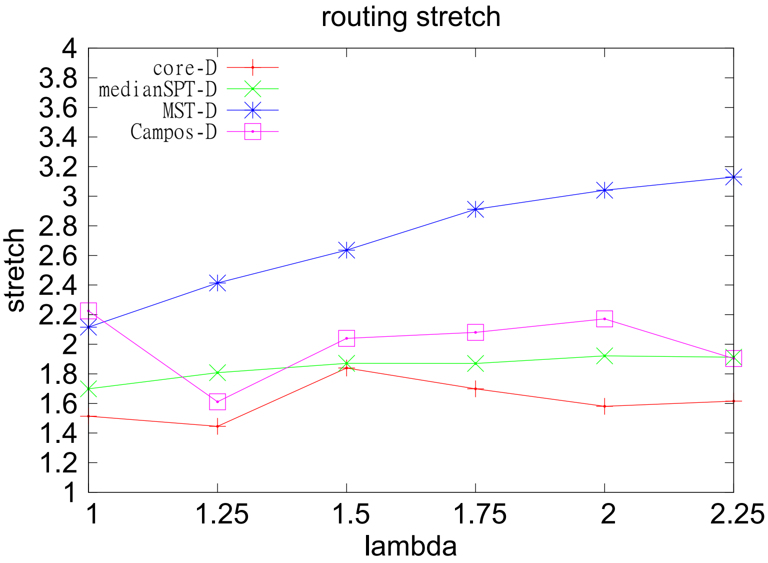
Results

It is evident from Figure 7.3 that the difference in topologies seems to be more influential than during the building of the discovery overlay. For Waxman topologies, the approach outperforms all other approaches, producing a stretch on the discovery overlay lower than 1.8 for all settings of λ.

The results on the transit-stub topology is promising for $\lambda \leq 1.5$, providing less than 1.3 stretch on the discovery overlay. However, a slight increase of the locality factor ($\lambda = 1.75$) causes doubling of the stretch. An explanation for this phenomenon will be given in chapter 7.8 showing that the stretch of the core-based approach is solely dependent on the right choice of



(a) Transit-stub topology

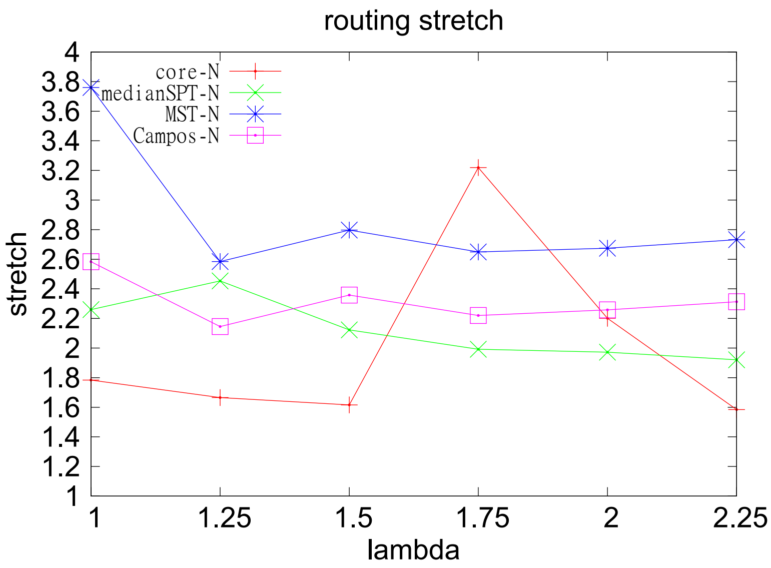


(b) Waxman topology

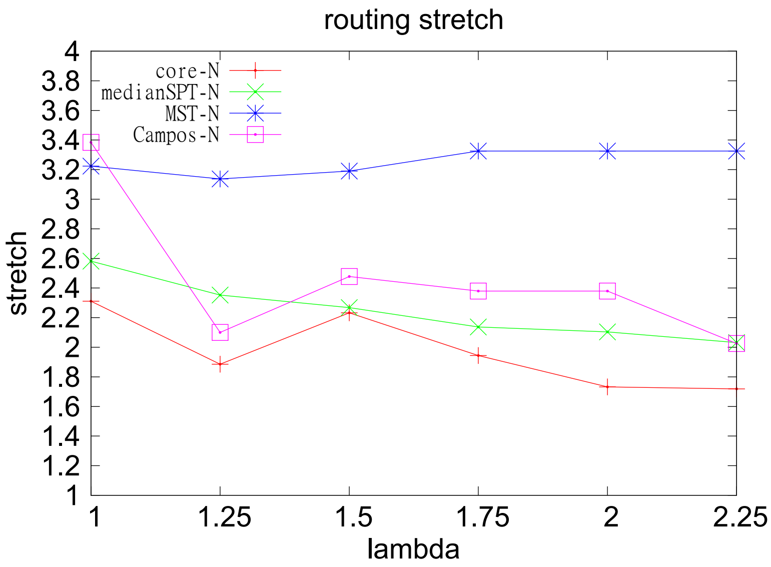
Figure 7.3: Overlay routing stretch compared to discovery overlay for different λ-values. The maximum number of neighbors is limited to n.

relay nodes. That leads to the conclusion that the behavior is caused by a wrong selection of relay nodes.

The described behavior for the transit-stub topology also occurs when considering the stretch compared to the underlay network as shown in figure 7.4. Starting at stretch values lower



(a) Transit-stub topology



(b) Waxman topology

Figure 7.4: Overlay routing stretch compared to the underlay network for different λ-values. The maximum number of neighbors is limited to n.

than 1.8, and performing distinctly better than the other approaches, the stretch again nearly doubles for higher setting of the locality factor.

In the Waxman topology again the stretch on the underlay is slightly higher compared to transit-stub, ranging from about 1.75 to 2.3. The same is valid for stretch on the discovery overlay, ranging from 1.4 to 1.8.

We will extent simulations on the same topologies, by limit $max_{\mathcal{N}}$ to a more reasonable setting of $\sqrt{n}$.

Considering the transit-stub topology, the stretch on D of the core-based overlay is again lower than that of the other approaches for $\lambda < 1.75$. We can observe that the same holds for the stretch on the underlay network, which is obviously higher. The observed effect is again occurring for $\lambda \geq 2$. The discovery overlay should clearly focus on locality when running the core-based approach.

For Waxman topologies the improvement over the other approaches is - similar to transit-stub - given for smaller values of $\lambda \leq 1.5$. The achieved improvement in stretch is not that distinct as in transit-stub topologies with unbounded number of neighbors, but still the approach yields the best results of all.

7.7.3 Summary of results

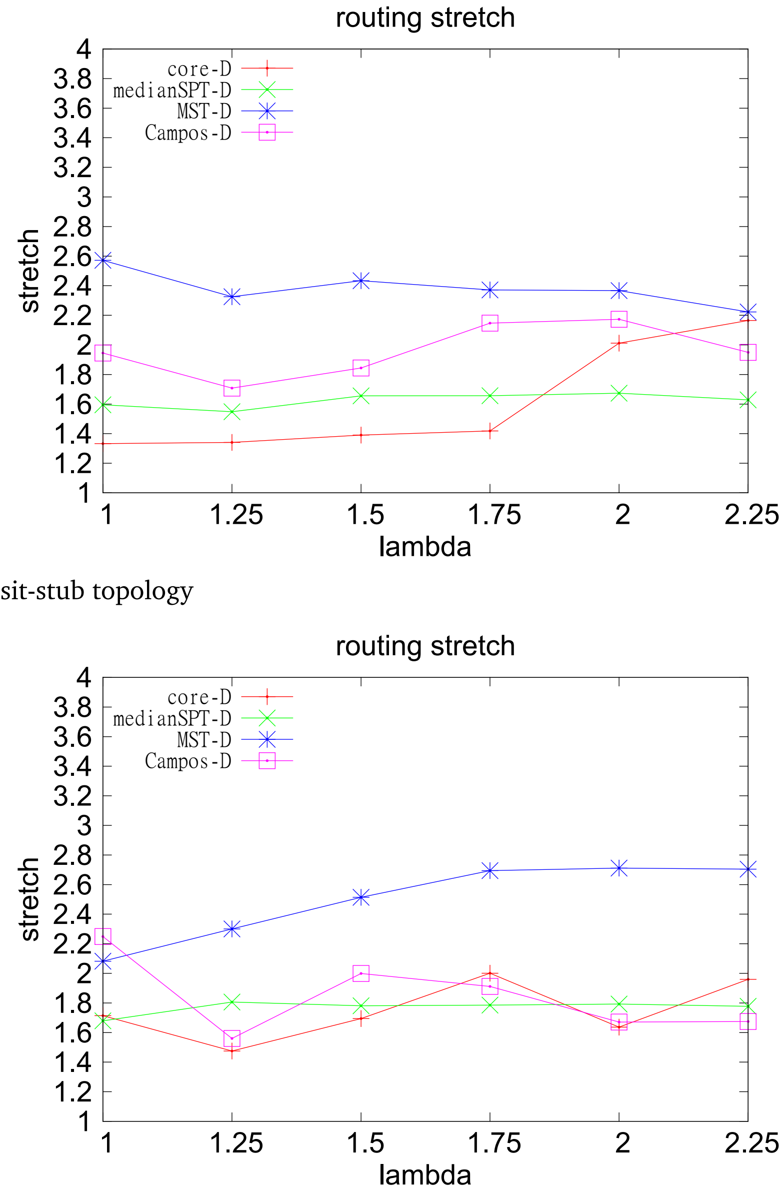
Overall, we can suggest that the discovery overlay should focus on locality when running the core-based approach. We have shown in simulations, that for values of $\lambda < 1.5$ the approach outperforms all other approaches on both topologies under these conditions.

7.8 Upper and lower bounds on cost and stretch

We will first investigate to what extent upper and lower bounds of overall routing cost and stretch are determined by the core-based structure of the routing overlay. Bounds and approximation ratios on MRCTs were extensively researched by Wu, Chao, Tang et al. [29, 28].

We are considering lower bounds first. All distances on the routing overlay are obviously at least as high as the distances in the discovery overlay, since R is a subgraph of D. So, for all $u, r \in P_D$, $d_R(u,v) \geq d_D(u,v)$ holds. A trivial lower bound of the overall routing cost on R can be given by:

$$
\begin{aligned}
c(R) &= \sum_{u,v \in P_D} d_R(u,v) \\
&\geq \sum_{u,v \in P_D} d_D(u,v)
\end{aligned}
$$



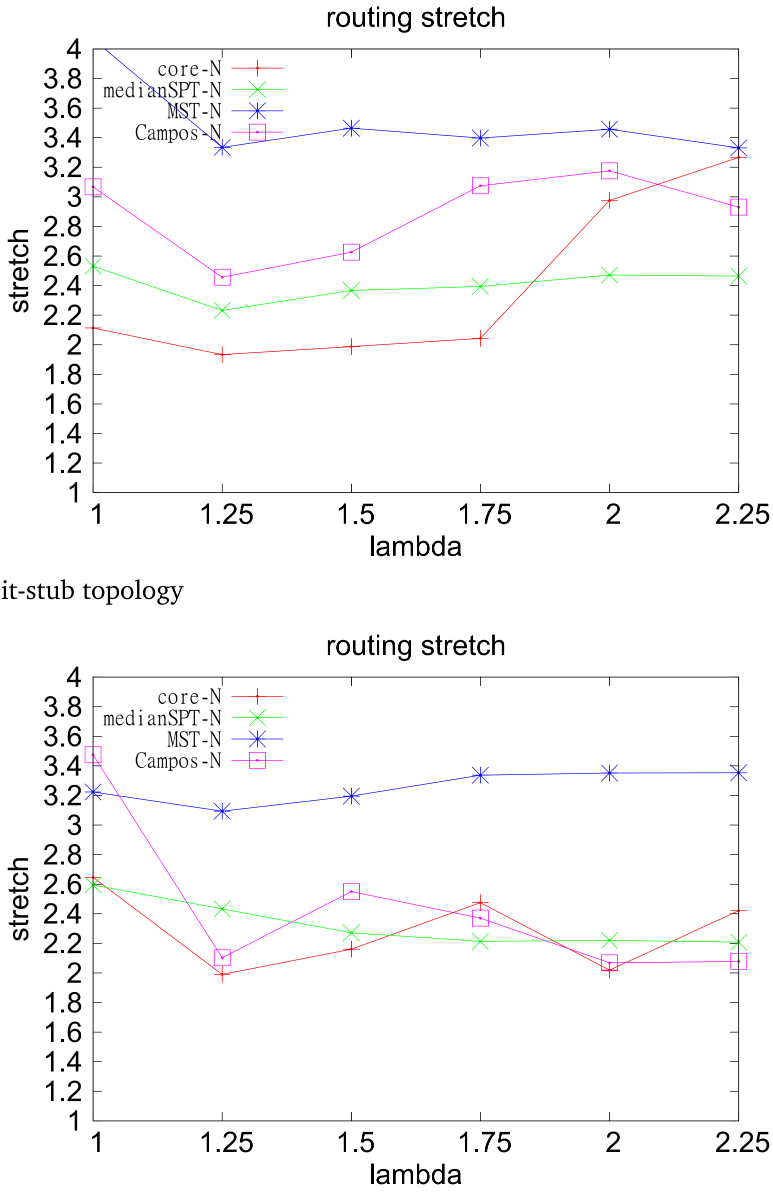
(a) Transit-stub topology

(b) Waxman topology

Figure 7.5: Overlay routing stretch compared to discovery overlay for different λ-values. The maximum number of neighbors is limited to $\sqrt{n}$.

A path on the tree between two peers u, v can generally be divided in at most three sub-paths if the peers are not connected to the same relay node: The path from u to the core, the path on the core, and the path from the core to v. The following holds:

$$d_R(u,v) \leq \begin{cases} d_R(u, rel(u)) + d_R(rel(u), rel(v)) + d_R(rel(v), v), & \text{if } rel(u) \neq rel(v) \\ d_R(u, rel(u)) + d_R(rel(u), v), & \text{else} \end{cases}$$



(a) Transit-stub topology

(b) Waxman topology

Figure 7.6: Overlay routing stretch compared to the underlay network for different λ-values. The maximum number of neighbors is limited to $\sqrt{n}$.

So the case where paths on the core are used determines the upper bound of the distance for any u, v on R. We will denote the tree induced by the relay nodes as T_C. It is a subgraph of R, so the set of edges of a path ρ_R that are part of the core is given by $\rho_R(u, v) \cap E_{T_C}$. Their contribution to the distance is calculated by the summed weight of all edges that are part of the core when using a shortest path from u to v:

$$d_{T_C}(u,v) = \sum_{e \in \rho_R(u,v) \cap E_{T_C}} \omega_D(e)$$

Summing up the upper distance bounds of d_R for all pairs of peers, we get an upper bound of the overall routing cost by [28]:

$$\begin{aligned} c(R) &= \sum_{u,v \in P_D} d_R(u,v) \\ &\leq \sum_{u,v \in P_D} d_R(u, rel(u)) + d_R(rel(u), rel(v)) + d_R(rel(v), v) \\ &\leq \sum_{u,v \in P_D} d_R(u, rel(u)) + d_R(rel(v), v) + \sum_{u,v \in P_D} d_{T_C}(u,v) \\ &\leq 2|P_D| \sum_{u \in P_D} d_R(u, rel(u)) + \sum_{u,v \in P_D} d_{T_C}(u,v) \end{aligned}$$

Consider the first sum-term of that inequality. Since for all non-core peers u the path to $rel(u)$ is the same as the shortest path on the discovery overlay, we can substitute $d_R(u, rel(u))$ in the above inequality by $d_D(u, rel(u))$, giving:

$$c(R) \leq 2|P_D| \textstyle\sum_{u \in P_D} d_D(u, rel(u)) + \sum_{u,v \in P_D} d_{T_C}(u,v)$$

The second sum term, representing the sub-paths through the core, cannot be substituted accordingly, as we do not know the exact structure of the core. But we can derive bounds on the routing load of all edges of the routing overlay $e \in E_R$.

Removing e from E_R would result in two subtrees. The set of vertices would be given by $P'_D(e)$ and $P_D \setminus P'_D(e)$. We set $x = |P'_D(e)|$, then, the routing load of e is given by $ld(e) = 2x(|P_D| - x) = 2|P_D|x - 2x^2$, where x is an integer and $x \in [1, (|P_D| - 1)]$. So, we are able to conclude that:

$$\forall e \in E_R \colon (ld_R(e) \geq 2(|P_D| - 1)) \wedge (ld_R(e) \leq \frac{|P_D|^2}{2})$$

So we can give an upper bound for the contribution of the core by [28]:

$$\begin{aligned} \sum_{u,v \in P_D} d_{T_C}(u,v) &= \sum_{e \in E_{T_C}} ld_R(e)\omega_D(e) \\ &\leq \frac{|P_D|^2}{2} \sum_{e \in E_{T_C}} \omega_D(e) \end{aligned}$$

That enables us to derive the upper bound of the overall routing cost directly by the structure of the topology discover overlay D and the core of the tree $\mathcal{C}$. It is given by [28]:

$$c(R) \leq 2|P_D| \sum_{u \in P_D} d_D(u, rel(u)) + \frac{|P_D|^2}{2} \sum_{e \in E_{T_{\mathcal{C}}}} \omega_D(e)$$

Since the stretch of R on top of N can only be influenced via $c(R)$, the choice of relay nodes forming the core is very important.

Indeed, more explicit bounds could be derived when the size of shortest path trees at the relay nodes would be enforced. Core nodes would need the ability to rearrange the shortest path trees rooted at them by explicitly rejecting or accepting non-core peers. If nodes would be rejected, they had to connect to a different relay node (still through a shortest path to that relay). This would involve much higher configuration overhead.

But, it can be shown that it is at least possible to keep the approximation ratio of the minimum routing cost tree in $[\frac{4}{3}, 2]$ under certain circumstances (see [28]).

8 Conclusion and future work

8.1 Conclusion

We studied the problem of establishing an efficient underlay-aware routing mechanism in the context of event dissemination in content-based publish/subscribe overlays. A multi-overlay approach that divides the problem into sub-problems and conquers them separately was developed. Another benefit of that technique is that single components of the approach are reusable or can be exchanged. Our work leads to multiple contributions:

1. We proposed a new topology inference scheme that generates an accurate representation of the underlying router-level network using a new distributed algorithm. The scheme is configurable via parameters to limit space usage, message overhead and to control its focus on locality which turned out to be important for our routing approach. We showed in simulations that it performs well on different types of Internet-like network topologies and that the emerged peer-to-peer overlay exhibits low stretch. As a side-effect, underlay link stress is reduced. It is possible to reach stretch values less than 7% higher than underlay routing. Even when limiting the local space on each peer to $O(\sqrt{n})$ stretches lower than 1.5 can be reached with about 10% of traceroute executions necessary compared to an n-by-n approach.

2. A new routing mechanism that focuses on reducing overall cost between the peers by approximating an minimum routing cost tree was introduced. The distributed algorithm detects a high-load subgraph of the routing overlay, which we called the *core*. We are able to connect peers through shortest path to the core and showed in simulations that it is possible to achieve reasonable underlay stretch. We compared the results to different other spanning tree algorithms where it performs very well as long as the discovery overlay focuses on locality.

3. We showed how the derived structure can provide underlay-awareness to a content-based publish/subscribe systems that benefits from the focus on routing cost minimization.

8.2 Future work

We focused on end-to-end delay as cost metric for the routing overlay throughout this work. With a slight enhancement, it is possible to add support for all sorts of constraints. Assume, we are also given the requirement $r(p, q)$ for each pair of peers. We redefine the communication cost to be the path length between the two peers on R multiplied by the requirement. The aim is approximating the spanning tree with the minimum communication cost. That problem is called the *optimum communication spanning tree* (OCT) problem, which is a generalization of the MRCT problem [28].

Bibliography

[1] B. Awerbuch, “Complexity of network synchronization,” *J. ACM*, vol. 32, no. 4, pp. 804–823, 1985. (Cited on page 55)

[2] B. Awerbuch, “Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems,” *Proceedings of the nineteenth annual ACM symposium on Theory of computing (STOC ’87)*, 1987. (Cited on pages 49, 54 and 60)

[3] P. Barford, A. Bestavros, J. Byers, and M. Crovella, “On the marginal utility of network topology measurements,” *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement (IMW ’01)*, 2001. (Cited on pages 42, 43 and 44)

[4] A. Broido and K. C. Claffy, “Internet Topology: connectivity of IP graphs,” *Proceedings from the SPIE International Symposium on Convergence of IT and Communication*, pp. 172–187, 2001. (Cited on page 42)

[5] M. Bui, F. Butelle, and C. Lavault, “A distributed algorithm for constructing a minimum diameter spanning tree,” *Journal of Parallel and Distributed Computing*, vol. 64, no. 5, pp. 571–577, 2004. (Cited on page 58)

[6] R. Campos and M. Ricardo, “A fast algorithm for computing minimum routing cost spanning trees,” *Computer Networks*, vol. 52, no. 17, pp. 3229–3247, 2008. (Cited on pages 14, 50, 51 and 61)

[7] K. Chao. (2006) Minimum Routing Cost Spanning Trees. [Online]. Available: http://www.csie.ntu.edu.tw/∼kmchao/tree06spr/mrct.ppt (Cited on pages 5 and 50)

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos, “On power-law relationships of the Internet topology,” *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM’99)*, pp. 251–262, 1999. (Cited on page 30)

[9] R. G. Gallager, P. A. Humblet, and P. M. Spira, “A Distributed Algorithm for Minimum-Weight Spanning Trees,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 5, no. 1, pp. 66–77, 1983. (Cited on page 49)

[10] J. A. Garay, S. Kutten, and D. Peleg, “A SubLinear Time Distributed Algorithm for Minimum-Weight Spanning Trees,” *SIAM Journal on Computing*, vol. 27, no. 1, p. 302, 1998. (Cited on page 60)

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, ser. Series of Books in the Mathematical Sciences. W.H.Freeman & Co, 1979. (Cited on pages 13 and 50)

[12] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann, "Self-organizing broker topologies for publish/subscribe systems," *Proceedings of the 2007 ACM symposium on Applied computing (SAC '07)*, 2007. (Cited on page 11)

[13] X. Jin, W. Tu, and S.-H. G. Chan, "Scalable and Efficient End-to-End Network Topology Inference," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 6, pp. 837–850, 2008. (Cited on page 13)

[14] D. S. Johnson, J. K. Lenstra, and A. H. G. Rinnooy Kan, "The complexity of the network design problem," *Networks*, vol. 8, no. 4, pp. 279–285, 1978. (Cited on page 13)

[15] E. Korach, D. Rotem, and N. Santoro, "Distributed algorithms for finding centers and medians in networks," *ACM Trans. Program. Lang. Syst*, vol. 6, no. 3, pp. 380–401, 1984. (Cited on page 51)

[16] M. Kwon and S. Fahmy, "Path-aware overlay multicast," *Computer Networks*, vol. 47, no. 1, pp. 23–45, 2005. (Cited on pages 13 and 20)

[17] C. P. Low and X. Song, "On Finding Feasible Solutions for the Delay Constrained Group Multicast Routing Problem," *IEEE Transactions on Computers*, vol. 51, no. 5, pp. 581–588, 2002. (Cited on page 12)

[18] A. Majumder, N. Shrivastava, R. Rastogi, and A. Srinivasan, "Scalable Content-Based Routing in Pub/Sub Systems," *Proceedings of the 28th Conference on Computer Communications (INFOCOM 2009)*, pp. 567–575, 2009. (Cited on page 11)

[19] A. Medina, A. Lakhina, I. Matta, and J. Byers. (2001) BRITE: Universal Topology Generation from a User's Perspective. [Online]. Available: http://www.cs.bu.edu/brite/publications/usermanual.pdf (Cited on page 30)

[20] A. Montresor and M. Jelasity, "PeerSim: A Scalable P2P Simulator," *Proceedings of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pp. 99–100, 2009. (Cited on page 31)

[21] G. Parmer, R. West, and G. Fry. (2006) Scalable Overlay Multicast Tree Construction for QoS-Constrained Media Streaming. [Online]. Available: http://cs-pub.bu.edu/faculty/richwest/papers/TR-2006-020.pdf (Cited on page 12)

[22] M. B. Sharma, J. Chen, and S. Iyengar, "Distributed algorithms for locating centers and medians in communication networks," *Proceedings of the 1992 ACM/SIGAPP symposium on Applied computing (SAC'92)*, pp. 808–817, 1992. (Cited on page 51)

[23] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos, "Power laws and the AS-level internet topology," *IEEE/ACM Trans. Netw*, vol. 11, no. 4, pp. 514–524, 2003. (Cited on page 30)

[24] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. (2002) Network Topology Generators: Degree-Based vs. Structural: Technical Report 02-760. [Online]. Available: http://www.cs.usc.edu/research/02-760.pdf (Cited on page 30)

[25] M. A. Tariq, G. G. Koch, B. Koldehofe, I. Khan, and K. Rothermel, “Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints,” *Proceedings of the Sixteenth International Conference on Parallel Computing (EURO-PAR)*, 2010. (Cited on page 11)

[26] B. M. Waxman, “Routing of Multipoint Connections,” *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, 1988. (Cited on page 30)

[27] R. T. Wong, “Worst-Case Analysis of Network Design Problem Heuristics,” *SIAM Journal on Algebraic and Discrete Methods*, vol. 1, pp. 51–63, 1980. (Cited on page 50)

[28] B. Y. Wu and K. M. Chao, *Spanning trees and optimization problems*. Boca Raton, FL: Chapman & Hall/CRC, 2004. (Cited on pages 13, 14, 50, 52, 54, 64, 67, 68 and 70)

[29] B. Y. Wu, K. M. Chao, and C. Y. Tang, “Approximation algorithms for the shortest total path length spanning tree problem,” *Discrete Applied Mathematics*, vol. 105, no. 1-3, pp. 273–289, 2000. (Cited on pages 13 and 64)

[30] E. W. Zegura, “Modeling Internet Topology,” 2002. (Cited on page 30)

[31] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, “How to Model an Internetwork,” *Proceedings of the Fifteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1996)*, pp. 594–602, 1996. (Cited on page 30)

[32] E. Zegura. (2000) Modeling Topology of Large Internetworks. [Online]. Available: http://www.cc.gatech.edu/projects/gtitm/ (Cited on page 30)

[33] Y. Zhu and B. Li, “Correlation-Aware Multimedia Content Distribution in Overlay Networks,” *Proceedings of the Thirteenth Annual SPIE/ACM Conference on Multimedia Computing and Networking (MMCN 2006)*, pp. 127–138, 2006. (Cited on page 12)

[34] Y. Zhu and B. Li, “Overlay Networks with Linear Capacity Constraints,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 2, pp. 159–173, 2008. (Cited on page 12)

All links were last followed on August 3, 2011.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Christian Schieberle)