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

Diplomarbeit Nr. 3151

Underlay Aware Approach to Provide Reliable and Timely Dissemination of Events in a Publish Subscribe System

Tom Quist

Course of Study:	Software Engineering
Examiner:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Supervisor:	M. Sc. Muhammad Adnan Tariq
Commenced:	April 01, 2011
Completed:	August 31, 2011
CR-Classification:	C.2.4, C.2.1, C.2.2

Contents

List of Figures

List of Algorithms

Nomenclature

API Application Programming Interface

ARQ Automatic Repeat reQuest

AS Autonomous System

BRITE Boston university Representative Internet Topology gEnerator

DHT Distributed Hash Table

GHS Gallager-Humblet-Spira Algorithm (Distributed Minimum Spanning Tree algorithm)

GT-ITM Georgia Tech - Internetwork Topology Models

IP Internet Protocol

LAN Local Area Network

MRST Maximum-Reliability-Spanning-Tree

MST Minimum Spanning Tree

mwoe Minimum Weight Outgoing Edge

P2P Peer-to-Peer

QoS Quality-of-Service

RTT Round-Trip-Time

TDO Topology-Discovery-Overlay

TTL Time-To-Live

Abstract

Publish-subscribe is a well-known paradigm for building distributed applications. Events produced by peers, called publishers, are disseminated to interested consumers, called subscribers. Usually publishers and subscribers are arranged in a peer-to-peer overlay network, which helps in dissemination of events in a decentralised manner. Recent research tries to provide Quality-of-Service like delay bounds or reliability in such a system. In order to provide reliability current distributed publish-subscribe systems mostly either rely on overlay level acknowledgement protocols or try to find multiple disjoint paths in the overlay to increase redundancy without taking into account the underlay topology. Acknowledgements induce high delays affecting timeliness of event delivery. Providing multiple paths without looking at the underlay does not take into account correlations between paths within the underlay. We address these drawbacks by designing a content-based publish-subscribe system which provides reliability by taking into account the underlay topology to reduce correlations within the underlay in overlay links. The system consists of three layers: The Topology-Discovery-Overlay (TDO) layer constructs an underlay topology aware overlay which reflects the underlay topology by using a path-matching algorithm. On top of the TDO the Maximum-Reliability-Spanning-Tree (MRST) layer constructs *k* overlay link disjoint trees which contain the most reliable overlay links. The MRSTs are used by the content-based publish-subscribe layer for subscription flooding and event forwarding. The system has been evaluated by simulations in PeerSim using Internet-like topologies. The results show that the TDO discovers most of the underlay topology and constructs overlay topologies reflecting the underlay topology. Simulations also show that the system converges towards a maximum event delivery probability.

1 Introduction

Traditional communication paradigms like *client-server* where information must be specifically requested at the source or at a centralised database lack in scalability. An asynchronous communication paradigm like publish-subscribe allows for better scalability. Information in the form of *events* produced by so-called *publishers* is delivered to interested participants called *subscribers*. Publish-subscribe can allow for greater scalability because publishers and subscribers are decoupled and events are pushed to subscribers rather than subscribers need to poll this information.

In Figure 1.1 the publish-subscribe API model is shown. The *EventService* is the main publish-subscribe middleware interface and provides several services which can be used by applications. Subscribers can subscribe and unsubscribe to specific events expressed by a *subscription*. Publishers can publish events to be consumed by all subscribers having a matching subscription. In some publish-subscribe systems publishers can also advertise and unadvertise the events they intend to publish. Advertisements are usually expressed using the subscription model.

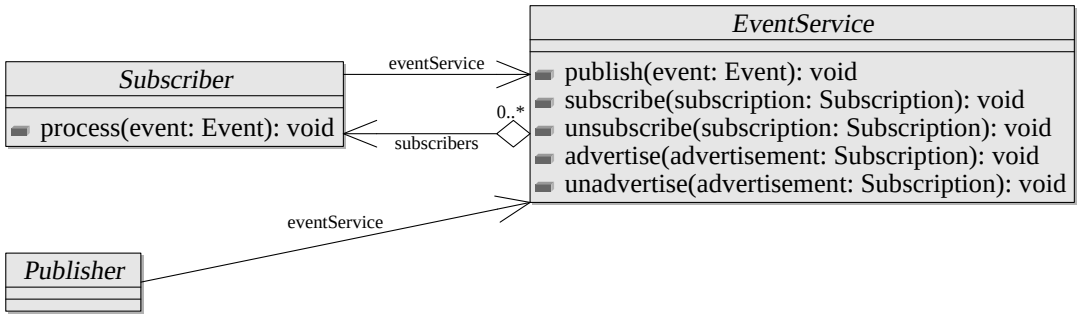


Figure 1.1: Publish-Subscribe API Model

Publish-subscribe systems can be categorised into *Topic-Based* and *Content-Based* systems:

Topic-Based A topic-based system categorises events into topics. Subscribers subscribe to a specific topic and receive all events published to this topic. However, the main drawback of the topic-based subscription model is the limited expressiveness. Several improvements for expressiveness are possible. Often the model is extended to be hierarchical i.e. a topic can have multiple sub-topics. A message published to a topic is then delivered to subscribers to both, the topic and all its parent topics. In many cases operators like wildcards can be used to subscribe to multiple topics using one subscription (cf. [She10]).

Content-Based The content-based model overcomes the limited expressiveness of the topic-based model. As the name implies, subscriptions in a content-based system describe complex filtering

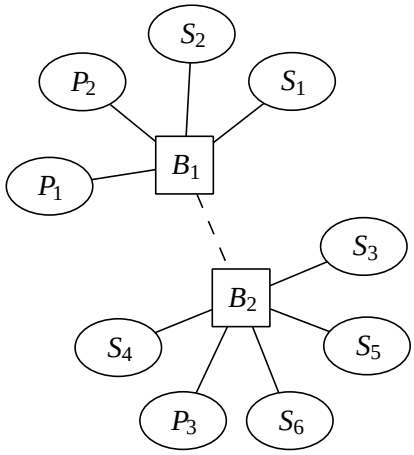


Figure 1.2: Publish-Subscribe Broker Network

criteria on the content of events. The expressiveness depends on the subscription language defined by the specific system. Usually subscriptions are defined as sets of constraints composed of disjunction or conjunction. Most subscription languages allow equality (“ $=$ ”) and comparison (“ $>$ ”,“ $<$ ”,“ $\in [min;max]$”,“ $\in [min;max[$”,“ $\in]min;max]$”,“ $\in]min;max[$”) operators for constraints, some also allow regular expressions. The following example shows a subscription that matches all events where the location is “Stuttgart", the temperature is within 10 and 20 and the pressure is above 1000: $location =$ “*Stuttgart*” $\wedge\, temperature \in [10,20] \wedge pressure > 1000$ (cf. [She10]).

In general an increase in expressiveness affects scalability of the system. However, because content-based systems are much more expressive many systems in literature try to make use of that model.

Traditionally, publish-subscribe systems consist of a set of dedicated servers, called *brokers* that provide a publish-subscribe interface. Clients connect with brokers to subscribe to or to publish events. Brokers form a *broker network* where events are routed along to other brokers having subscribers with matching subscriptions.

However, broker networks are usually not self-organizing and managed manually by administrators. This is the reason why most of recent research concentrates on peer-to-peer (P2P) publish-subscribe systems. In a P2P publish-subscribe system every subscriber not only receives messages but also participates in event forwarding. Also, potentially every peer can be a publisher. Because of this flexibility this thesis will focus on a P2P publish-subscribe system.

In order to deliver events to subscribers with a matching subscription in a P2P publish-subscribe system, subscriptions are usually flooded across the peer network to set up a reverse path for events (event routing table). However, flooding introduces high message complexity which can be reduced by taking into account covering relations among subscriptions. For example if a subscription defines to receive only events where $x > \alpha$, it is unnecessary to forward a second subscription on the same path which defines $x > \beta, \alpha \leq \beta$. The routing table constructed while flooding subscriptions is used to forward events, the reverse path of the subscription.

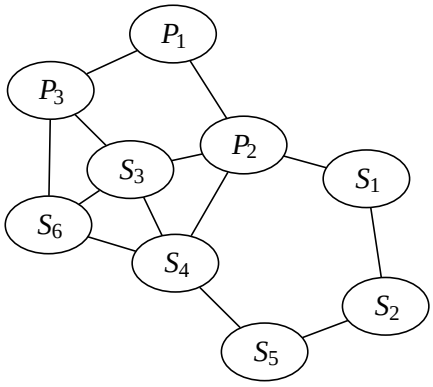


Figure 1.3: P2P Publish-Subscribe

Figure 1.2 shows an example for a broker-network consisting of two brokers B_1 and B_2. Each broker has some connected publishers P_i and subscribers S_i. If for example publisher P_1 publishes an event which matches the subscriptions of subscribers S_1 and S_4, the event is first sent to broker B_1, who then forwards it to B_2 and delivers the event to S_1. B_2 then delivers the event to S_4.

In contrast Figure 1.3 shows an example for a P2P publish-subscribe system with publishers P_i and subscribers S_i. In this case an event published by P_1 which matches a subscription of S_1 and S_4 is forwarded by other publishers and subscribers to S_1 and S_4. For example P_1 forwards the event to P_2, who then delivers the event to S_1 and S_4 but other paths may also be possible depending on the publish-subscribe routing algorithm.

Recent research addresses Quality-of-Service (QoS) in publish-subscribe systems. QoS refers to the ability of a system to provide services satisfying user expectations for all non-functional criteria. QoS metrics in a publish-subscribe system can be minimum bandwidth requirements, order of event notification or delay bound requirements for event delivery. For example in [TKK+11] a system which meets subscriber defined delay bounds was proposed (discussed in Section 2.1.1).

Providing reliability in a publish-subscribe system is another challenging problem in this field. Reliable event delivery is important for critical events, especially. Providing reliability in a publish-subscribe system, however, must be considered a complex task. Publishers usually do not know the set of subscribers, thus making it difficult to meet reliability requirements because paths to subscribers and their quality are also unknown.

In order to provide reliable event dissemination in a publish-subscribe system, most of the existing solutions introduce reliability mechanisms like overlay level acknowledgements or add redundancy like providing multiple redundant overlay paths [GPS04, MB07, TSN10]. On the other hand, overlay level acknowledgements introduce substantial delay in event delivery due to retransmits. By only providing multiple redundant overlay paths without looking at the underlay, correlations in the underlay are neglected. Usually an overlay link is mapped to an underlay path determined by the underlying routing protocol. An underlay path is composed of a sequence of physical links in the underlay. When two overlay paths share a common physical underlay link, properties of this link are shared by both overlay paths.

This is a reason why recent research also considers the underlay network topology when providing QoS constraints in a publish-subscribe system. However, most solutions concentrate on broker networks which have some drawbacks as mentioned previously [KB07, ECG09].

Admittedly, only looking at the underlay is not enough. Peers in the overlay may fail at any time, which can cause losses while forwarding events in the overlay.

This thesis develops an approach for a P2P publish-subscribe system. It takes into account correlations in the underlay topology in order to reduce correlations in underlay paths. To overcome failures of overlay peers, the system establishes redundant overlay paths for message diffusion.

The thesis is outlined as follows: In Chapter 2 related work in the field of QoS for publish-subscribe systems and topology discovery will be discussed. In Chapter 3 the system model will be explained and the problem to be solved will be described. In Chapter 4 an overview on the approach developed in this thesis and the different protocol layers will be given. The topology discovery layer will be described in Chapter 5 whereas the layer providing a publish-subscribe interface will be explained in Chapter 6. All evaluation results will be presented and discussed in Chapter 7. Finally, a conclusion will be drawn in Chapter 8 and future work in this topic will be discussed.

2 Related Work

2.1 Publish-Subscribe System

Providing QoS in a P2P publish subscribe system is a complex task. Publishers usually do not know the complete set of subscribers and their location in the network when publishing events, which makes it difficult to adapt event forwarding according to QoS requirements. This section will discuss related work about providing QoS in publish-subscribe systems.

2.1.1 Meeting Subscriber-Defined QoS Constraints in Publish-Subscribe Systems

An approach to meet subscriber-defined QoS constraints in a broker-less publish-subscribe system has been proposed by Tariq et al. [TKK+11]. Subscribers can define individual delay requirements for their subscriptions in the presence of bandwidth constraints. Subscribers construct overlays which maintain the delay bounds in a way that subscribers with tight delay requirements are served before subscribers with more lenient requirements. The system relies on advertisements and defines a subscription model based on spatial indexing where the event space, which is composed of a globally known ordered set of d distinct attributes and modelled geometrically as a d-dimensional space, is divided into sub-spaces which are created by recursive binary decomposition of the event-space. Subscriptions are expressed by *dz-expressions* which are bit-strings of "0"s and "1"s. When a sub-space is divided, its dz-expression is used as prefix to the newly created sub-space (Figure 2.1). The system consists of two levels of subscriptions, namely the *user-level* and the *peer-level* subscriptions. User-level subscriptions represent the original subscriptions as defined by the application and are mapped to an approximated peer-level subscription which is a set of dz-expressions $DZ(p) = \{dz_i | i \geq 1\}$.

In the following, the system will be described briefly. Each subscriber maintains a peer view *pView* that caches information about peers which are relevant because they have covering subscriptions. The *pView* is maintained by a modified version of an epidemic protocol. Periodically, each peer checks whether each dz_i in $DZ(p)$ is covered by a subscription of a subscriber or by all relevant publishers the peer is connected to. If a dz-expression is not covered, the peer selects a suitable parent from *pView* and sends a connection request to the potential parent. As soon as the parent receives the connection request from the peer it acknowledges the request if the delay requirement of the peer's subscription is less tight than its own delay requirement. If not, it sends a hint about the most suitable parent, according to its knowledge, which the peer will add to its *pView* and take into account as a potential parent in its next iteration. To prevent cycles when accepting connections, a message to detect the cycle is sent to the parent and forwarded along all peers having the same subscription and delay constraints. When the message is received by the originator, a cycle is detected and removed. Sometimes delay requirements cannot be satisfied. In this case peer-level subscriptions can be coarsened according to

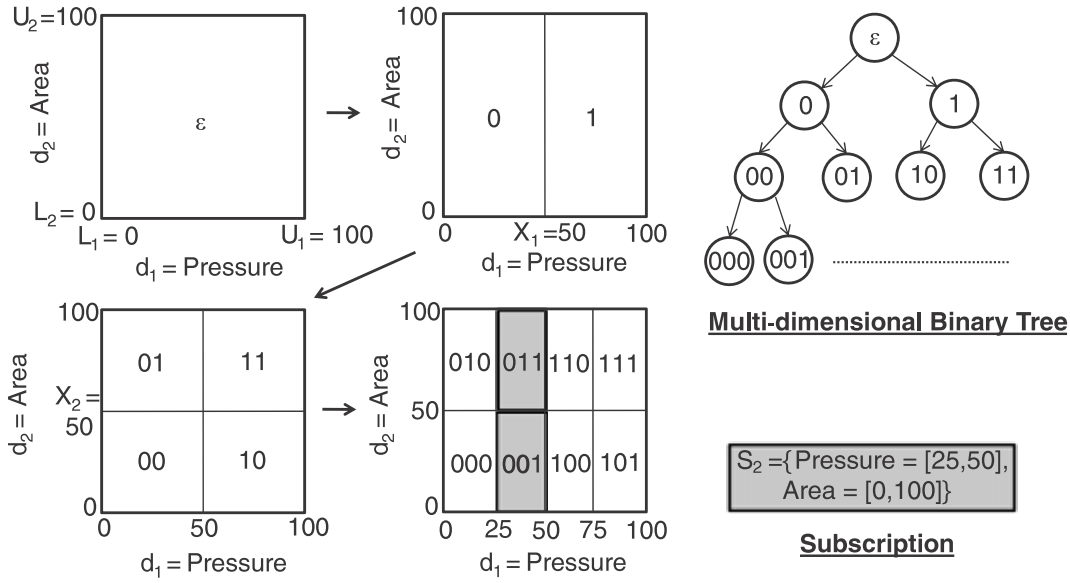


Figure 2.1: Spatial indexing (Source: [TKK+11])

bandwidth constraints. Because less selective subscriptions are placed higher (nearer to publishers) in the dissemination graph, delay requirements will most likely be held at the cost of bandwidth.

The proposed system allows for subscriber-defined delay bounds. It scales up to a large number of subscribers and performs robustly. However, the subscription model only allows for a closed set of *d* globally known attributes which has a negative influence on the expressiveness. Concerning reliability it is difficult to convert the delay requirements into reliability requirements because of the different nature of reliability. I.e. delay is an additive metric whereas reliability is a multiplicative metric. To introduce reliability in such a system, additional mechanisms have to be implemented, such as connecting to *k* different parents instead of only one.

2.1.2 Reliable Publish-Subscribe Middleware for Time-Sensitive Internet-Scale Applications

Esposito et al. [ECG09] gives an overview of the state of the art publish-subscribe systems as far as reliability is concerned. They introduce the taxonomy of failures in publish-subscribe middleware. Several kinds of failures may affect a publish-subscribe system. These faults can be classified as follows:

Network anomalies Temporary misbehaviour of a link

Link crash Links experience loss of connectivity, i.e. packets are always lost in a certain period of time which is not necessarily permanent but may dynamically appear and disappear

Node crash Nodes crash due to failure of hardware/software

Churn Nodes unexpectedly join/leave the system

Network anomalies can be further broken down into the following:

Loss Links behave as a fairy-loss channel, i.e. messages in transit through the link may be lost randomly

Ordering Messages are not received in the same order as they have been published

Corruption Messages are corrupted

Delay A message is delivered later than expected

Congestion A link/router is overloaded, suddenly causing several anomalies

Partitioning Network may get fragmented into several disconnected parts

Link and node crashes can appear in both, the underlay and the overlay. A node crash in the underlay usually refers to a crashed router whereas a node crash in the overlay refers to a crashed peer. A link crash in the overlay, however, is usually caused by a link crash in the underlay, because overlay links are composed of a series of underlay links and nodes.

Besides, a classification of the current approaches adopted in literature to implement reliable publish-subscribe services is given:

Retransmission (ARQ) A common approach to provide reliable communication is to retransmit the message when a loss is detected. This can be initiated by the sender, for example by using acknowledgements (ACK). When the sender does not receive an ACK-Message from the receiver within a given period of time, the sender retransmits the message. Another possibility is to detect the loss on the receiver side, for example by using package-sequence-numbers. In general this approach is not optimal. The approach can only deal with lost messages and does not specifically handle node failures. In case of a link crash which partially or completely disconnects a subset of subscribers from the publisher, there are no guarantees to achieve agreement. Moreover, the time to recover dropped messages is unpredictable. Another drawback is that messages can be received twice due to false-negative loss detection.

Forward Error Correction Forward Error Correction adds redundancy to the messages so that receivers can reconstruct lost messages in case that some packets have been dropped. The main drawback is that the degree of recoverable messages depends on the amount of redundancy added to the message. However, choosing the right redundancy is a difficult task in real world Internet since losses are highly variable over time.

Epidemic Algorithm Epidemic algorithms distribute the recovery responsibility among the leaf nodes of the forwarding tree. Nodes exchange their history of received messages with a random neighbour. By comparing their own message history with the received histories, message drops are detected and a retransmission can be requested. This system avoids duplicate messages due to false-negative time-outs and can also recover lost messages due to crashed nodes or links which is not the case for basic ARQ. However, because of the probabilistic nature of epidemic algorithms there is no guarantee that a message reaches all subscribers.

Reconfiguration Topological reconfiguration enables a system to recover connectivity in the forwarding tree[1] after a node or link has crashed. For example a soft state approach can be used where routing entries expire if they have not been refreshed within a given time-to-live. The approach aims to guarantee a consistent connectivity for the system but it does not cope with message drops.

Broker Replication Brokers in an event forwarding tree are replicated. In case of a broker failure the state of the failing broker can be easily substituted by a replication broker without losing subscription consistency. The solution, however, only considers node failures but link crashes may cause message losses or some subscribers may not be reachable any more.

Path redundancy By adopting path redundancy a system establishes redundant paths among the nodes of the system. Messages are sent through multiple paths so that in case of a link failure on one of the paths, another path may still be available. Paths, however, should be diverse to avoid the situation where a single failure of a component within one path affects all the other paths as well. This approach circumvents link crashes but can also cope with node crashes.

The approach described in the paper to provide reliable and timely event dissemination in a topic-based publish-subscribe system exploits the fact that the Internet is composed of interconnected *Routing Domains*. These domains may consist of *Local Area Networks* (LAN) or *Autonomous Systems* (AS). The system uses a hierarchical approach which reflects the consideration on the Internet topology. Nodes in the same domain are clustered together and each cluster holds a coordinator that allows interaction with other clusters. Nodes in the same cluster use *IP Multicast* for intra-cluster communication. Because such domains are usually highly reliable it is assumed that timely and reliable data delivery within clusters is guaranteed. Clusters are connected to other clusters through their coordinators. A non-coordinator node hence never directly communicates with a node from another cluster but uses its coordinator as a proxy. For inter-cluster communication a tree-based peer-to-peer application-level multicast solution is used which is built on top of a structured or *Distributed Hash Table* (DHT) overlay. In case of a DHT, *Pastry*[2] is used as a routing substrate. For each topic at intra-cluster level an IP Multicast group is built, whereas at inter-cluster level an application-level multicast dissemination tree is built. If a coordinator wants to join an existing multicast tree because one of the peers in its cluster or itself subscribed to a specific topic, it sends a join message to the root of the appropriate multicast tree using the overlay routing substrate. If a coordinator along the way is already in the tree, it registers the joining coordinator as a child. A message which is published by a node first is sent to all interested nodes in the same cluster and to the coordinator using IP Multicast. Then the coordinator passes it towards the root of the application level multicast tree which then passes the message along the tree. Each coordinator receiving the message performs an IP Multicast in their cluster and then forwards the message to their children.

The system uses a combination of *Path Redundancy* and *Broker Replication*, which handles both, node failures and link crashes.

At cluster level the system is vulnerable when a coordinator fails. This is the reason why a hybrid replication scheme is used, where a cluster has p redundant coordinators which all have equal respon-

[1] Tree a message is forwarded along towards the destinations where the root of the tree is the sender of the message
[2] Pastry is an overlay routing substrate based on a Distributed Hash Table

sibilities. Since there are several coordinators present at the same time, there is always at least another coordinator available in case of a crashed coordinator, which achieves timeliness. Additionally, each coordinator has *k* back-ups which replace a coordinator after it has failed.

At inter-cluster level the system is vulnerable to losses due to link crashes. To overcome this situation, the system uses a multiple-tree approach where for each topic *p* trees are built. An underlay link crash on an overlay link on one of the trees should not affect an overlay link on the other tree. This is the reason why paths should be diverse. [ECG09] introduces a measure for reciprocal diversity of paths, namely $Q(P_1,P_2)$ which is the number of overlapping components within the two paths P_1 and P_2. Using this measure, different formulations of path diversity can be given. The *Global Diversity* is defined as follows: A forest F of n trees, namely T_i with $i = 1,\ldots,n$, verifies the global diversity constraint, if and only if it is not possible to find two paths that exhibit a positive value as measure of their reciprocal diversity in any of the n trees:

$$F = \bigcup_{i=1,\ldots,n} (T_i) : F \text{ is diverse } \Leftrightarrow \nexists P_i, P_j \in F : (Q(P_i,P_j) > 0) \wedge (i \neq j)$$

However, because it is impossible to verify whether trees satisfy this condition a weakened constraint has been specified (namely *Local Diversity*) which verifies the path diversity constraint if and only if all the paths from a specific node N_A to its parents and children in the i^{th} tree, namely $P_{i|N_A}$ do not exhibit a positive value as measure of their reciprocal diversity:

$$F = \bigcup_{i=1,\ldots,n} (T_i) : F \text{ is diverse } \Leftrightarrow \forall N_A \nexists P_{i|N_A}, P_{j|N_A} : (Q(P_{i|N_A}, P_{j|N_A}) > 0) \wedge (i \neq j)$$

Local Diversity does not imply Global Diversity but it makes it possible to implement a distributed algorithm that is able to construct locally-diverse multiple trees.

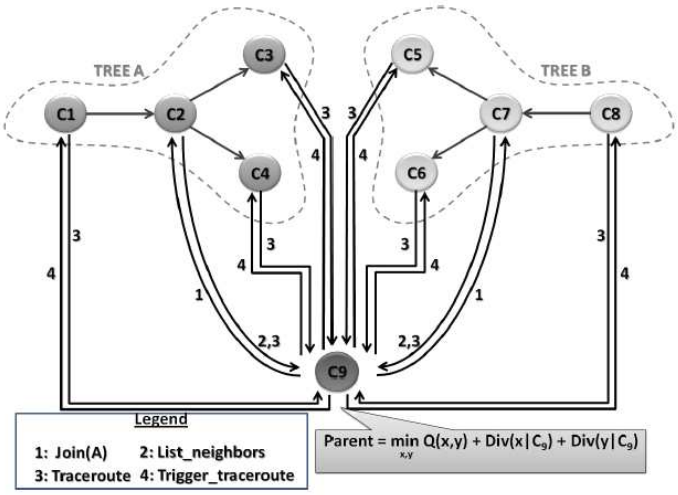


Figure 2.2: A node joining two distinct trees (Source: [ECG09])

How a node joins multiple distinct tree is explained based on the example in Figure 2.2. Node C_9 wants to join two different trees *A* and *B*. Initially C_9 sends a join message towards the root of each

tree. The message is intercepted by C_2 and C_7. These two nodes reply with a message containing a list of their neighbours and details about the path to them and a traceroute[3] result to C_9. Then C_9 contacts all nodes of the received list and receives traceroute messages about the path to them, too. Now C_9 can decide which node will be its parent in each tree by determining the set of parents that minimises the diversity of the paths to its parents and the diversity of the paths from the new parent to its children.

The approach provides a system for reliable event delivery through replication and diverse path redundancy. However, the system assumes the availability of IP Multicast within the same cluster. While this is applicable for clusters within the same LAN, it is unusual that IP Multicast is available in Autonomous Systems. At inter-cluster level the system tries to minimise the path diversity to the stage where a node joins an application level multicast tree. However, this may not lead to an optimal solution and the system does not restructure the tree to further optimise the diversity after the inclusion of the new node. Another drawback is that the system only allows for topic-based publish subscribe systems because subscriptions have to be mapped to hash-values to work on top of a DHT.

2.1.3 A distributed algorithm for underlay aware and available overlay formation in event broker networks

Kumar and Bellur [KB07, KB08] proposed an underlay aware overlay formation algorithm for event broker networks. It forms a network having k paths which are node disjoint in the underlay. Therefore two availability models have been defined, namely the *Manifest Availability Model* and the *Latent Availability Model*. The Manifest Availability Model states that distinct paths at the overlay level are node disjoint in the underlay as well whereas in the Latent Availability Model any two overlay nodes have a guaranteed number of node-disjoint paths between them in the underlay. In general both models are NP-complete but under a set of practical constraints, constructing a latent availability overlay reduces the complexity to a polynomial time problem.

The overlay formation algorithm requires underlay quality awareness, which means the nodes gather and store information about the underlying path for the overlay links originating from the nodes, including information about the routers in the path and the node overlap in the overlay link from the node. Nodes are classified into four types:

Overlay nodes Nodes selected to be brokers

Expander nodes Non-Broker nodes with node-degree more than 2

Connector nodes Nodes with degree equal to 2

Trivial or client nodes Nodes with degree of 1

Because the system can only run on brokers which have a physical degree of at least k, the paper only described an overlay formation algorithm for the case $k = 2$ but the concept of the algorithm can also be used for $k > 2$. The algorithm is based on the expansion lemma of Whitney's theorem which states that if G is a k connected undirected graph, then a graph G' obtained by adding a new node x and

[3]Tool to discover the routers on an IP network path from a specific node to an arbitrary destination

adding k distinct edges from x to k distinct nodes of G, is also k-connected. The algorithm is described briefly in the following:

Initially two broker nodes are selected manually as *stellar nodes* with two node disjoint physical paths between them. The path information of the paths between the stellar nodes is gathered by the messages in the algorithm as they proceed from node to node. Stellar nodes remain a part of the system as long as the broker network exists. When a new broker joins the system it sends a message through all its links towards the stellar nodes. The stellar nodes reply with the path the message took as well as with suggested alternatives. The joining node tries to find k node disjoined paths in the set of received paths, if no such set of paths can be found, it tries to contact brokers through the suggested paths. Intermediate brokers intercepting join messages do not forward the message, but instantly reply in the same way as a stellar node.

The proposed overlay formation algorithm forms overlay networks with k node disjoint paths between any pair of nodes. However, the authors make assumptions to the underlay that usually do not hold in real world networks, for example overlay nodes can specify the underlay path an individual message should take. On the Internet this is usually controlled by the underlay routing protocol and not under the sending nodes control. Also, it assumes that the broker nodes take part in underlay routing and hence is able to intercept messages in transit to other broker nodes. Another problem is that brokers not having a degree of k or brokers where no k disjoint paths to the network exist, cannot be part of the system at all. One could argue that allowing nodes with a degree lower than k to be part of the system decreases the total availability. This may be a feasible constraint for managed broker networks where administrators can change conditions manually. For unmanaged distributed P2P publish-subscribe systems, however, this is not practical.

2.1.4 Reliable Routing of Event Notifications over P2P Overlay Routing Substrate in Event Based Middleware

A broker-based publish-subscribe system to provide subscriber defined reliability has been proposed by Mahambre and Bellur [MB07]. The algorithm tries to determine a path which has a reliability value greater than or equal to the threshold specified by the subscriber. It leverages the Pastry overlay routing substrate to route events to subscribers based on a multiplicative reliability measurement model. This model considers a path of an event notification to be a series system where all components (brokers) are so interrelated that the entire system fails if any one of its components fail:

The probability that a node i will drop a packet is given by $P(\omega_i)$ and the probability that the link $l_{i,j}$ (link between node i and j) will drop a packet is given by $P(\phi_i)$, hence the reliability of the node and the link is $(1-P(\omega_i))$ and $(1-P(\phi_i))$, respectively. The reliability of an entire path is given by:

$$\prod_{i=1}^{N}(1-P(\omega_i))(1-P(\phi_i)) \tag{2.1.1}$$

To find a path that meets the subscriber defined reliability, the event publisher determines the primary path for event delivery. The subscriber then returns its identifier and reliability value of the primary path. Now the publisher is aware of all subscribers. If the primary path does not meet the reliability

threshold, a pruning algorithm is being executed, which performs partial flooding up to a particular level. While (partially) flooding the network, the algorithm calculates the partial path reliability[4] using equation (2.1.1). Circles can be avoided because the algorithm forwards the partial path established so far while flooding. Flooding is stopped when a previously defined TTL is reached. Once the flooding is stopped, the message is routed to the destination using Pastry routing. The destination then sends all the discovered paths along with their reliabilities to the publisher. Now the publisher can select a path having a reliability higher than or equal to the threshold reliability requirement specified by the subscriber. If no such single path exists, the publisher combines multiple paths such that reliability of combined paths is greater than or equal to the threshold. While combining paths, two cases arise:

1. Paths are **disjoint**, which means no two paths share a component. If the reliability of path Θ_i is defined by $R(\Theta_i)$ and there are N paths, the combined reliability is defined by

$$1 - (\prod_{i=1}^{N}(1 - R(\Theta_i)))$$

2. Paths are **intersecting**, which means subsets of paths intersect with each other. Given $R(\Theta_i)$ is the reliability of path Θ_i and there are N paths, the combined reliability is defined by

$$\sum_i R(\Theta_i) - \sum_i \sum_{j>i} R(\Theta_i \cap \Theta_j) + \sum_i \sum_{j>i} \sum_{k>j} R(\Theta_i \cap \Theta_j \cap \Theta_k) - \cdots + (-1)^{N+1} R(\Theta_1 \cap \Theta_2 \cap \ldots \cap \Theta_N)$$

Using these definitions, the publisher can find a set of paths meeting the required reliability. The system however relies on advertisements so that publishers can proactively establish paths. Another problem is that the underlay is completely ignored, which may lead to a suboptimal selection of paths with high underlay link correlation. Furthermore, the system uses partial flooding to discover additional paths to subscribers. However, flooding leads to a large number of messages within the system.

2.1.5 Topology-Aware Reliable Overlay Multipath (TAROM)

TAROM [TM05] is an approach to select high quality path pairs by exploiting knowledge about the physical topology and inferred link quality information. It adopts an active-path-first strategy which reactively finds a secondary overlay path that minimises the joint failure probability for a given primary overlay path. TAROM assumes the availability of underlay topology information at end nodes. Furthermore, each node knows the failure-probability of the links to its neighbours, which are recorded from probes. Additionally, each node maintains a local routing table to identify a next hop for any destination node. The paper specifies a reliability model to calculate the joint reliability of a set of paths using information about the path composition and the failure-probability of physical path segments. To get the physical path segment failure-probability it uses the random sampling approach proposed by Padmanabhan et al. [PQW03].

[4] Reliability of the path from the source node till the current node

The reliability model is presented in the following: A multipath between source node s and destination node t is defined as a union of multiple overlay paths $M(s,t) = \bigcup_i O_i(s,t)$ where $O_i(s,t)$ is the i^{th} overlay path between s and t. An overlay path consists of a union of overlay links $O_i(s,t) = \bigcup_j L_{i,j}$ where $L_{i,j}$ is the j^{th} overlay link of the i^{th} overlay path. Moreover, each overlay link consists of at least one physical path segment $S_{i,j,k}$, which is the k^{th} path segment of the j^{th} overlay link of the i^{th} overlay path. In sum a multipath can be modelled as $M(s,t) = \bigcup_i \bigcup_j \bigcup_k S_{i,j,k}$. Given the reliability function $r(x)$ which is the probability that no component in entity x is failed, we can express the reliability of a multipath according to reliability theory as the sum of the overlay path reliabilities minus the reliabilities that are counted twice:

$$R(M) = \sum_i r(O_i) - \sum\sum_{i<j} r(O_i \cup O_j) + \sum\sum\sum_{i<j<k} r(O_i \cup O_j \cup O_k) - \cdots + (-1)^{n+1} r(O_1 \cup O_2 \cup \ldots \cup O_n) \tag{2.1.2}$$

Due to the fact that the reliability of an overlay path is the product of the reliabilities of its overlay links, which are the products of the reliabilities of their path segments, we have:

$$r(O_i) = \prod_{j, L_{i,j} \in O_i} \prod_{k, S_{i,j,k} \in L_{i,j}} r(S_{i,j,k})$$

To establish a multipath connection, the source node sends a *path-establish* request to the destination using overlay routing. The message collects path information such as composition and reliability along the primary path. When the destination receives a *path-establish* message, it broadcasts a *path-explore* message to all its neighbours. A node that receives the message calculates the joint reliability of the primary path and the path from itself to the destination. If the quality-cost ratio is above a predefined threshold, it broadcasts the message to its neighbours, if not it sends the information to a single next hop towards the source which is determined by the routing table. The source receives a set of secondary paths which it can select a multipath from.

TAROM exploits information about underlay routes to find high-quality secondary paths. However, the approach is inapplicable for most unstructured overlay networks due to the fact that it needs an overlay routing layer. Also, TAROM does not take node failures into account. Additionally, similar to [MB07] the system uses partial flooding which leads to high numbers of messages.

2.1.6 Maximum-Reliability-Tree: Adaptive Gossiping Algorithm

An adaptive broadcast protocol based on gossiping has been proposed by Garbinato et al. [GPS04]. The approach tries to improve gossiping performance by taking into account topology and reliability information of the network (only overlay topology and link information is considered). First an optimal solution is presented, where each node has perfectly accurate knowledge about the system topology and node and link reliabilities. Then the assumption is replaced with a more realistic one, where each node approximates the topology and reliability parameters.

Optimality and adaptiveness are defined as follows:

Optimality *"A probabilistic reliable broadcast algorithm $\mathscr{O}^K$ is optimal to some configuration C w.r.t. the number of messages if there is no algorithm $\mathscr{X}^K$ such that processes executing $\mathscr{X}^K$ in C exchange fewer messages than processes executing $\mathscr{O}^K$ in C."* [GPS04]

Adaptiveness *"A probabilistic reliable broadcast algorithm $\mathscr{A}^K$ is adaptive to some configuration C if and only if the number of messages exchanged by processes executing $\mathscr{A}^K$ in C in response to a broadcast is eventually equal to the number of messages exchanged by processes executing $\mathscr{O}^K$ in C."* [GPS04]

The general idea of Garbinato et al. is to build a *Maximum-Reliability-Tree (MRT)* which is a spanning tree that contains the most reliable paths in the graph connecting all nodes. In the following the optimal algorithm will be described briefly:

When a process p_s broadcasts a message, it calculates the MRT $mrt_s(G,C)$ of graph G and configuration C locally using a modified version of Prim's algorithm[5]. Because all processes agree on the same topology, they all build the same MRT. Using $mrt_s(G,C)$ the optimal number of messages that must transit through each edge in order to reach all processes with probability K is computed. This algorithm is executed by each process.

The adaptive algorithm differs from the optimal solution in the knowledge which processes have about the topology G and the configuration C. Thus, the adaptive algorithm constantly tries to approximate G and C. To approximate G, processes exchange heartbeat messages with all their neighbours, containing their view of the topology. Eventually all processes agree on the same global system topology. Heartbeats are also used to obtain the reliability of links (configuration C) and to share this information with neighbours. The reliability of the processes is approximated by each process itself by periodically reading the value of its local clock and storing it to a stable storage. The probability of a failure is proportional to the number of intervals missing on the stable storage during some sufficiently large amount of time.

Building a MRT to reach a defined reliability is a promising approach. However, the system does not consider the underlay topology. Sending a message over the same link multiple times at once may not necessarily increase the probability that the message is delivered. A common reason for message losses is because of congestion in message queues of intermediate nodes (compare section 2.1.2). Depending on the drop-rule of the message queues, increasing the number of messages on an overloaded link usually does not improve the delivery probability. Another problem is that eventually each node has global knowledge about the overlay topology and failure-probability configuration which limits scalability.

[5] Non-distributed greedy algorithm to find a minimum spanning tree for a connected weighted undirected graph

2.2 Topology Discovery

In order to provide reliability underlay awareness is important. The system needs mechanisms to discover the underlying topology. This section presents related work to discover the underlay topology among sets of peers in a distributed system.

2.2.1 Topology-Aware-Grouping

Topology-Aware-Grouping (TAG) [KF02] is a heuristic application-level multicast approach. TAG leverages underlying network topology data to construct multicast overlay networks. It uses information about overlap in routes to the sender among group members to set up the overlay. The main focus of TAG is to build a tree with a low relative delay penalty and to introduce a limited number of identical copies of a packet on the same link. The tree is constructed from a single source which is called the sender. For underlay path discovery traceroute is used, but other path discovery methods like using a topology server are mentioned as well. For tree construction a relation for paths is defined. A path from A to node B, denoted by $P(A,B)$, is a sequence of routers comprising the shortest path according to the underlying routing protocol from node A to node B. For a given source S the relation is defined as follows:

$$A \succ B \Leftrightarrow P(S,A) \text{ is a prefix to } P(S,B)$$

In the example in Figure 2.3 $P(S,D5) = \langle R1,R2,R4\rangle$ whereas $P(S,D1) = \langle R1,R2\rangle$ which is a prefix to the former. Hence, $D1 \succ D5$.

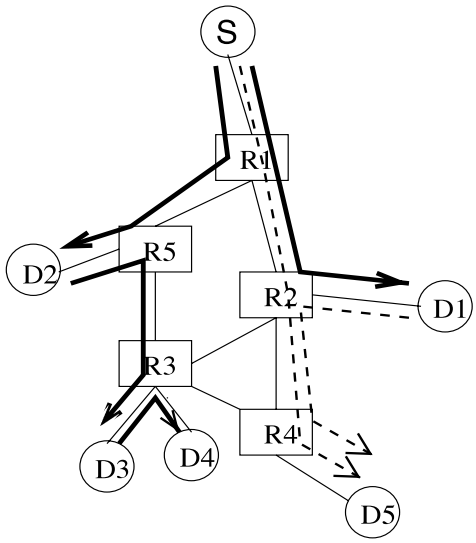


Figure 2.3: Example of a TAG tree (Source: [KF02])

A new joining member sends a JOIN-Message to the root of the tree, which is the sender S in the multicast tree. S then obtains the path to the joining node called *spath*. The request is then either

forwarded to the child of *S* with the longest *spath* which is a prefix to the new nodes *spath*, or if no such child exists, *S* adds the new node to the list of its own children. The complete path-matching algorithm can be found in algorithm 2.1.

Algorithm 2.1 Path-Matching algorithm of TAG ([KF02])

```
procedure PATHMATCH(currentNode,newNode)
    ch ← first child of currentNode
    flag ← condition(3)
    while ch ≠ NULL do
        if ch ≻ newNode then
            target ← ch
            flag ← condition(1)
        end if
        if newNode ≻ ch then
            add ch to children(newNode)
            newNode becomes parent(ch)
            flag ← condition(2)
        end if
        if flag ≠ condition(1) then
            ch ← next child of currentNode
        else
            ch ← NULL
        end if
    end while
    if flag = condition(1) then
        PATHMATCH(target, newNode)
    else
        add newNode to children(currentNode)
    end if
end procedure
```

TAG allows to connect peers according to the underlay. However, the main purpose for TAG is not to discover the whole underlay topology. TAG only discovers the underlay topology of the routing tree from a single source. To discover the complete topology still $O(N^2)$ traceroutes would be necessary but TAG provides a very easy mechanism to construct underlay aware overlay topologies.

2.2.2 Max-Delta

Max-Delta [JTC08] is a centralised approach to quickly and efficiently infer the router-level topology among a group of *N* hosts using end-to-end measurement tools such as traceroute. It considers the fact that a router-level network is a sparse graph so that $O(N^2)$ traceroutes among hosts are not necessary. It relies on a central server to collect traceroute results and to select paths for hosts to traceroute. To reduce traceroute overhead it uses an algorithm called *Doubletree*. Max-Delta works as follows:

First each host estimates its network coordinates using tools such as GNP [NZ02] or Vivaldi [DCKM04]. Then multiple iterations of the following procedure are executed: The server selects a target for each host to traceroute. Hosts traceroute their targets and send the results back to the server. Then the server starts the next iteration based on the combined results. The process is executed until a predefined stop rule is valid.

To select targets to traceroute the server does the following: For a certain host H_i the server computes the distance $D_p(H_i,H_j)$ in the so far discovered topology between H_i and another host H_j where the path between these two hosts has not been measured, using a shortest path algorithm like Dijkstra. It also computes the euclidean distance $euclidean(H_i,H_j)$ between hosts based on the network coordinates. In a d-dimensional space the distance between two points H_i and H_j with the coordinates $(X_{1,1},X_{1,2},\ldots,X_{1,d})$ and $(X_{2,1},X_{2,2},\ldots,X_{2,d})$ can be computed as follows:

$$euclidean(H_i,H_j) = \sqrt{(X_{1,1}-X_{2,1})^2+(X_{1,2}-X_{2,2})^2+\cdots+(X_{1,d}-X_{2,d})^2}$$

Ideally the euclidean distance approximates the real network distance between H_i and H_j. The gap between the euclidean distance and the distance defined by the discovered topology is defined as follows:

$$\Delta(H_i,H_j) = D_p(H_i,H_j) - euclidean(H_i,H_j)$$

If $\Delta(H_i,H_j)$ is large, the probability that some links between H_i and H_i are not yet discovered is high. These links would lead to a shorter distance in the discovered topology and hence reduce the gap $\Delta(H_i,H_j)$. The server selects the path with the maximum Δ as next traceroute target for H_i.

To reduce traceroute overhead Max-Delta makes use of the Doubletree algorithm proposed in [DFC05, DRFC04, DRFC06]. A normal traceroute sends a series of IP datagrams with increasing TTL to the destination. When an intermediate router receives a package with $TTL = 0$, it usually sends a Time-To-Live-Exceeded message or a similar message back to the source which contains the ID of the router. Large-scale traceroute measurements have high redundancies. Routers and links are often repeatedly discovered in several traceroutes. *Intramonitor redundancies* refer to traceroutes where the same links and routers are discovered repeatedly when a host H_1 conducts traceroutes to different hosts H_2 and H_3. Figure 2.4 shows an example where the Doubletree algorithm can reduce redundancies. Assume the path between H_1 and H_2 is $\langle H_1,R_1,R_2,R_3,H_2\rangle$ and the path between H_1 and H_3 is $\langle H_1,R_1,R_2,R_4,H_3\rangle$. Hence, the routers R_1 and R_2 and the links (H_1,R_1) and (R_1,R_2) are discovered twice in a traceroute measurement to these two hosts. *Intermonitor redundancies* exist in traceroutes from multiple monitors to the same destination. Let us assume that the paths from the last example are symmetric, a traceroute from H_2 to H_1 and from H_3 to H_1 would obtain the paths $\langle H_2,R_3,R_2,R_1,H_1\rangle$ and $\langle H_3,R_4,R_2,R_1,H_1\rangle$, respectively. Hence routers R_1 and R_2 and the links (R_2,R_1) and (R_1,H_3) are discovered by both hosts H_2 and H_3. To reduce these redundancies Doubletree modifies the traceroute process. Instead of starting with $TTL = 1$ it starts the process with $TTL = h$, whereby h is a predefined system parameter. Probing then proceeds in both directions, forward toward the destination and backward to the source. *Backward probing* uses a *local stop set*, which contains all routers already visited. If backward probing finds a router which is in this set, it stops probing. *Forward probing* uses the *global stop set* which contains all pairs $(router, destination)$ from all traceroutes made by the system. A pair is added to the global stop set if the router is visited in a traceroute toward the corresponding destination. Forward probing stops whenever a member of the global stop set is met.

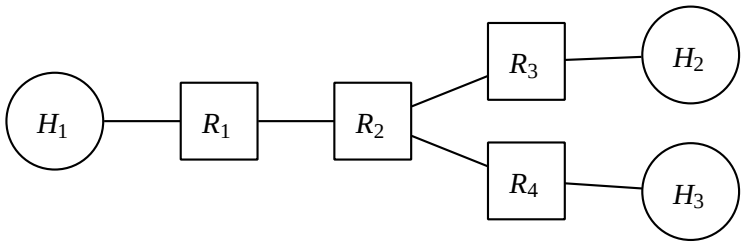


Figure 2.4: Example graph for intra-monitor and inter-monitor redundancies
R_i – Routers; H_j – Hosts

2.2.3 Distributed Max-Delta

There are several limitations in Max-Delta. First of all it does not scale for a large number of hosts. In each iteration the server takes $O(V_p \log V_p + E_p + N)$ time to select a traceroute target for a host, where N is the number of hosts and V_p and E_p are the number of nodes (hosts and routers) and links in the discovered underlay, respectively. Furthermore the server is a single point of failure.

This is the reason why Jin et al. [JTC08] additionally proposed a distributed inference scheme. Each host maintains a partially discovered topology and uses Max-Delta heuristics to select traceroute targets independently. Hosts are part of a low diameter overlay tree which is used to exchange traceroute results with others.

The actions of a new host are the following: First it estimates its network coordinates using one of the previously mentioned tools. Then the host identifies its parent node to join the tree using a tree maintenance mechanism which constructs low diameter trees by placing hosts in such a way that the larger the hosts outdegree-bound is, the closer to the root it is put. Next, the new host conducts a *first-round traceroute* which means the node performs a traceroute to its parent. The purpose of this phase is to form a connected graph. If each host traceroutes like this, at least a spanning tree is built. A connected graph is needed to enable peers to find the shortest path to any other peer in the discovered topology. Then the host sends its coordinates and first-round traceroute results to its tree neighbours. After this initialization phase the host periodically selects traceroute targets using Max-Delta heuristics, accepts data from its neighbours, which is then aggregated with its own results and sent to all its neighbours.

The distributed version of Max-Delta scales much better than the centralised one. However, each node eventually holds the completely discovered topology which limits scalability and often is not even needed. The system completely relies on traceroute which, however, has some drawbacks. For privacy and security reasons on today's Internet, there are routers which do not respond to traceroute messages. These routers are denoted as *anonymous routers*. As per [JTC08] nearly one third of probed paths contain anonymous, private or invalid routers. The next Section 2.2.4 will show a way to improve path discovery in presence of anonymous routers by inferring network topology using end-to-end measurements.

2.2.4 Topology Inference From End-to-End Measurements

Traceroute measurements have some drawbacks due to anonymous routers. Paths can be incomplete or inaccurate. A possible approach to overcome some of the problems with traceroute is network tomography. Network tomography refers to approaches that use end-to-end packet probing measurements such as packet loss and delay measurements. Compared to traceroute-like measurements the internal nodes (such as routers) do not require extra cooperation except for packet forwarding towards the destination. A source node sends probe packets to a set of destination nodes. Using correlations within the probing results makes it possible to partially infer the network structure. Ni et al. [NXTY10] proposed a model to discover the logical routing-topology. The logical routing-topology is a tree where each intermediate node (all except for root and leafs) has at least an out-degree of three. It uses a combination of end-to-end probing and traceroute measurements. The approach supports both, multicast probing where the underlay handles packet forwarding to a set of destination nodes and back-to-back unicast probing, where the source node sends a string of back-to-back unicast packets to the destination nodes, one packet for each destination node. The probing model will be explained briefly in the following:

Each node and each edge holds a set of *Link State Variables* X_k for node k and Z_e for edge e. By causality the Link State Variable of a node k is dependent on the Link State Variable of the parent $f(k)$ of node k and the state of the link $e_k = (f(k),k)$

$$X_k = g(X_{f(k)}, Z_{e_k})$$

It can be shown that the outcome variables X_K induced by the transmission of a probe form a *Markov Random Field* on the routing tree where tree topology and link parameters can be uniquely determined by joint distributions of outcome variables. To estimate joint distributions, a source node sends a sequence of n probes so that there will be n outcome variables $X_V^{(t)} = (X_k^{(t)} : k \in V), t = 1,2,\ldots,n$.

The approach is able to improve correctness of inferred routing topology. However, because of the probabilistic model used by the approach no guarantees on correctness can be given. Furthermore, it introduces additional overhead into the system when discovering underlay paths.

2.3 Minimum-Spanning-Tree (MST)

Building a Minimum Spanning Tree (MST) is a well-known field in research. Gallager et al. [GHS83] proposed the first distributed algorithm to build a MST with optimal message complexity of $O(n \log n + m)$, where n is the number of vertices and m is the number of edges. The algorithm is denoted as GHS algorithm in literature. However, it has a non-optimal time complexity of $O(n \log n)$. Even though faster algorithms have been proposed, in this thesis the algorithm is used because of its ease of extensibility to be used for building a forest of k MST (Chapter 6).

The basic idea of GHS is to build fragments which by themselves only contain minimum weight edges. These fragments are then merged using the minimum weight edge connecting two fragments. The fragments are built in levels. At the start of the algorithm all nodes are part of a fragment of size one at level zero, only containing themselves. In each step the fragment size is increased. A fragment has a leader that decides which edge to add to the MST in a step. A node holds three sets of edges,

namely *basic* edges, *branch* edges and *rejected* edges. The set of basic edges contains all edges which are not yet part of the MST and have not yet been processed. Initially all edges are part of this set. The set of branch edges contains the edges which are part of the MST. The set of rejected edges contains all edges which have been rejected by a neighbour node and will not be part of the MST. To find the *minimum weight outgoing edge* (mwoe) the leader broadcasts an `INITIATE`-Message along the spanning tree edges. When a node receives such a message, it starts probing its edges which are still in the set of basic edges in increasing order. To test an edge the node sends a `TEST`-Message along the edge. When a node receives such a message it can either accept by replying with an `ACCEPT`-Message or reject by sending a `REJECT`-Message. A `REJECT`-Message is sent, if the sender of the `TEST`-Message belongs to the same fragment. Otherwise, it replies with an `ACCEPT`-Message. A node keeps probing all its basic edges until it receives an `ACCEPT`-Message from one of its probed neighbours. The result of the probing is convergecast to the leader of a fragment along the branch edges, which means each node waits for all its children to report their results and each reports only the best edge out of the results, which has been accepted. While reporting each node stores the next hop to the node with the best edge. When the leader receives reports from all its children, it can decide which edge is the *mwoe* within the fragment. It sends a `CHANGE_CORE`-Message along the path to the best edge. When the node having the *mwoe* receives the message, it sends a `CONNECT`-Message across its *mwoe*. When `CONNECT`-Messages have been sent from both directions through the same *mwoe* and both fragments have the same level, the two fragments are merged. The new fragment increases its level by one and one of the nodes of the *mwoe* is selected to be the new leader. If a node receives a `CONNECT`-Message from a fragment with a lower level than its own fragment, the other fragment is *absorbed*. When a fragment gets absorbed, the lower level fragment gets incorporated into the higher level fragment.

Figure 2.5 shows an example of the GHS algorithm constructing a MST. At the beginning each node is part of a fragment of size one and the fragment level is zero (Figure 2.5(a)). Nodes start probing their *mwoe*. Node p_1 probes edge (p_1, p_2), node p_2 and p_4 probe edge (p_2, p_4) and nodes p_3 and p_5 probe edge (p_3, p_5). Because all nodes are part of different fragments, each node replies with an `ACCEPT`-Message. Nodes p_2, p_3, p_4 and p_5 then send a `CONNECT`-Message across their *mwoe* and merge to two fragments by adding edges (p_2, p_4) and (p_3, p_5) respectively to the set of branch edges (Figure 2.5(b)). Node p_1 then also sends a `CONNECT`-Message to node p_2 because it received an `ACCEPT`-Message from this node. Because the fragment level of node p_2 is higher than the one of the fragment of node p_1, node p_1's fragment is absorbed and edge (p_1, p_2) is added to the fragment of node p_2 and p_4 (Figure 2.5(c)). Now the graph consists of two fragments. Assume node p_2 and node p_5 are the leaders of the two fragments, both broadcast `INITIATE`-Messages along their branch edges to node p_1 and p_4 and to node p_3 respectively. Each node starts probing its minimum weight basic edge which is not yet part of the set of branch edges or rejected edges (for example node p_1 probes edge (p_1, p_3)). Node p_4 and p_3 will both report edge (p_3, p_4) to their leaders p_2 and p_5 respectively. The leaders both send a `CHANGE_CORE`-Message to these nodes which then send a `CONNECT`-Message across the edge (p_3, p_4). Because both fragments have the same level, they add edge (p_3, p_4) to their branch edges and select the new leader which is one of the endpoints of edge (p_3, p_4). The new leader will again broadcast an `INITIATE`-Message along branch edges and all nodes start probing their basic edges. Because all nodes are now part of the same fragment, nodes reply with a `REJECT`-Message whenever they receive a `TEST`-Message. After probing, each node sends a report towards the leader containing the information that there is no *mwoe* and the algorithm terminates.

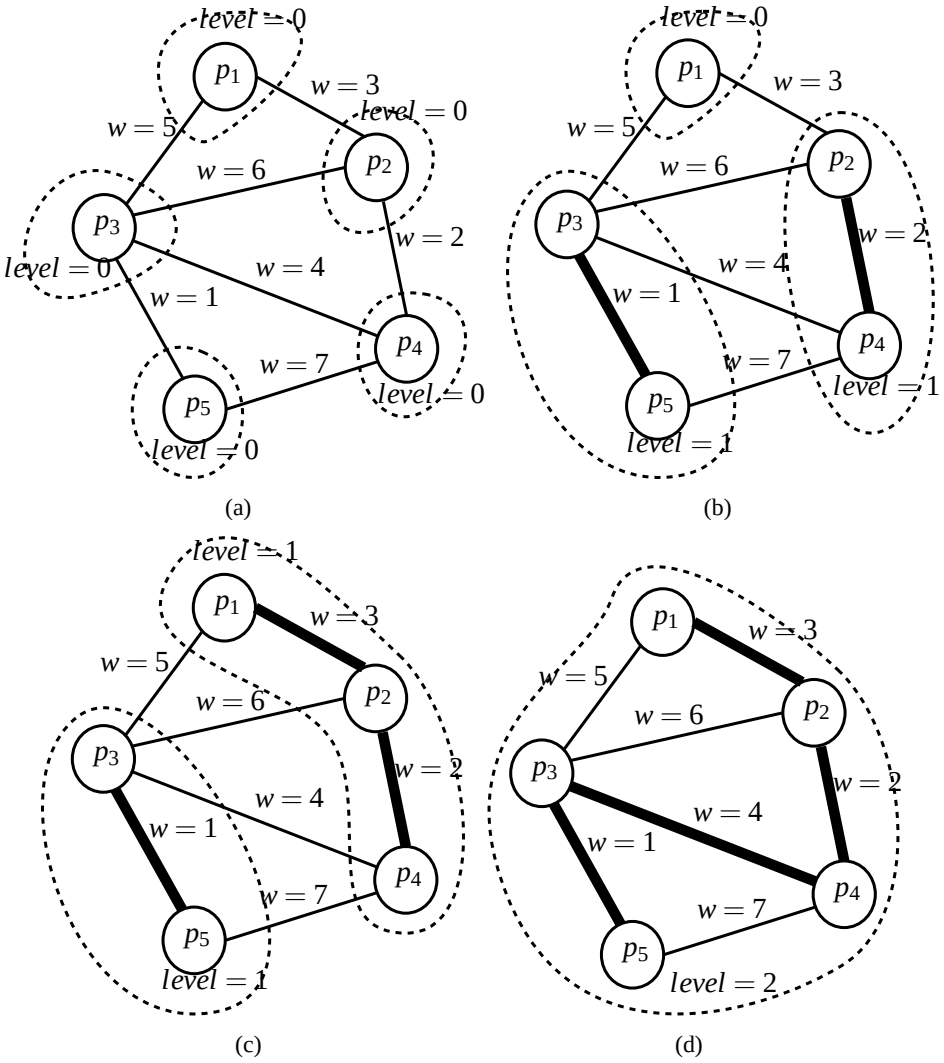


Figure 2.5: Example of a graph constructing a MST using the GHS algorithm

3 System Model and Problem Formulation

In this chapter the system model will be described and problems to be solved by the proposed system will be revealed. We consider a physical network consisting of an unbound set of routers $V^U = \{r_i | i \geq 1\}$. Routers are connected through a set of undirected physical links $E^U = \{l^U_{i,j} | r_i, r_j \in V^U\}$. The graph $G^U = (V^U, E^U)$ is denoted as *underlay*.

Figure 3.1 shows the interrelations in the system model. An underlay path $Path^U[r_i, r_j]$ between two routers r_i and r_j is a sequence of physical underlay links $\langle l^U_{i,1}, l^U_{1,2}, \ldots, l^U_{m-2,m-1}, l^U_{m-1,j} \rangle$. The graph G^U is connected, which means there is at least one path between any pair of routers. The path between any two routers is defined by a routing algorithm in the underlay.

The system furthermore consists of an unbound set of peers $V^O = \{p_i | i \geq 1\}$. A peer is an application instance connected to at least one router r_i. Peers can establish overlay links l^O_i between each other. An overlay link corresponds to an underlay path which is a sequence of underlay links. A set of peers forms an overlay by constructing a connected graph $G^O = (V^O, E^O)$ where the set of overlay links is denoted by $E^O = \{l^O_{i,j} | p_i, p_j \in V^O\}$. Peers can either communicate by establishing an overlay link or by using an overlay path. An overlay path $Path^O[p_i, p_j]$ between two peers p_i and p_j is a sequence of overlay links $\langle l^O_{i,1}, l^O_{1,2}, \ldots, l^O_{m-2,m-1}, l^O_{m-1,j} \rangle$ where intermediate peers act as forwarders of messages.

A peer p_i can discover the corresponding underlay path of an overlay link $l^O_{i,j}$, which is the path defined by the routing algorithm in the underlay, using traceroute techniques. An underlay path between two peers p_i and p_j, connected to the two routers r_k and r_h respectively is $Path^U[p_i, p_j] = Path^U[r_k, r_h]$.

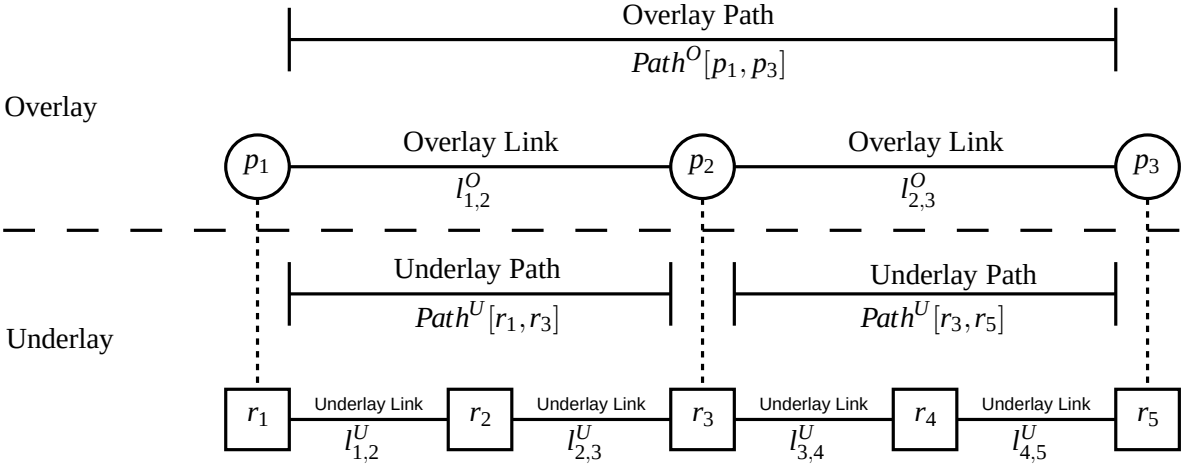


Figure 3.1: System model

An underlay path is defined as a sequence of underlay links but a traceroute measurement returns a sequence of underlay routers $\langle r_1, r_2, \ldots, r_m \rangle$. However, the two concepts can be used interchangeably, i.e. $Path^U[r_i, r_j] = \langle l^U_{i,1}, l^U_{i,2}, \ldots, l^U_{m,j} \rangle = \langle r_i, r_1, \ldots, r_m, r_j \rangle$. The length $len(Path^U[r_i, r_j])$ of an underlay path $Path^U[r_i, r_j]$ is the number of routers within the path.

Similarly, an overlay path can be expressed as a sequence of peers $Path^O[p_i, p_j] = \langle l^O_{i,1}, l^O_{i,2}, \ldots, l^O_{m,j} \rangle = \langle p_i, p_1, \ldots, p_m, p_j \rangle$ and the length $len(Path^O[r_i, r_j])$ of an overlay path $Path^O[r_i, r_j]$ is the number of peers within the path.

Underlay routers, underlay links and peers can fail independently, which causes messages being dropped when sent through them. A link failure of an underlay link $l^U_{i,j}$ causes all messages sent over an overlay link $l^O_{k,h}$ that uses an underlay path containing $l^U_{i,j}$ to be lost during the failure. Hence, a message which is sent over an overlay path also gets lost whenever it contains $l^O_{k,h}$. The same is true for underlay routers. Similarly, a failure of a peer p_i causes all messages sent through an overlay path containing an overlay link $l^O_{i,j}$ to be lost.

Each router and link has a failure probability defined by $\varphi(r_i)$ and $\varphi(l^U_{i,j})$. Hence, the reliability of a link $l^U_{i,j}$ is defined by $(1 - \varphi(l^U_{i,j}))$. Similarly, the reliability of a router r_i is defined by $(1 - \varphi(r_i))$. Because an overlay link directly maps to an underlay path, the reliability of an overlay link is the product of the individual reliabilities of all underlay routers and links within the underlay path. If we consider an overlay link $l^O_{k,h}$ which is mapped to an underlay path consisting of n routers, the failure probability of this overlay link is given by the following equation:

$$\varphi(l^O_{k,h}) = 1 - \prod_{1}^{n}(1 - \varphi(r_i)) \cdot \prod_{1}^{n-1}(1 - \varphi(l^U_i))$$

Peers can leave and join the system at arbitrary times and they can fail temporarily or permanently. Moreover, a peer p_i fails with probability $\varphi(p_i)$. The reliability of an overlay path is given by the product of the reliabilities of all overlay links and peers within the path. If we consider an overlay path $Path^O[p_k, p_h]$ with $len(Path^O[p_k, p_h]) = n$, the failure probability of this overlay path is given by the following equation:

$$\varphi(Path^O[p_k, p_h]) = 1 - \prod_{1}^{n}(1 - \varphi(p_i)) \cdot \prod_{1}^{n-1}(1 - \varphi(l^O_i))$$

Moreover, we consider a broker-less content-based publish-subscribe system. The publish-subscribe software runs on the overlay peer. Peers can be both, publishers and subscribers at the same time. A publisher can publish any message without advertising it.

An event e_i published by a publisher consists of a set of attribute-value pairs $(attribute, value)$ where the attributes are simple names and the value is taken from a limited set of types like boolean, number and string. An event, for example, could look like the following:

$$\{(x, 10), (y, \text{“some text”}), (z, true)\}$$

A subscription S_i is a filter defined by the subscriber. A filter is a set of predicates which are triples $(attribute, operator, value)$. The attribute is a simple name, similar to the attributes in events. The set of possible operators is $\{“=”, “\neq”, “>”, “<”, “\leq”, “\geq”, “\in”\}$. In order for an event to match a subscription, every predicate within the subscription must be satisfied by an attribute-value pair in the event. For example the subscription $S_i = \{(x, “>”, 2), (z, “=”, true)\}$ matches the above event, but the filter $S_j = \{(x, “>”, 15), (z, “=”, true)\}$ does not match because the first predicate is not satisfied. An event e_i matching a subscription S_i is denoted as $e_i \in S_i$.

An event e_i published by a publisher is intended to be received by all subscribers having at least one subscription S_i where $e_i \in S_i$. The set of peers with a matching subscription for an event e_i is denoted as $S(e_i) = \{p_j \in V^O | p_j$ subscribed for a subscription that matches $e_i\}$.

The intention of this work is to provide reliable and timely dissemination of events, which means the probability that an event e_i published by a peer p_i is delivered to all subscribers in $S(e_i)$ has to be maximised.

Given a decentralised P2P content-based publish-subscribe system the following problems need to be solved:

1. *Reliability*: Reliable dissemination of events e_i to all subscribers in $S(e_i)$ considering node failures in G^O (peer failures) and link and node failures in G^U.
2. *Timeliness*: Events should be delivered timely, which means no procedures should be used for message forwarding which are very time-consuming or lead to an unpredictable delay, like request/reply protocols.

4 Approach Overview

In Chapter 3 two main problems will be considered. The first problem is providing reliability. Failures affecting reliability can occur in two places. The first source of failures is within the underlay topology, i.e. underlay link and underlay node failures. The second source of failures is peer failures within the overlay. The second problem is timeliness.

In order to tackle the problem of underlay failures, we use an approach based on underlay diversity. Underlay diversity means that the number of common components within underlay paths used by overlay links is low. Therefore the underlay topology G^U has to be discovered. Using knowledge about underlay paths we construct an overlay topology which reflects the underlay topology. Because the overlay topology reflects the underlay topology, overlay links are diverse in the underlay.

In order to avoid losses due to peer failures, an approach to provide path-redundancy is focussed. The system establishes redundant paths among peers of the system to increase the message delivery reliability in a timely manner. Events are sent through multiple paths so that in case of a link failure on one of the paths another path may still be available. Redundant paths, however, only increase the reliability, if the intersection of the set of components within paths is small or ideally empty. If both paths share a common underlay link, both paths also share properties of this link. If the shared link fails, both overlay paths using this link also fail. Because of the low underlay diversity of overlay links, diverse overlay paths are most likely to be diverse in the underlay as well.

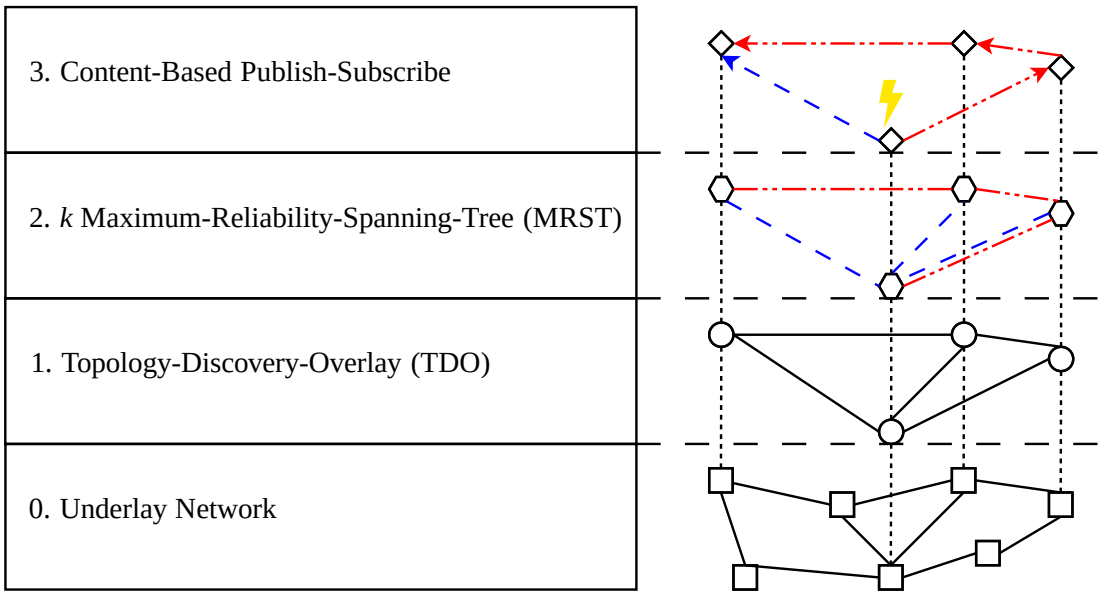


Figure 4.1: Protocol stack

In Figure 4.1 the protocol stack of the system is shown. In the following each protocol layer will be described briefly.

At the bottom we have the *Underlay Network*. This layer is assumed to be available and provided by the underlying network (for example IP-Network). The underlay provides P2P routing through an underlay routing protocol like distance-vector routing.

The first layer of the system, layer 1, is the *Topology-Discovery-Overlay* (TDO) layer. The main purpose of this layer is to discover the underlying network topology to form an overlay topology reflecting the underlay topology. To discover underlay paths, traceroute is used, which is an off-the-shelf mechanism to discover IP routing paths. After discovering underlay paths, the protocol on this layer establishes overlay links using a path-matching algorithm based on an approach of Kwon et al. [KF02]. When a peer joins the network, an overlay peer already part of the network compares the underlay path to the new peer to underlay paths of overlay links already established. In the case paths partly overlap each other, join-requests are forwarded towards a peer where the overlap is minimal. One of the characteristics of a so formed overlay network is that it has high path diversity because overlaps are reduced. Traceroute is usually slow because several IP packages have to be sent for each traceroute measurement. This is the reason why traceroute measurements should be reduced. When deciding which paths to traceroute, there is a trade-off between overhead and discovery accuracy. Therefore, we specifically propose two different protocols which introduce a limited number of traceroute measurements. The first protocol is a *landmark-based approach*. A small dedicated subset of the available peers is selected as landmark. Each peer then contacts each of the landmarks which then discover the direct underlay path to the new peer. Using overlap information among this underlay path and other already discovered paths, landmarks forward the join-requests of new peers towards a neighbour with minimal path overlap according to the underlay topology. The second protocol is a *random approach*, where peers discover the underlay paths to a small random set of other peers already part of the system. Based on overlap information among paths in their own neighbourhood, the contacted peers then forward join-requests of new peers towards a better peer.

The next layer 2 builds k overlay link disjoint *Maximum-Reliability-Spanning-Trees* (MRST) on top of the TDO. This guarantees that the k most reliable overlay links of each peer are part of the publish-subscribe dissemination tree. The MRST forest has the property that between any pair of peers there are k link disjoint overlay paths. The MRST is constructed using the well-known Minimum Spanning Tree (MST) algorithm GHS [GHS83]. The main contribution of this thesis to this layer is an approach to build k connected link disjoint MRSTs, which means that on each MRST there is a path between any two peers.

The publish-subscribe layer 3 uses MRSTs to disseminate events through the k overlay link disjoint paths. Subscriptions are flooded through all MRSTs to create event routing tables. Events are then forwarded by overlay peers according to routing tables on each of the k paths.

5 Topology-Discovery-Overlay (TDO)

In this chapter the Topology-Discovery-Overlay (TDO) layer will be described. The goal of this layer is to discover the overlay and to construct an overlay topology reflecting the underlay topology. By this path diversity of overlay links is decreased. The TDO is based on an algorithm by which overlap information within underlay paths is used to find a location for peers with low correlations in the underlay.

To discover the whole underlay topology in the general case in a peer network of n peers, $O(n^2)$ traceroutes are necessary. However, because of high redundancies in underlay paths, $O(n^2)$ traceroutes are not always necessary in real Internet topologies. To reduce measurement complexity two approaches will be proposed: The first is a landmark-based approach by which only a small subset of peers has to traceroute other peers. The second is a more random approach by which each peer traceroutes a random set of peers. Later an optimization for the random approach will be given, which leverages the hierarchical structure of the Internet.

Before considering the two protocols more thoroughly, the prefix-relationship "$\succ$" between two underlay paths is defined: $Path^U[r_i,r_j]$ is a prefix to $Path^U[r_i,r_k]$, expressed by $Path^U[r_i,r_j] \succ Path^U[r_i,r_k]$, if the sequence of routers in $Path^U[r_i,r_j]$ with $m = len(Path^U[r_i,r_j])$ equals the first m routers in $Path^U[r_i,r_k]$. For example, considering the three underlay paths $Path^U[r_1,r_3] = \langle r_1,r_2,r_3\rangle$, $Path^U[r_1,r_4] = \langle r_1,r_2,r_4\rangle$ and $Path^U[r_1,r_5] = \langle r_1,r_2,r_3,r_5\rangle$ leads to the conclusion that path $Path^U[r_1,r_3]$ is a prefix to $Path^U[r_1,r_5]$ ($Path^U[r_1,r_3] \succ Path^U[r_1,r_5]$) but $Path^U[r_1,r_4]$ is not a prefix to $Path^U[r_1,r_5]$ ($Path^U[r_1,r_4] \not\succ Path^U[r_1,r_5]$).

5.1 Landmark-Based Approach

The landmark-based approach uses a tree-based approach where each landmark is the root of a tree. A tree is constructed from a single peer (the landmark) using traceroute measurement results. Whenever a peer joins a tree, it contacts the root of the tree. The root then traceroutes the path to the peer and either adds it as a child or forwards it to one of its children based on overlapping information within the paths. If the root has a child whose path is a prefix to the path to the new peer, it forwards the request to the child. Otherwise it adds the new peer as a child.

Let us, for example, consider a landmark peer p_1 which already has two children, p_2 and p_3, with $Path^U[p_1,p_2] = \langle r_1,r_2,r_3\rangle$ and $Path^U[p_1,p_3] = \langle r_1,r_4,r_5\rangle$ (arrows in Figure 5.1(a)). When peer p_4 joins the tree it contacts p_1, which discovers $Path^U[p_1,p_4] = \langle r_1,r_4,r_5,r_6,r_7\rangle$ (dashed line in Figure 5.1(a)). $Path^U[p_1,p_2]$ is not a prefix to $Path^U[p_1,p_4]$ but $Path^U[p_1,p_3]$ is a prefix, which is the reason why p_1 forwards the join-request to p_3, which adds p_4 as a child (Figure 5.1(b)).

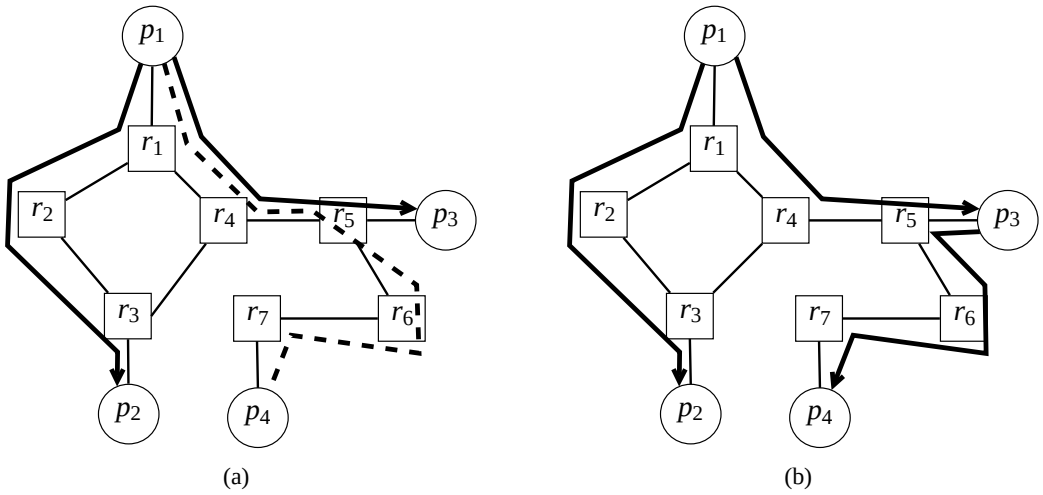


Figure 5.1: Tree construction in the landmark-based approach

The basic idea is to construct a forest of multiple trees. To get the optimal result in a network of n peers, n trees would be necessary. The problem with that approach is that it introduces a high overhead in traceroute measurements because doing so requires $O(n^2)$ traceroutes, which is the worst case for topology discovery.

The landmark-based approach reduces the number of traceroute measurements by only constructing a small number of trees. A subset of peers individually decides to be a landmark peer, which is the root of a tree. How a peer decides to be a landmark is not within the scope of this thesis but previous research has been done in this field which can be used to solve the problem. Similar to Singla et al. [SGF10] a possible solution would be that peers estimate the number of total peers n within the system, for example, using synopsis diffusion[1]. Then each peer individually picks a random number p uniform in $[0,1]$ and decides to become a landmark if $p < \frac{k}{n}$, where k is the number of desired landmarks. The expected number of landmarks is $n \cdot \frac{k}{n} = k$, so there will be k landmarks with high probability.

Each landmark peer is the root of a tree. As a result a forest of k trees is constructed in which each peer is a member of each of the k trees.

In Figure 5.2 a graph containing seven peers in total is shown, of which three are landmarks p_2, p_4 and p_7, with dashed green, dotted blue and solid red lines respectively. Each landmark is the root of a tree (Figure 5.3). Compared to an $O(n^2)$ approach which requires $7 \cdot (7-1) = 42$ traceroutes, only $3 \cdot (7-1) = 18$ traceroute measurements are necessary to build this overlay in which all underlay links and routers are discovered.

[1] Approach to compute aggregates (for example average sensor measurements) in a distributed manner

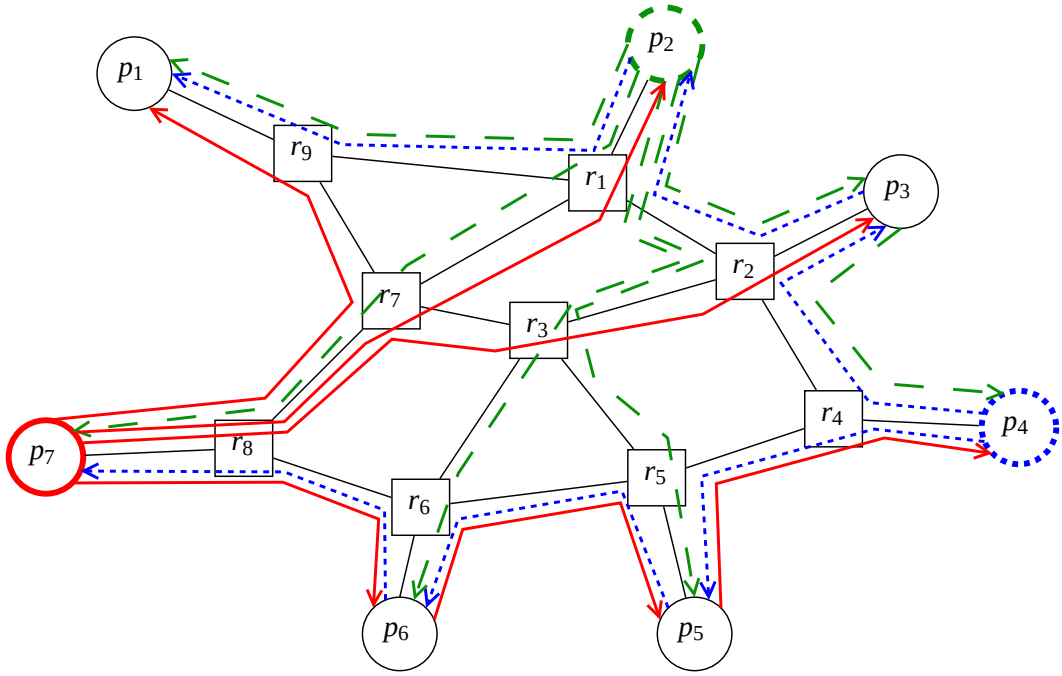


Figure 5.2: Landmark-Based overlay

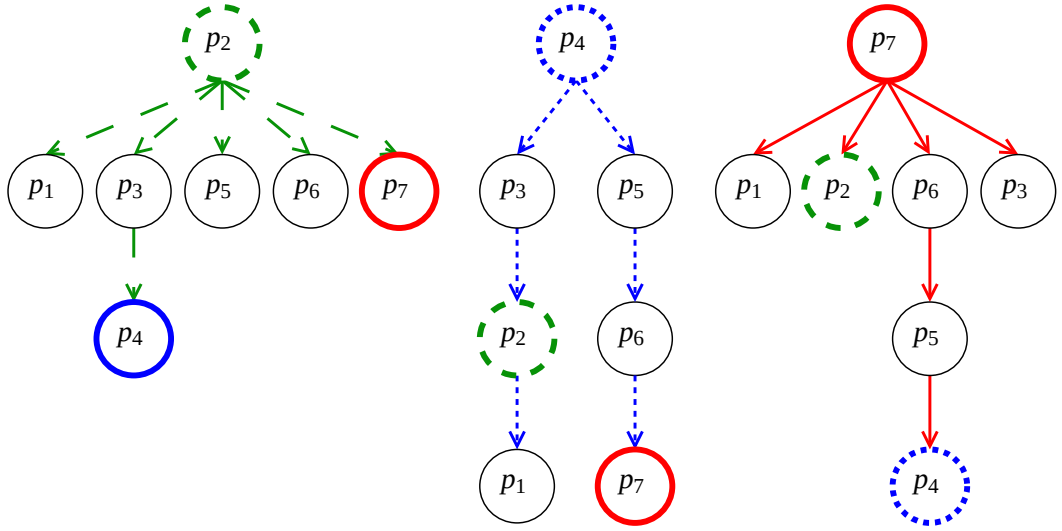


Figure 5.3: Three TAG trees of the example in Figure 5.2

5.1.1 Preliminaries

We assume that peers know a random set of other peers, for example, each peer knows *m* random overlay peers. This, for example, can be accomplished by using a random walk strategy. Peers exchange their set of landmarks through an epidemic algorithm with their random-known peers, for example,

each peer periodically picks a single random peer out of the random sample and sends the set of known landmarks to it. Eventually, each peer knows the complete set of landmarks.

A peer holds a set of family tables FT, one family table for each tree. Let us consider a family table where p_i is the root. The family table of a peer p_j within the tree, which is a child of p_k, is a three-tuple $(Path^U[p_i, p_j], p_k, children)$ consisting of the underlay path from the root peer of a tree to the current peer, the ID of the direct parent within the tree and a set of children. For each child within the set of children the ID of the child and the underlay path from the root of the tree to the child is stored in a tuple $(p_k, Path^U[p_i, p_k])$. These paths are needed to forward a join-request to a peer in a tree with the longest matching prefix.

A family table is identified by the peer ID of the root (which is always a landmark). If we consider p_i to be the root of a tree, the corresponding family table can be accessed by $FT(p_i)$.

The ID of the peer which is currently executing an algorithm can be accessed by $p_{current}$ by the executing peer.

5.1.2 Peer Network Joining

When a peer discovers a new previously unknown landmark from one of its neighbours, it sends a `JOIN`-Message to the landmark by executing Algorithm 5.1.

Algorithm 5.1 Algorithm executed when a peer discovers a new landmark

```
1: procedure JOINLANDMARK(p_i)
2:     SEND(JOIN, p_current, p_i)
3: end procedure
```

Algorithm 5.2 Algorithm called when a landmark receives a `JOIN`-Message

```
1: receive JOIN(p_i) at peer p_current begin
2:     Path^U[p_current, p_i] ← TRACEROUTE(p_i)
3:     PATHMATCH(p_current, p_i, Path^U[p_current, p_i])
4: end
```

When a landmark receives a `JOIN`-Message it executes Algorithm 5.2. First it discovers the path to the new peer p_i, then it calls the PATHMATCH procedure (Algorithm 5.1.2).

The PATHMATCH procedure iterates through all children of the family table for the specified root peer p_{root}. For each of its children the prefix relation is checked. There are three possible cases:

1. The path of the child p_{child} is a prefix to the path of p_i (i.e. $Path^U[p_{root}, p_{child}] \succ Path^U[p_{root}, p_i]$).
2. The path of p_i is a prefix to the path of the child p_{child} (i.e. $Path^U[p_{root}, p_i] \succ Path^U[p_{root}, p_{child}]$).
3. The path of p_i is neither a prefix to the path of the child, nor vice versa.

Algorithm 5.3 Path-Matching algorithm

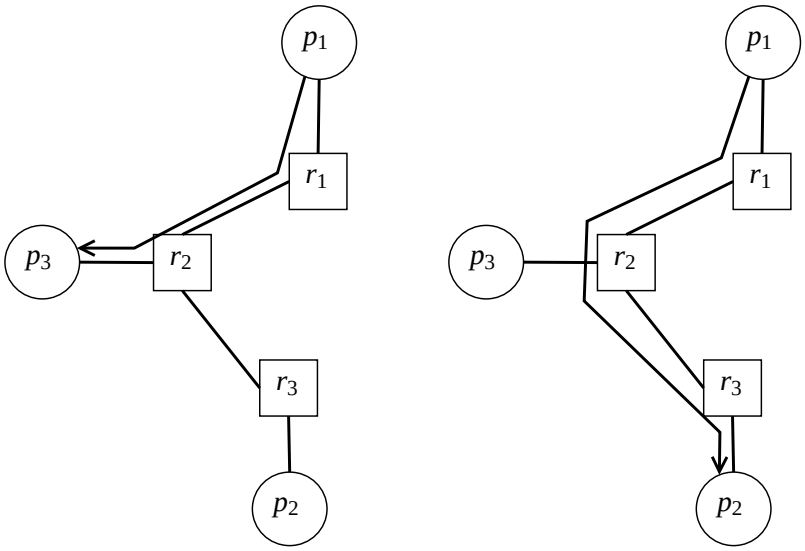
```
1:  procedure PATHMATCH(p_root, p_i, Path^U[p_root, p_i])
2:      p_target ← NULL
3:      parentFound ← FALSE
4:      p_child ← first child in FT(p_root)
5:      children = ∅
6:      while p_child ≠ NULL do
7:          if Path^U[p_root, p_child] ≻ Path^U[p_root, p_i] then
8:              p_target ← p_child
9:              parentFound ← TRUE
10:         end if
11:         if Path^U[p_root, p_i] ≻ Path^U[p_root, p_child] then
12:             children ← children ∪ {(p_child, Path^U[p_root, p_child])}
13:             remove p_child from set of children in FT(p_root)
14:         end if
15:         if ¬parentFound then
16:             p_child ← next child in FT(p_root)
17:         else
18:             p_child ← NULL
19:         end if
20:     end while
21:     if parentFound then                          // Forward JOIN-Request to a child peer
22:         SEND(FIND, p_target, p_i, p_root, Path^U[p_root, p_i])
23:     else                                         // Current peer will be the new parent
24:         add (p_i, Path^U[p_root, p_i]) to set of children in FT(p_root)
25:         SEND(ACK, p_i, p_root, Path^U[p_root, p_child], p_current, children)
26:     end if
27: end procedure
```

In case 1, we find a target to forward the join-request and send a FIND-Message to p_{child} (line 22). In case 2 we have the situation in which p_{child} has to be the child of p_i and will be removed as a child from $p_{current}$. Because there may be more peers satisfying this condition, we add p_{child} to a list which will be used later.

In Figure 5.4 both cases 1 and 2 are shown. In Figure 5.4(a) the path to p_1's child peer p_3, which is $Path^U[p_1, p3] = \langle r_1, r_2 \rangle$, is a prefix to $Path^U[p_1, p2] = \langle r_1, r_2, r_3 \rangle$. This is the reason why p_1 sends a FIND-Message to p_3 when p_2 is joining p_1. In Figure 5.4(b) peer p_2 is already a child of p_1. When peer p_3 sends a join-request to p_1 the peer p_1 has to remove p_2 from its children and add it as a child to p_3. The resulting tree is the same for both cases.

One may note that if there is any child which is satisfying case 1, there cannot be another child which is satisfying case 1 or case 2, so the two cases 1 and 2 are mutually exclusive.

To clarify, why no two children can satisfy case 1 at the same time, let us assume children p_j and p_k both are satisfying case 1. This means $Path^U[p_{root}, p_j] \succ Path^U[p_{root}, p_i]$ and $Path^U[p_{root}, p_k] \succ$



(a) p_3 satisfies case 1 when joining p_1

(b) p_2 satisfies case 2 when joining p_1

Figure 5.4: Possible cases when executing the PATHMATCH procedure

$Path^U[p_{root}, p_i]$. Hence, either $Path^U[p_{root}, p_j]$ is a prefix to $Path^U[p_{root}, p_k]$ or vice versa (or both, in case the paths are equal), because the router sequence of both paths matches the first couple of routers in $Path^U[p_{root}, p_i]$. Assuming $Path^U[p_{root}, p_j]$ is a prefix to $Path^U[p_{root}, p_k]$, peer p_k would have been added to p_j as a child and hence they cannot have the same parent.

To clarify why no child can fulfil case 2 if there is another child which is fulfilling case 1, assume child p_j fulfils case 1, i.e. $Path^U[p_{root}, p_j] \succ Path^U[p_{root}, p_i]$. Furthermore, let us assume there is another child p_k where $Path^U[p_{root}, p_i] \succ Path^U[p_{root}, p_k]$ is true (case 2). Then, because of transitivity of the prefix operator "$\succ$", we have $Path^U[p_{root}, p_j] \succ Path^U[p_{root}, p_k]$. In this case the PATHMATCH algorithm would have added p_k as a child of p_j and hence they cannot have the same parent.

However, there may be multiple peers fulfilling case 2. In line 12 the path match algorithm collects all these children and removes them from the set of children in line 13. If case 1 is not true, the new peer is added as a child of the current peer $p_{current}$ by sending an `ACK`-Message to p_i. If the path to the new peer is a prefix to the path to any child, the child is removed from the list of children and added as a child of the new peer. This is done by sending the set of removed children together with the `ACK`-Message (line 25).

A peer that receives a `FIND`-Message executes the PATHMATCH algorithm as well (Algorithm 5.4). Because the PATHMATCH algorithm runs on each peer, it recursively forwards a joining peer along the tree to its place of destination.

When a peer receives an `ACK`-Message, it first checks if it already is part of the tree (Algorithm 5.5). If not, it creates a new family table and sends an `ACK`-Message to all children in the set of received children. This is done to let the children change their parent from the previous parent – which is now the parent of the current peer – to the current peer. In this case when such a child receives an

Algorithm 5.4 Algorithm called when a peer receives a FIND-Message

1: **receive** FIND$(p_i, p_{root}, Path^U[p_{root}, p_i])$ **at peer** $p_{current}$ **begin**
2: PATHMATCH$(p_{root}, p_i, Path^U[p_{current}, p_i])$
3: **end**

ACK-Message, a family table for the tree should already exist and the only thing a peer has to do is to change its parent (line 8 of Algorithm 5.5).

Algorithm 5.5 Algorithm executed when a peer receives an ACK-Message in the landmark-based approach

1: **receive** ACK$(p_{root}, Path^U[p_{root}, p_{current}], p_{parent}, children)$ **at peer** $p_{current}$ **begin**
2: **if** $\nexists FT(p_{root})$ **then**
3: $FT(p_{root}) \leftarrow (Path^U[p_{root}, p_{current}], p_{parent}, children)$
4: **for** $(p_{child}, Path^U[p_{root}, p_{child}]) \in children$ **do**
5: SEND$(\text{ACK}, p_{child}, p_{root}, Path^U[p_{root}, p_{child}], p_{current}, \emptyset)$
6: **end for**
7: **else**
8: set parent of $FT(p_{root})$ to p_{parent}
9: **end if**
10: **end**

5.2 Random Approach

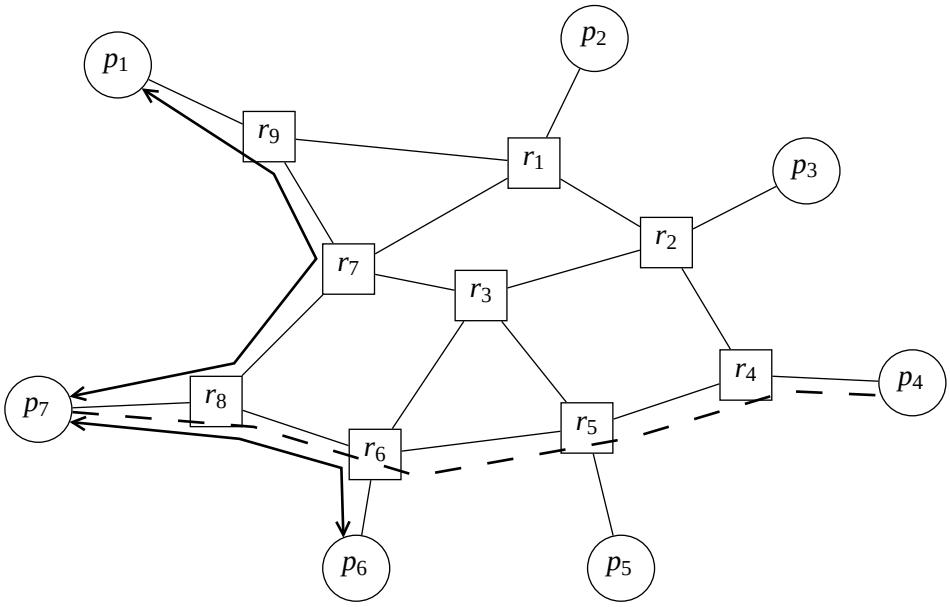
The landmark-based approach constructs multiple trees reflecting the underlay route from the root of the trees. However, landmarks have to be selected carefully to increase the gain in newly discovered links within the underlay. Additionally a landmark has a heavy load because each landmark has to do traceroute measurements to all other peers.

The random approach tries to overcome some of the limitations. Instead of building trees, each node is connected to a set of neighbours, which is constructed using a path-matching algorithm similar to the landmark-based approach. To establish connections peers do traceroute measurements to a random sample of peers in the overlay network and forwards join requests to connected neighbours with overlapping paths.

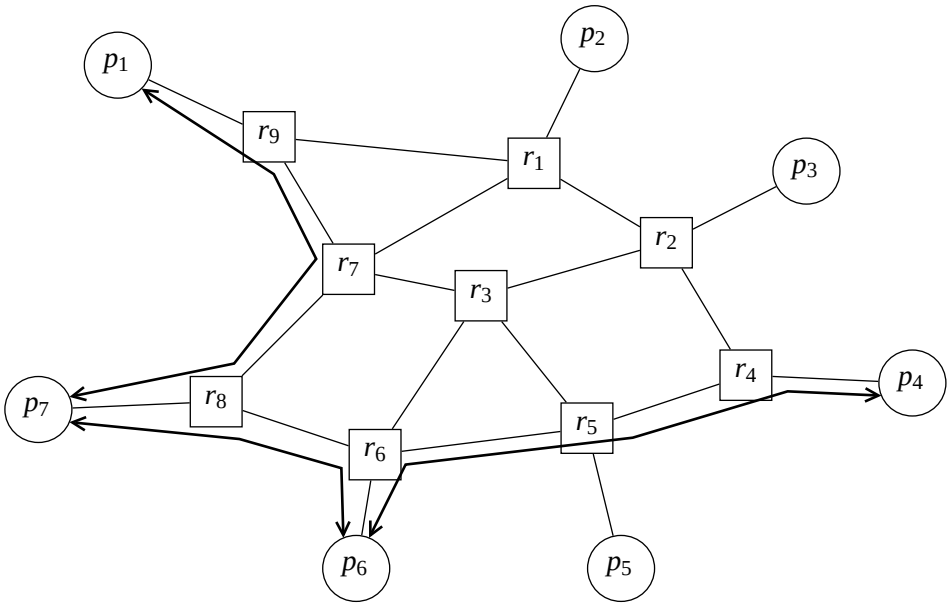
Compared to the landmark-based approach this approach distributes the traceroute measurement work of landmarks across the peer network. If peers have an average of k neighbours, each peer only has to conduct k traceroute measurements.

In the example in Figure 5.5 a graph is show where peer p_7 has two connected neighbours p_6 and p_1. Because p_4 sent a join-request to p_7, the peer p_7 discovers the path to peer p_4 (dashed line in Figure 5.5(a)) and compares it to the paths to all its neighbours. Because $Path^U[p_7, p_6] \succ Path^U[p_7, p_4]$ is true, it forwards the JOIN-Request to p_6. The peer p_6 does not have to do new traceroute measurements to p_4, because path $Path^U[p_6, p_4]$ can be inferred from the two paths $Path^U[p_7, p_6]$ and $Path^U[p_7, p_4]$ (for

a detailed description how the path is inferred, see Section 5.2.2). Using the inferred path to p_4, peer p_6 connects to p_4 (Figure 5.5(b)). If p_7 was already connected to p_4 but not to p_6 and p_6 sent a join-request to p_7, peer p_7 would discover the path to p_6, disconnect from p_4 and establish a connection to p_6. Then it requests p_6 to connect to p_4. The resulting overlay would be the same as in the previous example (Figure 5.5(b)).



(a) p_7 discovered underlay path to p_4



(b) `JOIN`-Request is delegated to p_6

Figure 5.5: Random approach overlay

5.2.1 Preliminaries

Similar to the landmark-based approach, the random approach assumes that each peer initially knows a random set of k other peers. Furthermore a peer holds a set of connected neighbours *neighbourList*. The list contains tuples $(p_{neighbour}, Path^U[p_{current}, p_{neighbour}])$ consisting of the neighbour's peer ID and the underlay path from the current peer to the neighbour peer.

5.2.2 Peer Network Joining

Initially, a peer sends a `JOIN`-Message to all peers in the random set of known peers. Upon receiving a `JOIN`-Message, Algorithm 5.6 is executed. The receiving peer first conducts a traceroute measurement to the joining peer p_i to get the underlay path $Path^U[p_{current}, p_i]$. Using this underlay path, it iterates through the list of already connected neighbours to compare the discovered path with the underlay paths to the neighbours. Similar to the landmark-based approach, three cases arise:

1. The underlay path to the connected neighbour is a prefix to the discovered underlay path to p_i, i.e. $Path^U[p_{current}, p_{neighbour}] \succ Path^U[p_{current}, p_i]$
2. The discovered underlay path is a prefix to the underlay path to the connected neighbour, i.e. $Path^U[p_{current}, p_i] \succ Path^U[p_{current}, p_{neighbour}]$
3. Neither case 1, nor case 2

In case a neighbour could be found, satisfying case 1, the neighbour is a better candidate to connect with. Hence, the current peer forwards the join-request to this neighbour p_{target} by sending a new join-request in the name of p_i. To avoid that the neighbour p_{target} has to do another traceroute measurement to p_i, the path $Path^U[p_{target}, p_i]$ is inferred from the two paths $Path^U[p_{current}, p_i]$ and $Path^U[p_{current}, p_{target}]$. Inferring a path from two other paths uses the fact that any prefix or suffix of a shortest path is usually also a shortest path. Because $Path^U[p_{current}, p_{target}]$ is a prefix to $Path^U[p_{current}, p_i]$ we can get $Path^U[p_{target}, p_i]$ by removing all routers of $Path^U[p_{current}, p_{target}]$ from $Path^U[p_{current}, p_i]$, except for the last one (which is the router to which p_i is connected to). For example consider the two paths $Path^U[p_1, p_2] = \langle r_1, r_2, r_3 \rangle$ and $Path^U[p_1, p_3] = \langle r_1, r_2, r_3, r_4, r_5 \rangle$. The path $Path^U[p_1, p_2]$ is obviously a prefix to $Path^U[p_1, p_3]$, hence we can determine $Path^U[p_2, p_3] = \langle r_3, r_4, r_5 \rangle$.

If there are neighbours satisfying case 2, these neighbours have to be disconnected and connected to p_i. First the algorithm at line 15 collects all these neighbours in the set *neighboursToAddToNewPeer*. While collecting, the current peer infers the path from the new peer p_i to the neighbour $p_{neighbour}$, which is $Path^U[p_i, p_{neighbour}]$ from the two paths $Path^U[p_{current}, p_i]$ and $Path^U[p_{current}, p_{neighbour}]$. This is done as explained above by removing all routers of $Path^U[p_{current}, p_i]$ from $Path^U[p_{current}, p_{neighbour}]$. After collecting all neighbours fulfilling case 2, it disconnects from all these neighbours (line 22). Then an acknowledgement is sent to p_i, containing the set *neighboursToAddToNewPeer* (line 24).

Similar to the PATHMATCH algorithm in the landmark-based approach, we will never have the situation that two peers both satisfy case 1 at the same time. To clarify this, consider two neighbour peers p_1 and p_2 both satisfying case 1. This means $Path^U[p_{current}, p_2] \succ Path^U[p_{current}, p_i]$ and $Path^U[p_{current}, p_2] \succ Path^U[p_{current}, p_i]$ is true. Hence, either $Path^U[p_{current}, p_1]$ is a prefix to

Algorithm 5.6 Algorithm executed when a peer receives a JOIN-Message in the random approach

```
1: receive JOIN(p_i, Path^U[p_current, p_i]) at peer p_current begin
2:     if Path^U[p_current, p_i] = NULL then
3:         Path^U[p_current, p_i] ← TRACEROUTE(p_i)
4:     end if
5:     p_target ← NULL
6:     Path^U[p_current, p_target] ← NULL
7:     neighboursToAddToNewPeer ← ∅
8:     for (p_neighbour, Path^U[p_current, p_neighbour]) ∈ neighbourList do
9:         if Path^U[p_current, p_neighbour] ≻ Path^U[p_current, p_i] then
10:            p_target ← p_neighbour
11:            Path^U[p_current, p_target] ← Path^U[p_current, p_neighbour]
12:        else if Path^U[p_current, p_i] ≻ Path^U[p_current, p_neighbour] then
13:            Path^U[p_i, p_neighbour] ← Path^U[p_current, p_neighbour] − Path^U[p_current, p_i]
14:            n ← (p_neighbour, Path^U[p_i, p_neighbour])
15:            neighboursToAddToNewPeer ← neighboursToAddToNewPeer ∪ {n}
16:        end if
17:    end for
18:    if p_target ≠ NULL then
19:        Path^U[p_target, p_i] ← Path^U[p_current, p_i] − Path^U[p_current, p_target]   // Remove overlapping
       prefix from the path to the new peer so that we get the path between our neighbour and the new
       peer
20:        SEND(JOIN, p_target, p_i, Path^U[p_target, p_i])   // Forward JOIN-Request to the target
       neighbour
21:    else
22:        disconnect from all neighbours in neighboursToAddToNewPeer
23:        neighbourList ← neighbourList ∪ {(p_i, Path^U[p_current, p_i])}
24:        SEND(ACK, p_i, p_current, neighboursToAddToNewPeer)
25:    end if
26: end
```

$Path^U[p_{current}, p_2]$ or vice versa. If $Path^U[p_{current}, p_1] \succ Path^U[p_{current}, p_2]$ was true, then p_2 would not have been added to $p_{current}$ but a join-request had been forwarded to p_1.

Also the situation where one neighbour satisfies case 1 and another neighbour satisfies case 2 is not possible. For clarification let us consider a neighbour peer p_1 satisfying case 1 and assume there was a second neighbour peer p_2 satisfying case 2. This means, $Path^U[p_{current}, p_1] \succ Path^U[p_{current}, p_i]$ and $Path^U[p_{current}, p_i] \succ Path^U[p_{current}, p_2]$ is true. Because of transitivity, hence $Path^U[p_{current}, p_1] \succ Path^U[p_{current}, p_2]$ is true. If this was the case when p_1 joined $p_{current}$ after p_2, peer p_2 would have been disconnected from $p_{current}$. If p_2 joined $p_{current}$ after p_1, the join-request would have been forwarded to p_1.

Algorithm 5.7 is executed when a peer receives an ACK-Message. The algorithm is very similar to Algorithm 5.6. Before doing any path-matching, a peer receiving an ACK-Message connects to all

Algorithm 5.7 Algorithm executed when a peer receives an ACK-Message in the random approach

```
1:  receive ACK(p_i, neighboursToAdd) at peer p_current begin
2:      for (p_neighbour, Path^U[p_current, p_neighbour]) ∈ neighboursToAdd do
3:          RECEIVE(JOIN, p_neighbour, Path^U[p_current, p_neighbour])
4:      end for
5:      Path^U[p_current, p_i] ←TRACEROUTE(p_i)
6:      p_target ←NULL
7:      Path^U[p_current, p_target] ←NULL
8:      addToNewNeighbour ← ∅
9:      for (p_neighbour, Path^U[p_current, p_neighbour]) ∈ neighbourList do
10:         if Path^U[p_current, p_neighbour] ≻ Path^U[p_current, p_i] then
11:             p_target ← p_neighbour
12:             Path^U[p_current, p_target] ← Path^U[p_current, p_neighbour]
13:         else if Path^U[p_current, p_i] ≻ Path^U[p_current, p_neighbour] then
14:             Path^U[p_i, p_neighbour] ← Path^U[p_current, p_neighbour] − Path^U[p_current, p_i]
15:             addToNewNeighbour ← addToNewNeighbour ∪ {(p_neighbour, Path^U[p_i, p_neighbour])}
16:         end if
17:     end for
18:     if p_target ≠NULL then
19:         neighbourList ← neighbourList ∪ {(p_i, Path^U[p_current, p_i])}
20:         if addToNewNeighbour ≠ ∅ then
21:             disconnect from all neighbours in addToNewNeighbour
22:             SEND(MULTI_JOIN, p_i, addToNewNeighbour)
23:         end if
24:     else
25:         disconnect from p_i
26:         Path^U[p_target, p_i] ← Path^U[p_current, p_i] − Path^U[p_current, p_target]
27:         SEND(JOIN, p_target, p_i, Path^U[p_target, p_i])
28:     end if
29: end
```

peers in a set called *neighboursToAdd* by calling Algorithm 5.6 for each of the peers. This will connect all peers in this set, while respecting the rules defined in the join-algorithm.

From the perspective of the sender p_i of the ACK-Message, there is no other peer in between the underlay path of the current peer and the sender peer. But from the perspective of the current peer, there may be such a peer connected to $p_{current}$, which would be a better candidate to connect to. This is the reason why the receiver tries to find a target neighbour first, similar to Algorithm 5.6. If no such target neighbour exists, the peer agrees to connect to the sender of the ACK-Message and adds it to the *neighbourList* (line 19 of Algorithm 5.7). However, there may be some peers which have to be disconnected, because the underlay path to the p_i is a prefix to the underlay path to them. In this case peer $p_{current}$ disconnects from all these neighbours and sends the list of peers to the new neighbour p_i by sending a MULTI_JOIN-Message (line 22).

Algorithm 5.8 Algorithm executed when a peer receives a `MULTI_JOIN`-Message in the random approach

```
1: receive MULTI_JOIN(peersToAdd) at peer p_current begin
2:     for (p_i, Path^U[p_current, p_i]) ∈ peersToAdd do
3:         RECEIVE(JOIN, p_i, Path^U[p_current, p_i])
4:     end for
5: end
```

When a peer receives the `MULTI_JOIN`-Message, it calls the join-algorithm 5.6 for each of the peers in the list (Algorithm 5.8). By this, all peer in the list will be connected to the peer, without the need of new traceroute measurements.

In the following, the complete algorithm is explained based on the example in Figure 5.6. Initially peers are connected as in Figure 5.6(a), i.e. p_2 is connected to p_3 and p_4, hence the set *neighbourList* of p_2, p_3 and p_4 is $\{(p_3,\langle r_2,r_3\rangle),(p_4,\langle r_2,r_1,r_4\rangle)\}$, $\{(p_2,\langle r_3,r_2\rangle)\}$ and $\{p_2,\langle r_4,r_1,r_2\rangle\}$, respectively. Let us assume p_3 sends a join-request to p_1 (Figure 5.6(b)). Upon receiving the join-request, peer p_1 initiates a traceroute measurement to p_3 to get $Path^U[p_1,p_3]=\langle r_1,r_2,r_3\rangle$ (Figure 5.6(c)). Because p_1 does not have any connected neighbour yet, it sends an `ACK`-Message to p_3 (Figure 5.6(d)). Now p_3 compares the underlay path to p_1 to the underlay path to its connected neighbour p_2. Path $Path^U[p_3,p_2]$ is a prefix to $Path^U[p_3,p_1]$, so p_2 is a better candidate to connect to. That is why p_3 disconnects from p_1 (Figure 5.6(e)). From the two underlay paths $Path^U[p_3,p_2]$ and $Path^U[p_3,p_1]$ peer p_3 can infer $Path^U[p_2,p_1]=\langle r_2,r_1\rangle$. Peer p_3 sends a join-request to p_2 in the name of p_1, adding the inferred underlay path (Figure 5.6(f)). Because $Path^U[p_2,p_1]$ has already been inferred, upon receiving the join-request from p_3, peer p_2 does not have to do a new traceroute measurement to p_1. Peer p_2 compares $Path^U[p_2,p_1]$ with the underlay path to it its connected neighbours p_3 and p_4. None of them is a prefix to $Path^U[p_2,p_1]$, hence p_2 sends an `ACK`-Message to p_1 (Figure 5.6(g)). Because $Path^U[p_2,p_1]$ is a prefix to $Path^U[p_2,p_4]$, peer p_2 has to disconnect from p_4 (Figure 5.6(h)) and let p_1 connect to p_4 by adding this information to the `ACK`-Message in Figure 5.6(g). Finally p_1 sends an `ACK`-Message to p_4 which establishes a connection between p_1 and p_4. As a result, the set *neighbourList* or p_1,p_2,p_3 and p_4 is $\{(p_2,\langle r_1,r_2\rangle),(p_4,\langle r_1,r_4\rangle)\}$, $\{(p_1,\langle r_2,r_1\rangle),(p_3,\langle r_2,r_3\rangle)\}$, $\{(p_2,\langle r_3,r_2\rangle)\}$ and $\{p_1,\langle r_4,r_1\rangle\}$, respectively. The constructed overlay topology matches the underlay topology.

5.2.3 Optimization

The random approach randomly selects *k* peers to traceroute. However, without considering the topology of real Internet, the same routers and links can be discovered multiple times. The Internet has a hierarchical structure in which smaller networks are interconnected through a higher level topology (for example Backbone-Routers). If a peer in one Autonomous System (AS) traceroutes many peers in another AS, links and routers on the path to border routers of the AS are repeatedly discovered. However, the majority of links is within AS's and hence not discovered.

Taking the structure of the Internet into consideration, a possible solution to avoid this problem is to prefer near peers when doing traceroute measurements. Measuring high numbers of intra-domain paths discovers more links and routers for the same number of traceroutes.

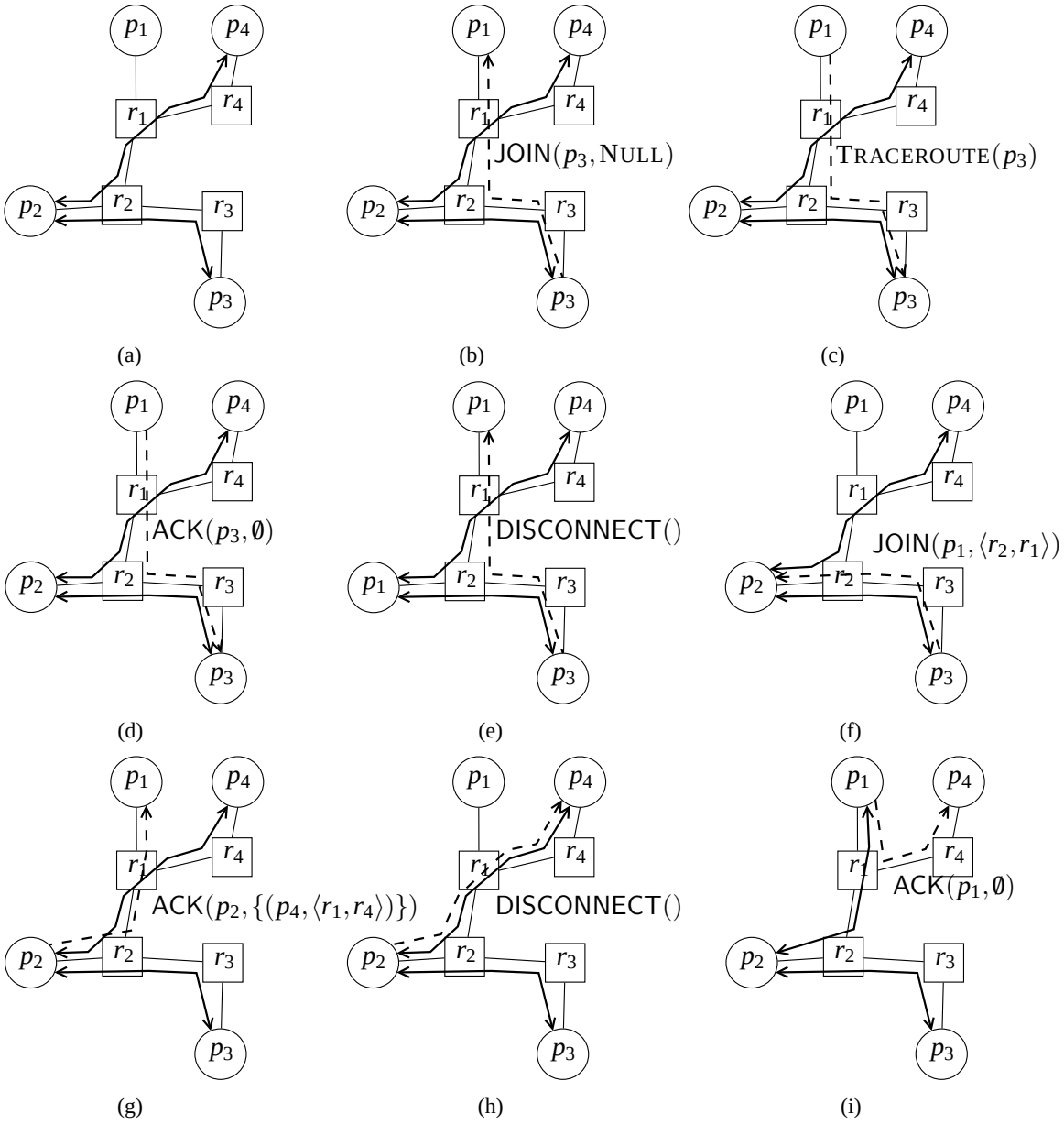


Figure 5.6: Random approach example

The optimization therefore selects the k nearest peers from random peer sample of size $k+m, m > 0$. The distance of peers is defined by measuring the Round-Trip-Time[2] (RTT) to a peer.

[2] Amount of time of a packet to travel to a neighbour peer and back to the source. Usually this is measured using a tool called *Ping*

In total the same amount of traceroute measurements is done by the peer as in the general random approach. However, because nearby peers are preferred, more intra-domain links are discovered which leads to a higher discovery ratio.

6 Reliable Publish-Subscribe

The reliable publish-subscribe system which is built on top of the TDO layer uses a minimal spanning tree based on failure-probabilities of links. Because the spanning tree contains links with minimal failure-probabilities, reliability of the spanning tree is maximised, hence such a tree will be called *Maximum-Reliability-Spanning-Tree* (MRST) in the following. Instead of building only one spanning tree, k spanning trees which use disjoint links whenever possible are built. The forest consisting of k spanning trees form the basis of the publish-subscribe routing substrate to forward subscriptions and route events.

The TDO layer described in Chapter 5 connects overlay peers according to the underlay topology. On top of this overlay topology a reliable publish-subscribe system is implemented. To increase reliability we introduce multiple paths in the publish-subscribe overlay network. However, peers in the overlay may fail at any time. By providing multiple overlay paths on top of the TDO, a failure on one overlay path usually does not affect all overlay paths. Because the TDO already provides underlay diverse links in the overlay, overlay paths get diverse in the underlay by constructing overlay paths which are link diverse in the overlay.

6.1 Maximum-Reliability-Spanning-Tree (MRST)

A MRST has the characteristic feature that it contains the most reliable links in the system. However, reliability is a multiplicative metric which means the reliability of a path is the product of the reliabilities of all components within the path. Hence, the end-to-end reliability of an overlay path may be still too low. In addition to that, peers may fail which causes a complete overlay path to fail.

In this section we will propose an algorithm where k edge disjoint MRSTs are constructed. The developed k-MRST algorithm is based on an approach by Young et al. [YKPW04], where a k-MST algorithm has been proposed. Constructing k MSTs guarantees k edge disjoint overlay paths between any two overlay nodes. By this fault-tolerance is increased and overlay path diversity is enabled.

In the following we will describe the k-MST approach of Young et al. Next, the intention for modifying the algorithm and modifications to the algorithm will be described.

The algorithm is based on the well-known GHS algorithm by Gallager et al. [GHS83] described in Section 2.3. We assume that the reader is familiar with the GHS-algorithm. The approach of Young et al. computes k-MSTs in a greedy fashion. The basic idea is to construct one MST after the other. The i^{th} MST is constructed using GHS by removing all edges used in all j^{th} MST where $j < i$.

$MST(G)$ represents the MST of graph G or a forest of disconnected MSTs in case G is not connected. $G - MST(G)$ is a subgraph of G where all edges of $MST(G)$ are removed from G.

Algorithm 6.1 Basic *k*-MST algorithm

$G_0 \leftarrow G$
$F^0 \leftarrow \emptyset$
for $1 \leq j \leq k$ **do**
  $F_j \leftarrow MST(G_{j-1})$
  $F^j \leftarrow F^{j-1} \cup F_j$
  $G_j \leftarrow G_{j-1} - F_j$
end for

The *k*-MST algorithm, however, computes the MSTs concurrently. Each node executes *k* instances of the GHS algorithm denoted as $MST_1, MST_2, \ldots, MST_k$. Initially all available edges are put into the set of basic edges of MST_1, which starts executing the GHS algorithm. Whenever an edge is put into the set of rejected edges on MST_i, it is handed over to the basic edges of MST_{i+1}. A MST_{i+1} can instantly use this edge because MST_i processes its basic edges in ascending order and hence also rejects edges in ascending order. This makes it possible to start constructing higher level MSTs while the construction of a lower level MST is not yet finished.

The *k*-MST algorithm solves the problem of constructing *k*-MST concurrently. However, the purpose of Young et al. to construct multiple MSTs is to build a subgraph out of the MST forest containing the *k* minimum weight links. It may be that in practice some MST_i are not connected but partitioned into several smaller trees so that there is not a path between any two peers on every MST. This can happen, if all edges are used up by a lower level $MST_j, j < i$. Connected MRSTs, however, are needed for subscription forwarding and notification routing (see Section 6.2 and 6.3).

To overcome this problem, higher level MRSTs should also be able to use links already part of a lower level MRST as a last alternative. The *k*-MST approach of Young et al. only hands over rejected links of a lower level MST to a higher level MST. In the approach proposed in this thesis also branch links are handed over to a higher level MRST so that the higher level MRST can make use of already used links. Therefore, each link is annotated with a level. The level of link $l^O_{i,j}$ is defined by $level[l^O_{i,j}]$ whereas the weight is defined by $weight[l^O_{i,j}]$. To construct a maximum reliability tree the weight of a link is defined using the failure probabilities of the overlay link and of the two end hosts p_i and p_j of link $l^O_{i,j}$:

$$weight[l^O_{i,j}] = 1 - [1 - \varphi(p_i)] \cdot [1 - \varphi(p_j)] \cdot [1 - \varphi(l^O_{i,j})]$$

One may note that the weight is actually not the reliability. It is the failure probability of the series system consisting of link $l^O_{i,j}$ and the two peers p_i and p_j, hence in order to calculate a maximum reliability tree, the value of $weight[l^O_{i,j}]$ has to be minimised.

We assume all reliability measurements have been done before starting the MRST algorithm. Additionally, we assume that all link weights are globally unique, which is a necessary precondition of the GHS algorithm. Even though this cannot be guaranteed in real world scenarios, the link weight of a link $l^O_{i,j}$ can always be made unique by involving the IDs of its two end hosts p_i and p_j.

In the following the i^{th} MRST is denoted as $MRST_i, i = 1,2,\ldots,k$. Each MRST has its own sets of links as defined in the GHS-Algorithm. The set of basic links of $MRST_i$ is denoted as $basic_i$ and contains all unprocessed overlay links available for $MRST_i$ to be tested. The set of branch links of $MRST_i$ is denoted as $branch_i$ and contains all overlay links part of $MRST_i$. Last, the set of rejected links is denoted as $rejected_i$ and contains all overlay links rejected by $MRST_i$.

Handing over an overlay link to a higher level MRST is done by Algorithm 6.2 for rejected links and Algorithm 6.3 for branch links respectively.

Algorithm 6.2 Algorithm to reject a link

```
1: procedure REJECTLINK(l^O_{i,j}, level)
2:     basic_level ← basic_level \ {l^O_{i,j}}
3:     rejected_level ← rejected_level ∪ {l^O_{i,j}}
4:     if level<k then
5:         level[l^O_i] ← level + 1
6:         basic_{level+1} ← basic_{level+1} ∪ {l^O_{i,j}}
7:     end if
8: end procedure
```

Algorithm 6.3 Algorithm to branch a link

```
1: procedure BRANCHLINK(l^O_{i,j}, level)
2:     basic_level ← basic_level \ {l^O_{i,j}}
3:     branch_level ← branch_level ∪ {l^O_{i,j}}
4:     if level<k then
5:         basic_{level+1} ← basic_{level+1} ∪ {l^O_{i,j}}
6:     end if
7: end procedure
```

When $MRST_i$ puts an overlay link into the set of branch links $branch_i$, the link is handed over to the set of basic links $basic_{i+1}$ of the next $MRST_{i+1}$ (Algorithm 6.3). The level of the link is not changed. However, when $MRST_i$ rejects an overlay link, the link is handed over to the set of basic links $basic_{i+1}$ of $MRST_{i+1}$ but the level of the link is increased (Algorithm 6.2). This mechanism allows to count the number of usages of an overlay link. If a link gets available for $MRST_i$, the MRST can determine the number of usages by subtracting the level of the overlay link from i (which is its own level), i.e. $usage\_count = i - level[l^O_{j,k}]$ for link $l^O_{j,k}$ on $MRST_i$.

This mechanism allows for an MRST to sort its available links. First, only links rejected by a lower level MRST should be probed, then if no such link is available any more, also branch links of lower level MRSTs can be used to get a single connected tree. A MRST defines the order within the set of basic links as follows:

$$l^O_{i,j} < l^O_{i,k} \iff (level[l^O_{i,j}] > level[l^O_{i,k}]) \vee (level[l^O_{i,j}] = level[l^O_{i,k}] \wedge weight[l^O_{i,j}] < weight[l^O_{i,k}]) \quad (6.1.1)$$

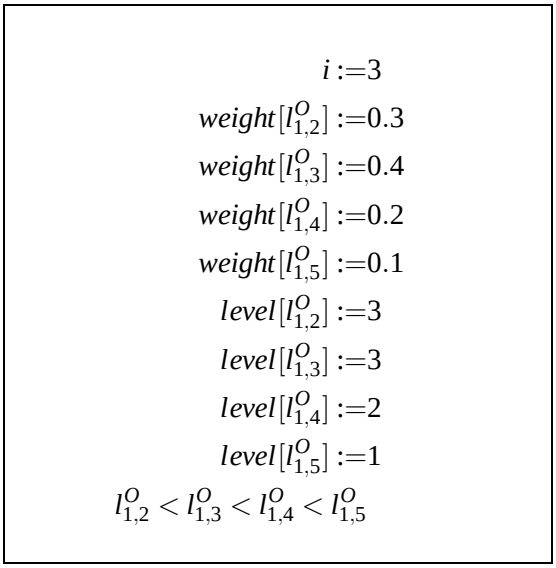


Figure 6.1: Example for link order in *k*-MRST

Applying this definition, a higher level MRST reuses links which are already part of a lower level MRST only in case that there is no unused link any more, because unused links would always be smaller with respect to the definition.

In Figure 6.1 the order definition is demonstrated. Link $l^O_{1,2}$ is the smallest, because it has never been used before ($i - level[l^O_{1,2}] = 3 - 3 = 0$) and has the minimum weight among the links with the same level. Link $l^O_{1,3}$ has a higher weight than $l^O_{1,2}$ and hence is worse. Even though $l^O_{1,4}$ and $l^O_{1,5}$ have a lower weight, their level is lower as well, which is the reason why they are both worse than $l^O_{1,2}$ and $l^O_{1,3}$. Link $l^O_{1,5}$ is the worst because it has already been used in two other MRSTs ($i - level[l^O_{1,5}] = 3 - 1 = 2$).

If $MRST_i$ only hands over rejected links to $basic_{i+1}$, these links can instantly be used by $MRST_{i+1}$ because links are rejected in increasing order by the link weight. However, we are also handing over links put into the set of branch links. By handing over branch links of $MRST_i$ to the set of basic links $basic_{i+1}$ of $MRST_{i+1}$, in some cases the link cannot be used instantly. This is the case, if there are still links left in $basic_i$ which could be rejected by $MRST_i$ and hence would be smaller with respect to equation (6.1.1).

To solve the problem, a $MRST_i$ has to watch the set of basic edges of all $MRST_j, 1 \leq j < i$. Whenever a new basic link needs to be tested by the GHS algorithm, the following condition must be satisfied.

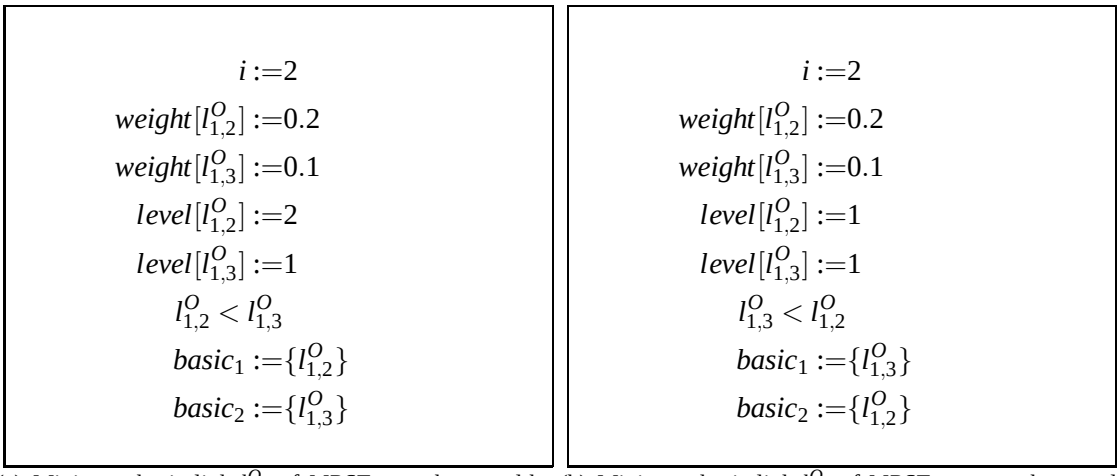
The minimum basic link on $MRST_i$ (with respect to equation (6.1.1)), denoted by $\min(basic_i)$, can be tested if and only if:

$$\forall j, 1 \leq j < i : \; basic_j = \emptyset \vee (level[\min(basic_i)] = i \wedge weight[\min(basic_i)] < weight[\min(basic_j)]) \tag{6.1.2}$$

Equation (6.1.2) defines a constraint which has to hold for any edge tested by a MRST. There are two possible cases:

1. The level of the minimum basic link $\min(basic_i)$ equals the level of the MRST that wants to test the link, i.e. when $MRST_i$ has to test link $l^O_{i,j}$, the following condition is true $level[l^O_{k,h}] = i$. In that case there must not be any other link in $basic_j, 1 \leq j < i$ with a lower weight than the minimum basic link of $MRST_i$. Let us assume there was a lower weight link in $basic_j$. If this link gets rejected by $MRST_j$ it is made available to $MRST_i$ and should be preferred to $l^O_{i,j}$.
2. The level of the minimum basic link $\min(basic_i)$ is lower than the level of MRST that has to test the link, i.e. when $MRST_i$ has to test $l^O_{i,j}$, the following condition is true $level[l^O_{k,h}] < i$. In this case the link has been used at least once by $MRST_j, 1 \leq j < i$. We can only test such a link, if there are no more links in any of the sets $basic_j$ of a level $MRST_j, 1 \leq j < i$.

If $MRST_i$ has to test a new link but its minimum basic link $\min(basic_i)$ is not satisfying equation (6.1.2), $MRST_i$ has to wait until $MRST_{i-1}$ hands over a new link to $basic_i$, which satisfies the condition. As soon as such a link is added to $basic_i$, the algorithm of $MRST_i$ can continue. However, as long as enough links are available, links get rejected very quickly by a lower level MRST so that a higher level MRST usually does not have to wait until the lower level MRST is finished.



(a) Minimum basic link $l^O_{1,3}$ of $MRST_2$ may be tested by $MRST_2$

(b) Minimum basic link $l^O_{1,3}$ of $MRST_2$ may not be tested by $MRST_2$

Figure 6.2: Example demonstrating the testing condition of *k*-MRST

In Figure 6.2 an example of two different situations demonstrating the testing condition is shown. In Figure 6.2(a) the minimum basic link of $MRST_2$ is $l^O_{1,3}$. The link can be tested, because the link's level is 2 and there is no basic link in $MRST_1$ which has a lower weight than $l^O_{1,3}$. In contrast Figure 6.2(b) shows a situation in which $MRST_2$ has to wait for $l^O_{1,3}$ to be added to $basic_2$. Link $l^O_{1,2}$ is the minimum basic link of $MRST_2$. This situation may occur due to the absorption of another MRST fragment (see Section 2.3), which causes links that are not minimal within the set of basic links to be added to the set of branch links $branch_1$.

6.2 Subscription-Forwarding on MRST

A MRST is an ideal substrate for subscription forwarding. Because a MRST is a tree, it is guaranteed that no cycles exist. This is an important characteristic trait for covering-based and merging-based subscription forwarding. In covering-based publish-subscribe systems, only subscriptions are forwarded which are not covered by a previously forwarded subscription. In merging-based publish-subscribe systems, subscriptions are merged with previously forwarded subscriptions and only the merger is forwarded, which ideally exactly covers the subspace of the event space of all subscriptions.

In this work we will only provide a simple identity-based subscription forwarding algorithm. In general, however, it is easy to exchange it by a covering-based or merging-based approach.

To use the property of a cycle-free graph, each MRST constructed by the approach in Section 6.1 is used separately for subscription forwarding. This means that for each of the k MRSTs a separate subscription routing table $RT_i, 1 \leq i \leq k$ exists. A peer subscribing to a new filter broadcasts its subscription along each tree. Therefore a subscription message contains the level of the tree the message is currently forwarded on.

Algorithm 6.4 Procedure to subscribe for a subscription

```
1: procedure SUBSCRIBE(subscription)
2:     localSubscriptions ← localSubscriptions ∪ {subscription}
3:     for 1 ≤ i ≤ k do
4:         for neighbour ∈ MRST_i do
5:             SEND(SUBSCRIBE, neighbour, currentPeerId, subscription, i)
6:         end for
7:     end for
8: end procedure
```

Algorithm 6.4 shows the procedure executed when an application subscribes for a subscription. First the subscription is added to the set of local subscriptions *localSubscriptions*. Then the subscription is forwarded to each neighbour on each MRST.

Algorithm 6.5 Algorithm executed when a subscription is received

```
1: receive SUBSCRIBE(sender, subscription, level) at peer p begin
2:     RT_level ← RT_level ∪ {(subscription, destination)}
3:     for neighbour ∈ MRST_level do
4:         if neighbour ≠ sender then
5:             SEND(SUBSCRIBE, neighbour, p, subscription, level)
6:         end if
7:     end for
8: end
```

When a peer receives a subscription a new routing table entry is added to the routing table RT_i, where i is the level of the MRST the subscription has been forwarded on (Algorithm 6.5). Then the subscription is forwarded to all neighbours on the MRST, except for the sender of the subscription.

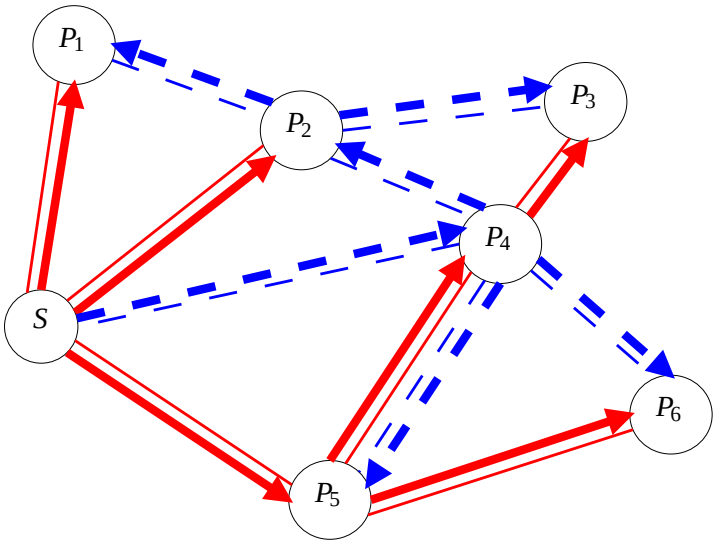


Figure 6.3: Example for subscription forwarding on a 2-MRST

In Figure 6.3 an example of subscription forwarding on a k-MRST is shown. The thin lines refer to the branch links of the MRSTs of which the thin red solid lines are the branch links of $MRST_1$ and the blue dashed lines are branch links of $MRST_2$. The node S is the subscriber that initiates the subscription forwarding process, whereas the nodes P_i are potential publishers from the perspective of S. The thick arrows describe the path a subscription uses while it is forwarded. The red solid arrow describes the path a subscription is forwarded along $MRST_1$ and the blue dashed lines describe the path on $MRST_2$. We can see that the solid and dashed paths are mostly overlay-link-disjoint, there is only one link which is used by both MRSTs.

6.3 Notification-Routing on MRST

The subscription forwarding procedure creates k routing tables RT_i on each peer. An event can potentially be published by any peer in the network. When a peer publishes an event, Algorithm 6.6 is executed. An event is simply forwarded on each of the k MRSTs, according to all routing tables.

Algorithm 6.6 Algorithm to publish an event

```
1:  procedure PUBLISH(event)
2:      if event ∈ localSubscriptions then
3:          pass event to the application layer
4:      end if
5:      for 1 ≤ i ≤ k do
6:          destinations ← ∅
7:          for (subscription, dest) ∈ RT_i do
8:              if event ∈ subscription then
9:                  destinations ← destinations ∪ {dest}
10:             end if
11:         end for
12:         for dest ∈ destinations do
13:             if dest ≠ sender then
14:                 SEND(NOTIFY, dest, p_current, event, i)
15:             end if
16:         end for
17:     end for
18: end procedure
```

Algorithm 6.7 Algorithm executed when a notification is received

```
1:  receive NOTIFY(sender, event, i) at peer p_current begin
2:      if event ∈ localSubscriptions then
3:          pass event to the application layer
4:      end if
5:      destinations ← ∅
6:      for (subscription, dest) ∈ RT_i do
7:          if event ∈ subscription then
8:              destinations ← destinations ∪ {dest}
9:          end if
10:     end for
11:     for dest ∈ destinations do
12:         if dest ≠ sender then
13:             SEND(NOTIFY, dest, p_current, event, i)
14:         end if
15:     end for
16: end
```

When a peer receives an event, Algorithm 6.7 is executed. First it checks whether any local subscription matches the event and if so, it hands the event over to the application layer. Next, the peer forwards the event on the MRST it received the event on, according to the corresponding routing table RT_i.

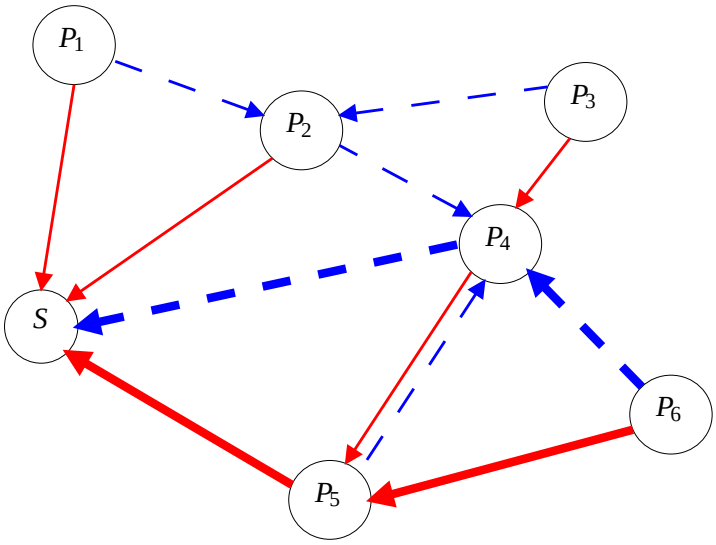


Figure 6.4: Example for notification forwarding on a 2-MRST

In Figure 6.4 an example of the routing entries added in the graph of example Figure 6.3 is shown. The solid red arrows show the routing entries on RT_1 and the dashed blue arrows show the routing entries of RT_2. The thick arrows show the two paths an event uses when publisher P_6 publishes an event matching a subscription of S.

7 Evaluation

In this chapter evaluations of the approaches described in Chapter 5 and 6 will be shown. First the simulation environment and configuration will be described, then the topology metrics will be shown. In Section 7.2 evaluation results of the TDO layer will be presented and in Section 7.3 results for the publish-subscribe layer will be revealed.

7.1 Simulation Configuration

7.1.1 Simulation Environment

All evaluations have been done in PeerSim [MJ09], which is a scalable open source P2P simulator written in Java. PeerSim provides two simulation engines, a cycle-based and an event-driven one. For our simulations the event-driven engine has been used, which allows a more realistic simulation of event-driven protocols. The simulator models propagation delay and link and node failures. Simulation attributes specifically for the simulated layers are described in detail in Sections 7.2 and 7.3.

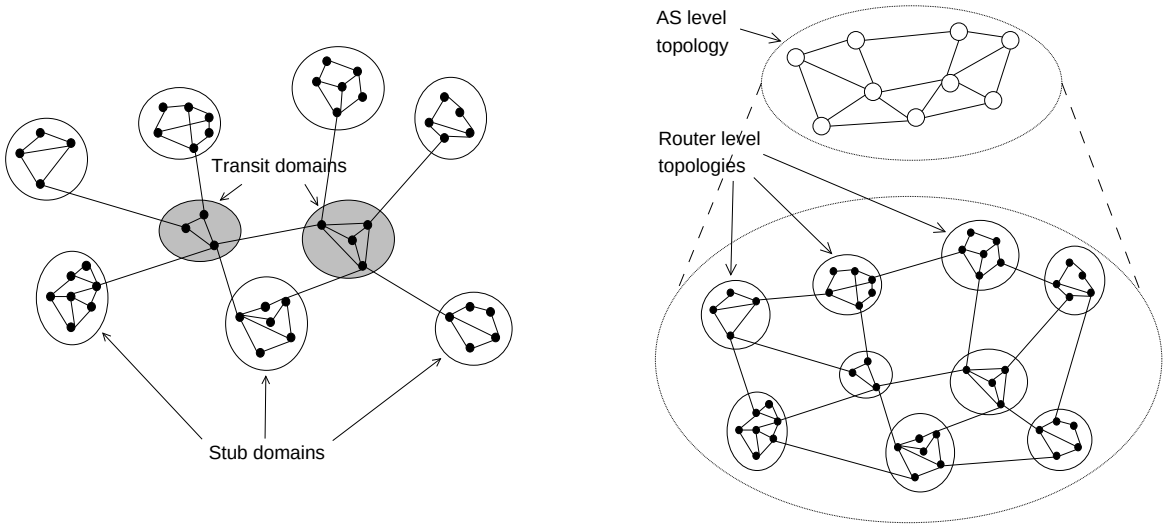
7.1.2 Topology

The simulation is performed on two different sets of topologies. The first set is generated by GT-ITM [GT-00] using a Transit-Stub topology model. The second set is generated by BRITE [MLMB01]. Both, the Transit-Stub model and BRITE generate a multi-level hierarchy. Transit-Stub generates interconnected higher level transit domains and lower level stub domains which are connected through the transit domains (Figure 7.1(a)). BRITE uses a top-down approach which first generates a high level AS topology, and for each of the AS nodes a low level router topology is generated (Figure 7.1(b)).

To generate the Transit-Stub topology the following parameters have been used: Each transit node is connected to 4 stub domains. In total there are 4 transit domains which are fully connected. Each transit domain and each stub domain contains 10 nodes. So in total the topology consists of $4 \cdot 10 \cdot (1 + 4 \cdot 10) = 1640$ nodes. The Transit-Stub topologies have about 7900 edges.

The BRITE topology has the same number of nodes. It has been generated by a top-down approach with 20 AS-level nodes and 82 router-level nodes for each AS node which results in $20 \cdot 82 = 1640$ nodes in total. All BRITE topologies have 3320 edges.

For each type five random topologies have been generated available as input data for simulations.



(a) Transit-Stub (b) BRITE

Figure 7.1: Topology models

7.2 Topology-Inference

In this section the simulation results of the TDO layer will be discussed. All simulations have been performed three times on each of the five topologies so in total 15 simulation runs have been made for each input parameter. The results represent the average value of simulation runs.

In each simulation 250 nodes out of the 1640 available routers are randomly selected to connect an overlay peer to. However, peers are only connected to routers in the stub-domain (for Transit-Stub topologies) or to router-level nodes (for BRITE topologies).

To measure the degree of topology discovery, the following metrics are used:

Link Discovery Ratio This is the fraction of discovered links in the underlay to the total number of underlay links in all $N \cdot (N-1)$ paths between all N peers. This metric provides a measurement of the degree of topology discovery.

Traceroute Efficiency Average fraction of new discovered links in each traceroute:

$$\frac{numberOfDiscoveredLinks}{numberOfTraceroutes \cdot totalNumberOfLinks}$$

This metric gives a hint about the discovery performance of the algorithm.

One of the targets of the overlay formation layer is to construct an overlay which reflects the underlay network. The following metrics are used to measure the equality of the overlay to the underlay topology:

Stretch Stretch is the average ratio between the propagation delay on the shortest overlay path to the propagation delay on the shortest path in the underlay:

$$\frac{overlayDelay}{underlayDelay}$$

The Stretch does not include the last mile delay between a router and a peer. Without considering the last mile delay, a stretch of one means that the overlay paths exactly matches the underlay paths, because the delay of both paths is the same. This metric hence can give a measurement about how good the overlay topology reflects the underlay topology.

Hop Ratio The Hop Ratio is the ratio of the number of peer-routers in the underlay path to the number of hops in the shortest overlay path between two peers. A peer-router is an underlay router which has a directly connected overlay peer:

$$\frac{peerRoutersInUnderlay}{peersInOverlayPath}$$

For example let us consider the network in Figure 7.2. If we assume the underlay path between peers p_2 and p_3 is $Path^U[p_2,p_3] = \langle r_3, r_2, r_1 \rangle$ and shortest overlay path between p_2 and p_3 is $Path^O[p_2,p_3] = \langle p_2, p_3 \rangle$ according to the overlay topology constructed by the TDO, the *Hop Ratio* value is $\frac{3}{2} = 1.5$. If the overlay topology reflects the underlay topology, the number of overlay peers within the overlay path should be equal to the number of peer-routers in the underlay path, hence the *Hop Ratio* should converge to one.

Path Match This metric consists of two parts. The first part is the number of peer-routers in an underlay path whose connected peer is not in the shortest overlay path between the same two peers, divided by the total number of peer routers in this path. The second part is the number of peers in the overlay path whose router is not part of the underlay path between the same two peers:

$$\frac{numberOfPeerRoutersNotInOverlayPath}{totalNumberOfPeerRoutersInUnderlayPath} + \frac{peersNotInUnderlayPath}{numberOfPeersInOverlayPath}$$

Let us consider the example in Figure 7.2. If we assume the underlay path between p_2 and p_4 is $Path^U[p_2,p_4] = \langle r_3, r_2, r_4, r_5 \rangle$ and the respective overlay path is $Path^O[p_2,p_4] = \langle p_2, p_3, p_4 \rangle$ according to the overlay topology constructed by the TDO, the *Path Match* value is $\frac{1}{3} + \frac{1}{3} \approx 0.67$. If the overlay topology reflects the underlay topology, all routers with a connected peer in an underlay path between two nodes should also be part of the overlay path, hence this metric should ideally converge to zero. This metric also gives a measurement on how good the overlay network reflects the underlay topology.

All simulations have been performed against the number of neighbours k. The parameter k is defined differently for each simulated approach and will be explained in the following.

The following approaches have been simulated:

Random This refers to the random approach described in Section 5.2. The parameter k is the size of the random peer sample a peer initiates traceroute measurements to. In order to have a connected graph for all simulations of **Random** we only simulate with $k \geq 2$.

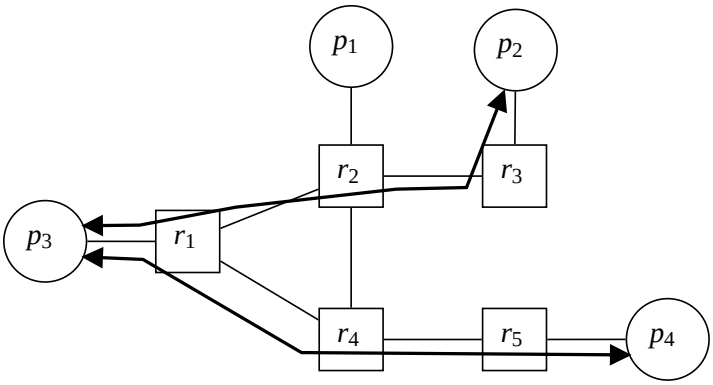


Figure 7.2: Example to explain the *Hop Ratio* and *Path Match* metrics

Random (*k* nearest neighbours) This approach refers to the optimization described in Section 5.2.3. Initially the algorithm selects the *k* nearest peers out of its random peer sample of size $(k+10)$. The *k* selected peers are used in the same way as in **Random**.

Random (*k* farthest neighbours) For comparison also simulations have been made where the random approach selects the *k* farthest peers from the available random peer sample of size $(k+10)$.

Landmark This refers to the landmark approach described in Section 5.1. In these simulations the parameter *k* defines the number of landmarks in the system.

7.2.1 Link Discovery Ratio

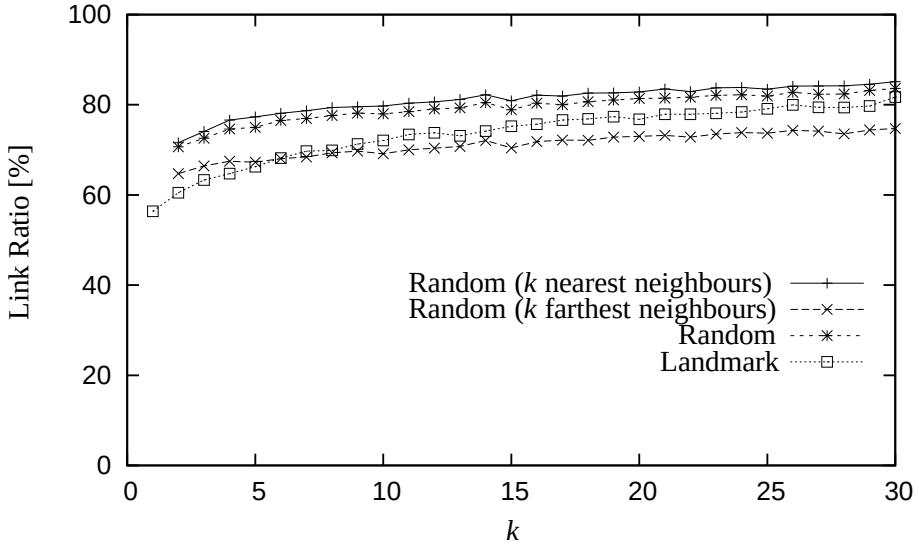
The *Link Discovery Ratio* is the fraction of discovered underlay links to all underlay links in the set of $N \cdot (N-1)$ paths between all *N* peers. The more links are discovered, the more topology is discovered. As much underlay topology as possible should be discovered to increase diversity in the underlay.

In Figure 7.3 the Link Discovery Ratio results are shown. As we can clearly see, **Random (*k* farthest neighbours)** performs worst. This is due to the fact that the same set of links are discovered multiple times by the same peer. **Landmark** performs better but only with a high number of landmarks. The best results have been realised by **Random** and **Random (*k* nearest neighbours)**.

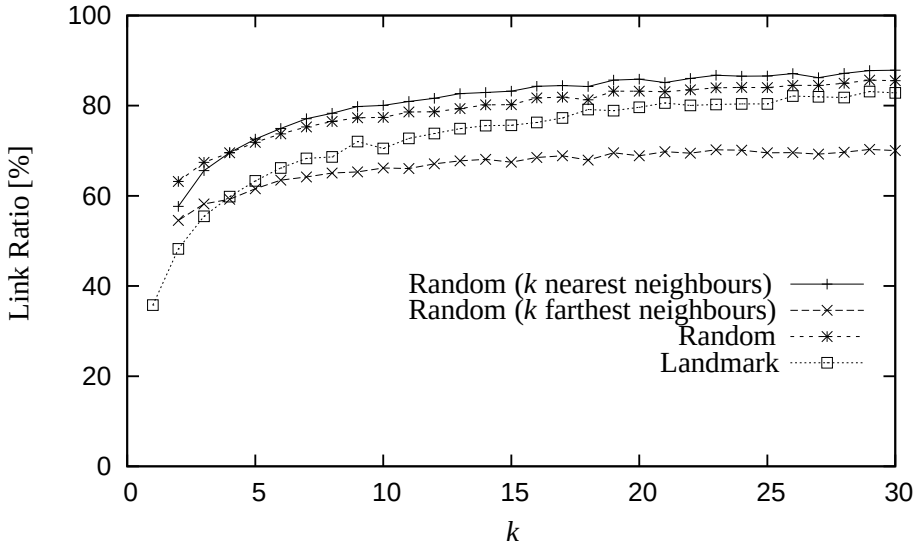
7.2.2 Traceroute Efficiency

Traceroute Efficiency is defined as the fraction of links discovered in each traceroute. This gives a hint about the efficiency of the algorithm. The higher the Traceroute Efficiency, the more of the topology is discovered by an algorithm for a given number of traceroute measurements.

According to the results in Figure 7.4 **Random (*k* nearest neighbours)** is most efficient. This is because this approach discovers more intra-domain links than the other approaches because it prefers to traceroute peers which are near. **Landmark** only has high traceroute efficiency with higher number of landmarks. This is most likely because it traceroutes a high number of both, intra-domain peer and peers farther away.



(a) Transit-Stub topology

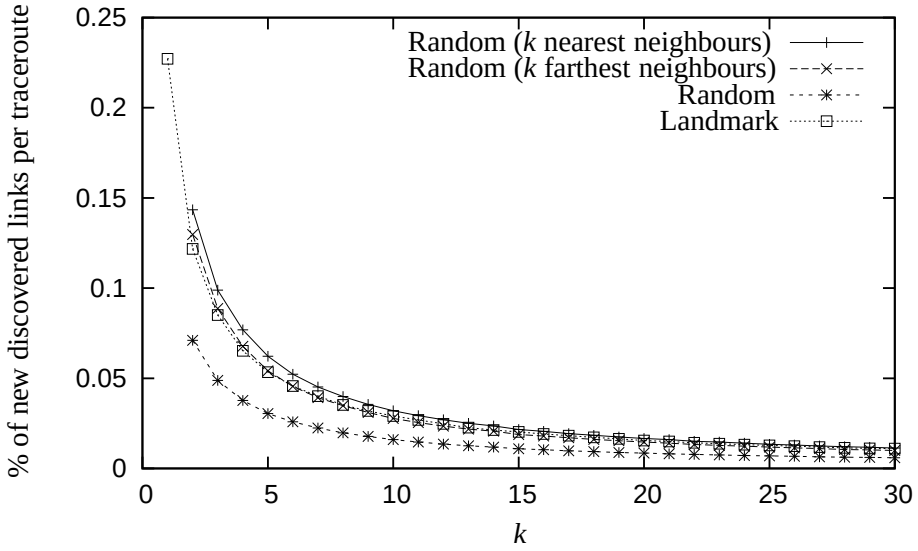


(b) BRITE topology

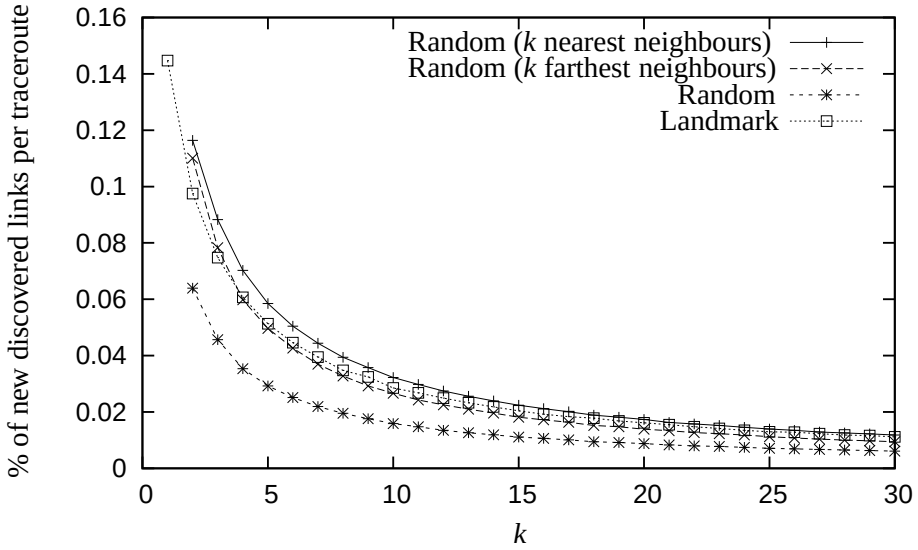
Figure 7.3: Link Discovery Ratio

7.2.3 Stretch

The *Stretch* compares the underlay path delay to the delay produced on the shortest overlay path between two peers. The underlay path delay is the shortest possible delay to transport a package between two peers. The more the overlay path delay equals the underlay path delay, the higher is the



(a) Transit-Stub topology



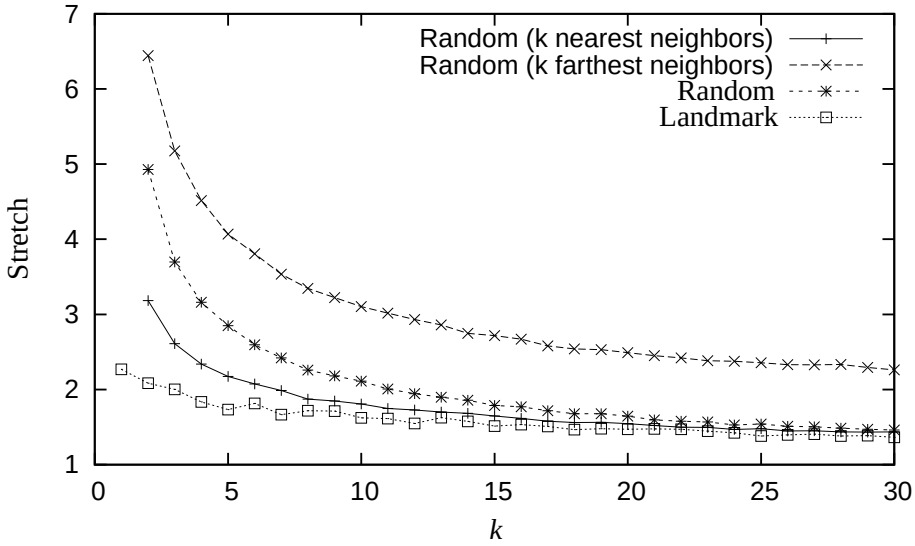
(b) BRITE topology

Figure 7.4: Traceroute Efficiency

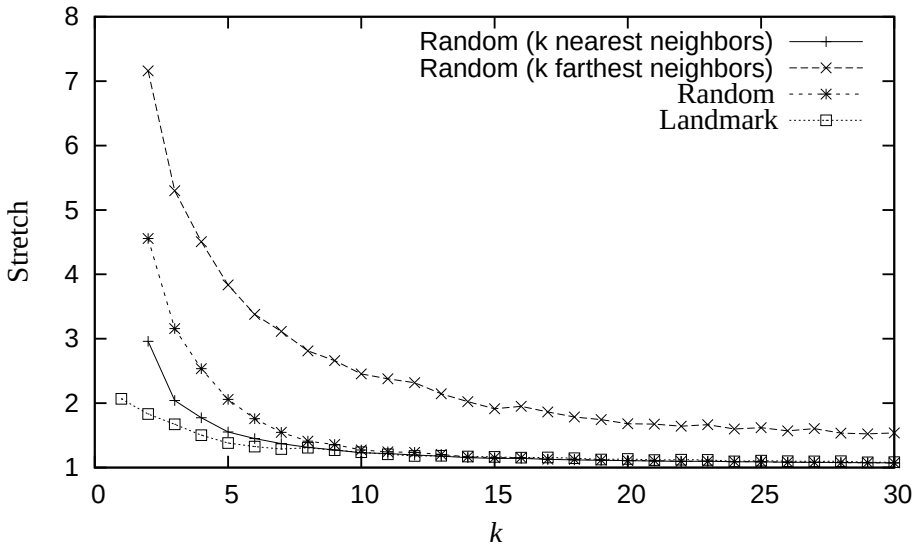
probability that the overlay path is similar to the underlay path. For stretch calculation we ignore the last mile delay which increases the stretch and distorts the results.

In Figure 7.5 the results of stretch measurements on the constructed overlay are shown. One can clearly see that **Random (*k* farthest neighbours)** performs worst. This is because it does not know many of the intra-domain links and hence often routes traffic through other AS which causes high

stretch. **Landmark** seems to perform best, however the landmark approach produces high node degree on the landmark peers. Even in the case $k = 1$ all peers are connected to the landmark through a path which perfectly matches the shortest underlay path. In the worst case all traffic is routed in the overlay through the landmark. Hence, the stretch cannot be much more than two. **Random** and **Random (*k* nearest neighbours)** both converge to a good stretch very fast.



(a) Transit-Stub topology



(b) BRITE topology

Figure 7.5: Stretch (without considering last mile delay)

Stretch is a good metric to measure equality of the overlay topology to the underlay topology. However, it can be easily distorted in the case of high node degree. For example in a mesh where each peer is connected to any other peer through an overlay link, the stretch is optimal, however the overlay does not reflect the underlay topology. This is the reason why the two following metrics will be introduced.

7.2.4 Hop Ratio

If the overlay reflects the underlay topology, overlay paths should be similar to the corresponding underlay paths. *Hop Ratio* describes the ratio of the number of routers with a connected peer in the underlay to the hop count in the corresponding overlay path. The value should converge to one, which most likely is an indicator that the overlay path equals the underlay path.

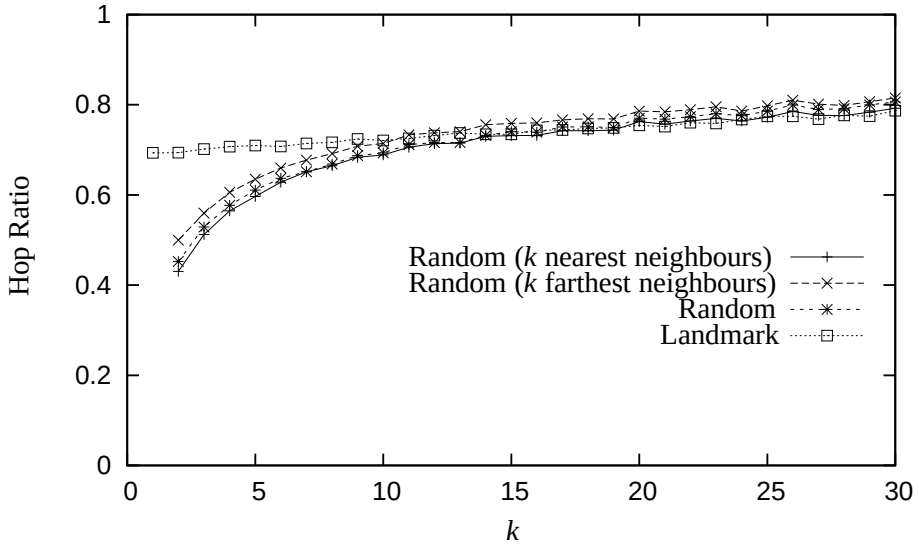
In Figure 7.6 the results of the Hop Ratio measurements are plotted. **Landmark** performs best for a small value of k. The reason for this most likely is that even in case $k = 1$ **Landmark** perfectly matches the shortest paths from the landmark to all peers. Hence also the shortest overlay paths from and to the border-routers of an AS perfectly match the corresponding underlay path. The path from and to the border-routers however make a large fraction of the total underlay path between any two peers. Peers are only connected to routers within an AS but not to routers in the transit-domain, which is the reason why routers in the transit-domain do not affect the Hop Ratio.

On a BRITE topology, **Random** performs best among the random approach simulations. Hop Ratio measures the ratio of peers connected to routers in an underlay to the peer count on the overlay path. However, it does not test whether peers along paths match each other. For example an overlay path which has the same number of peers as the number of peers connected to routers on the corresponding underlay path would result in a perfect Hop Ratio, even in the case in which the actual peers are totally different. The *Path Match* metric considers this problematic.

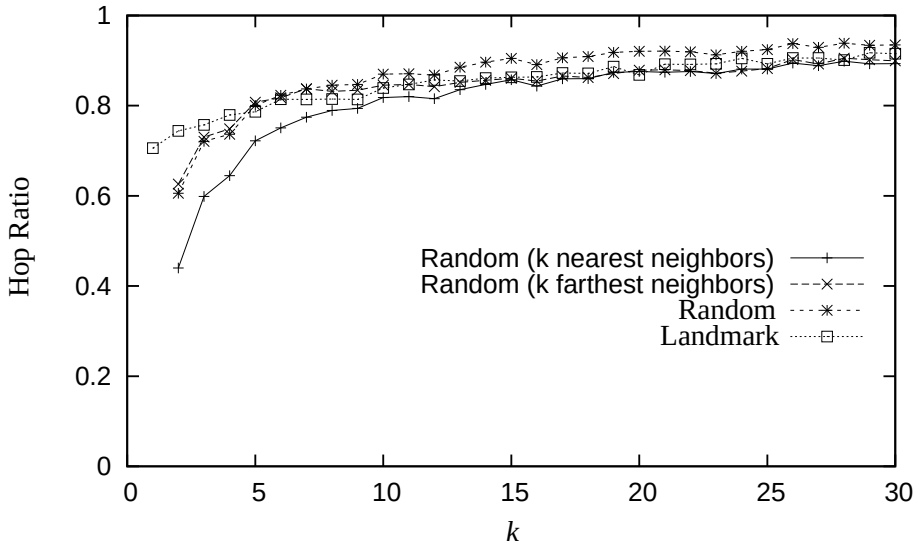
7.2.5 Path Match

Path Match is a comparison metric between underlay and overlay paths. It is defined as the fraction of peers in the overlay path with no corresponding router with the same connected peer in the underlay path, plus the fraction of routers with connected peers in the underlay path where no corresponding peer in the overlay path exists. A Path Match value of zero means that all overlay paths exactly match the corresponding underlay path, whereas the worst possible Path Match value two means that overlay path and underlay path are totally different.

In Figure 7.7 the Path Match results are shown. **Landmark** performs best on both, Transit-Stub and BRITE topologies. Again, the most likely reason for this is that **Landmark** discovers and reflects the path from and to the border routers for each peer. The random approaches perform similar in Transit-Stub, whereas there is a significant difference in BRITE. **Random** fast converges to the Path Match value of **Landmark** and seems to be best among the random approaches.



(a) Transit-Stub topology

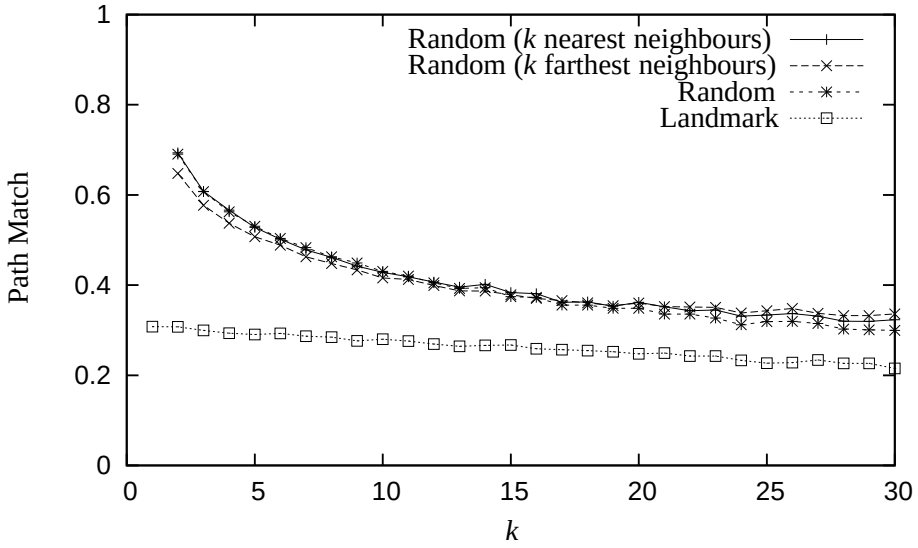


(b) BRITE topology

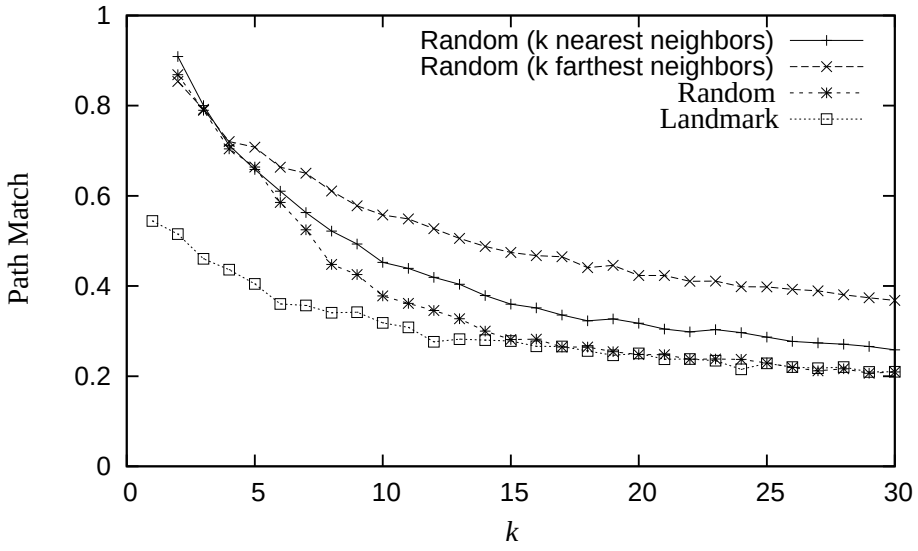
Figure 7.6: Hop Ratio

7.2.6 Summary

Simulation results for the TDO show that all approaches discover a large fraction of the underlay topology. All construct an overlay topology reflecting the underlay topology. In general we can say **Landmark** performs best of all approaches, however it has some scalability drawbacks because a landmark has to do traceroute measurements to all other peers in the network. As expected, **Random**



(a) Transit-Stub topology



(b) BRITE topology

Figure 7.7: Path Match

(*k* farthest neighbours) performs worst because it is poor in discovering intra-domain topology and this is critical with regard to discovering the underlay topology in the presence of a hierarchical Internet topology. However, it is not absolutely clear which of the other random approaches performs better.

7.3 *k*-MRST and Publish-Subscribe

In this section the simulation results of the *k*-MRST and publish-subscribe layer will be presented. Similar to previous simulations all simulations have been performed on both, the Transit-Stub and BRITE topologies. For each distinct parameter value 15 experiments were performed.

To each underlay router, underlay link and peer a random reliability value has been assigned. In order to generate reliabilities for system components random values distributed according to a modified (inverted) version of the Pareto distribution[1] between 0.85 and 1.0 have been generated. In Figure 7.8 the cumulative reliability distribution for different values of the parameter α is shown. The selection of α depends on the type of component, the reliability has been assigned to. In Table 7.1 the distribution curve used for each component type is shown.

Component type	α
Default underlay router	50
Transit domain router	100
Stub domain link/Router level link	50
Border link	80
Transit link	100
Last mile link to peer	200
Peer	100

Table 7.1: Usage of reliability value distribution based on component type

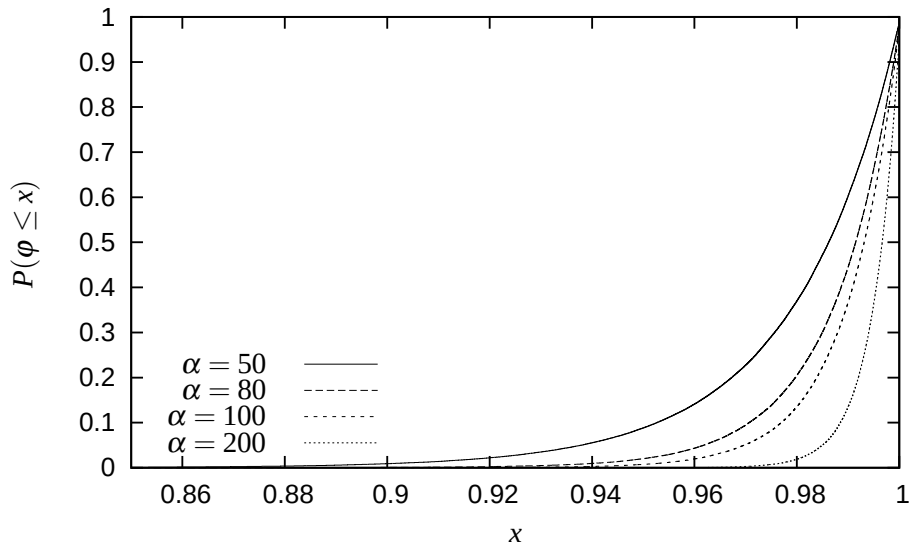


Figure 7.8: Cumulative distribution of reliability values assigned to links and nodes

[1] http://en.wikipedia.org/wiki/Pareto_distribution

To evaluate the simulation results we did measurements on the underlay path length of the shortest underlay paths. Specifically, we measured the average path length and the maximum path length (diameter) within the underlay topology. Furthermore, we calculated the average and minimum underlay path reliability. The results are presented in Table 7.2.

	Transit-Stub topology	**BRITE topology**
Average Path Length	6.97	13.49
Diameter	10	32
Average Path Reliability	88.63%	53.88%
Minimum Path Reliability	68.12%	4.18%

Table 7.2: Topology metrics

In each simulation 250 nodes out of the 1640 available routers have been selected randomly to connect an overlay peer to. Again, peers are only connected to routers in a stub-domain (for Transit-Stub topologies) or to router-level nodes (for BRITE topologies).

Simulation results are plot against the number of constructed MRSTs.

For publish-subscribe simulations a two-dimensional event space where each dimension can take a number between 0 and 1000 is considered (for example $(25,908)$). A subscription specifies a random range for each dimension which constructs a rectangle within the event space. All events published within the rectangle are considered matching events (a matching subscription for the above event is for example $([0,30],[900,920])$). In total 100 subscriptions have been subscribed and 5000 events have been published separately by random peers.

The goal of the publish-subscribe layer is to deliver events to subscribers reliably. Therefore some metrics have been defined which measure reliability and availability properties. The following metrics have been defined to measure the performance of the different approaches:

Underlay Link Resilience Resilience is a way to measure the tolerance against failures in a system. It is defined as the minimum number of components that have to fail simultaneously so that the complete system fails. The resilience of a single path, for example, is always one, because if any router or link on the path fails, the complete path fails. However, the *k*-MRST layer established multiple paths. The Underlay Link Resilience is defined as the minimum number of underlay links that have to fail simultaneously so that all paths of the *k* established paths between two peers fail. The values presented here are average values for all $N \cdot (N-1)$ overlay paths.

Reliability The *k*-MRST layer establishes *k* overlay paths between any pair of peers. This metric measures the reliability of all these multiple paths, which is the probability that a message sent over each of the established paths at the same time is delivered at the receiver. This is calculated by using equation (2.1.2) defined on page 23.

Event Ratio Ratio of the number of received events to the number of expected receives. If, for example, one event is published and two subscribers exist, which have a matching subscription, the expected number of receives is two. If only one of the subscribers receives the event, the Event Ratio is $\frac{1}{2} = 50\%$.

To measure how the TDO affects the performance of the *k*-MRST and the publish-subscribe layer, two types of simulations have been conducted. The first simulates the *k*-MRST and publish-subscribe layer on top of the TDO layer. The second simulates both higher level layer on top of a randomly connected overlay. Figure 7.9 illustrates the two types of simulation.

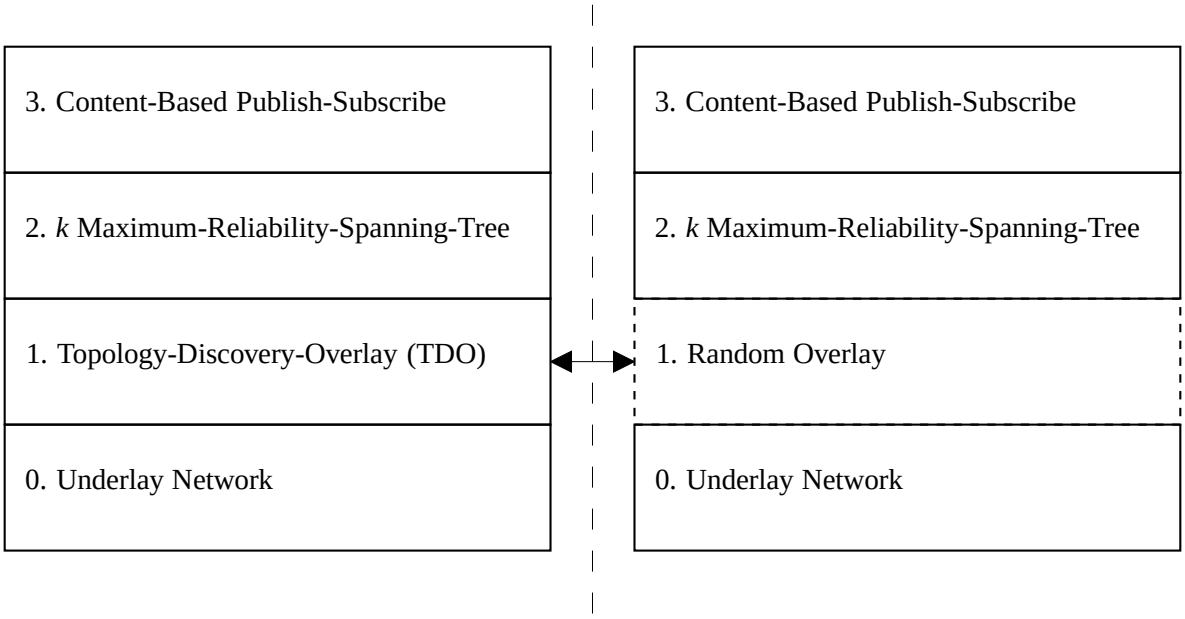


Figure 7.9: Protocol-Stacks of the two simulation types

For the first type the following set of simulation has been performed:

Random Simulations using the random approach described in Section 5.2 as TDO layer. The random approach builds its overlay based on traceroute measurements to 15 neighbours.

Random (*k* nearest neighbours) Simulations using the *k*-nearest-neighbours approach described in Section 5.2.3 as TDO layer. This layer selects the 15 nearest peers out of its $15 + 10 = 25$ random peers sample to perform traceroute measurements to.

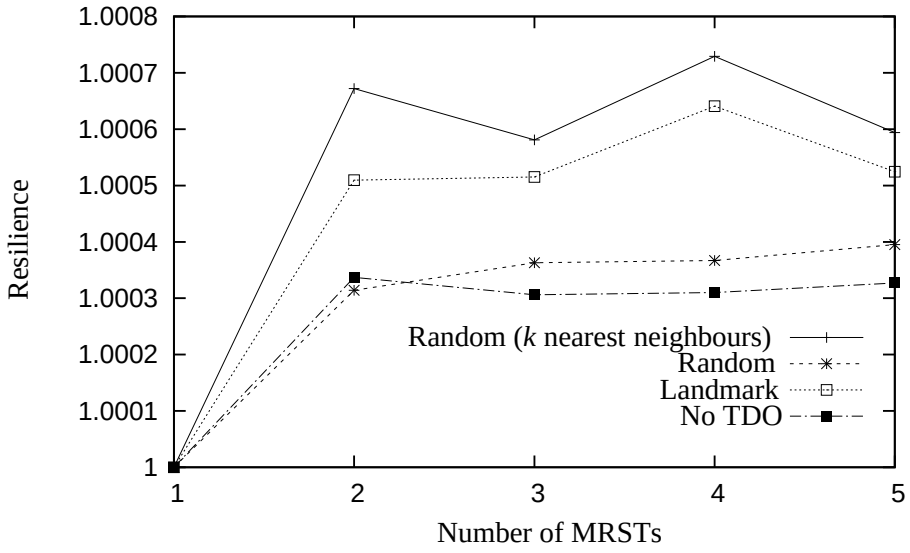
Landmark Simulations using the landmark-based approach described in Section 5.1 as TDO layer. All simulations have been performed with 15 landmarks.

Additionally, simulations have been conducted on top of a plain random overlay topology:

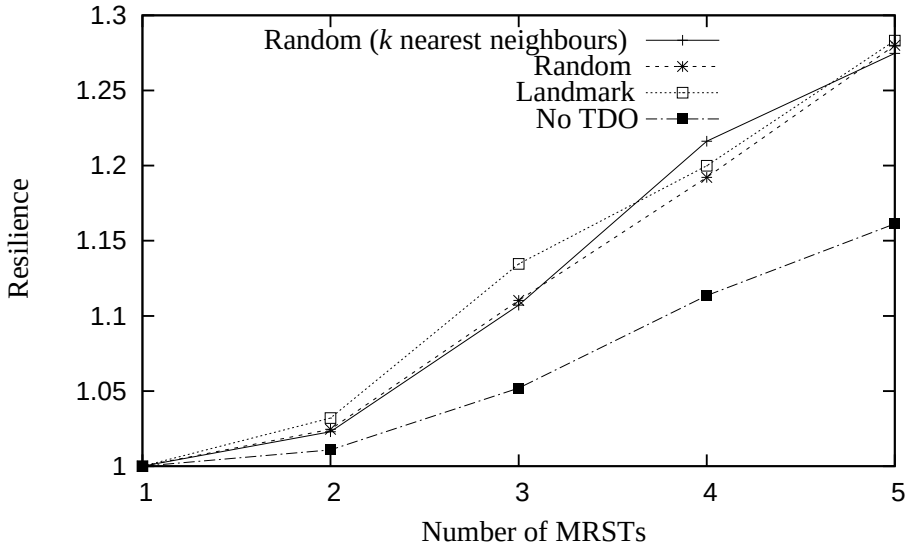
No TDO The TDO layer is exchanged by a random overlay where each peer is connected to 15 randomly selected other peers.

7.3.1 Underlay Link Resilience

The *Underlay Link Resilience* is a metric to describe the vulnerability of a system. It is a good metric to provide guarantees on the availability of a system.



(a) Transit-Stub topology



(b) BRITE topology

Figure 7.10: Underlay Link Resilience

Figure 7.10 shows the simulation results of the Underlay Link Resilience. On Transit-Stub the resilience is nearly equal for all approaches. The reason for this may be that the Transit-Stub topologies do not provide path sets with a higher resilience. The generated stub-domains are only connected through one underlay link to the transit-domain. This results in a resilience of at most one for all inter-domain overlay links. However, we can see that using a TDO results in a marginal higher resilience than without the TDO layer.

On BRITE topologies the situation is different. These topologies provide multiple underlay links for inter-domain communication. Hence, the average resilience is higher for simulations with these topologies. However, from the simulation results it is not clear which approach performs best. In this case, using the TDO makes a significant positive difference to simulations without the TDO.

BRITE topologies are more realistic in this case because ASs are usually directly connected to more than one other AS.

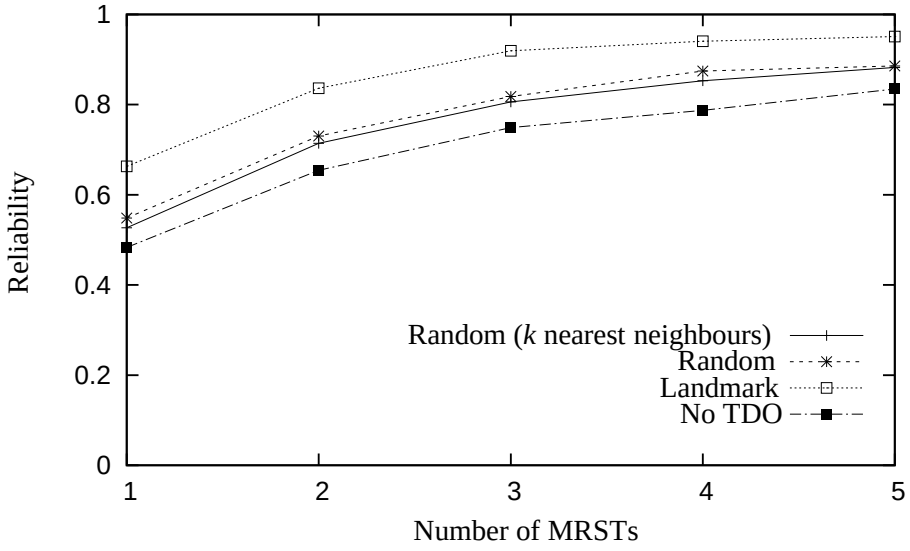
In case the underlay only provides a resilience of one it is simply impossible to reach a higher value for the Underlay Link Resilience. In such a case the metric is not a good basis to reveal reliability properties. The problem is that the metric does not take the reliability of underlay links into account. This is the reason why additional metrics will be defined in the next sections.

7.3.2 Reliability

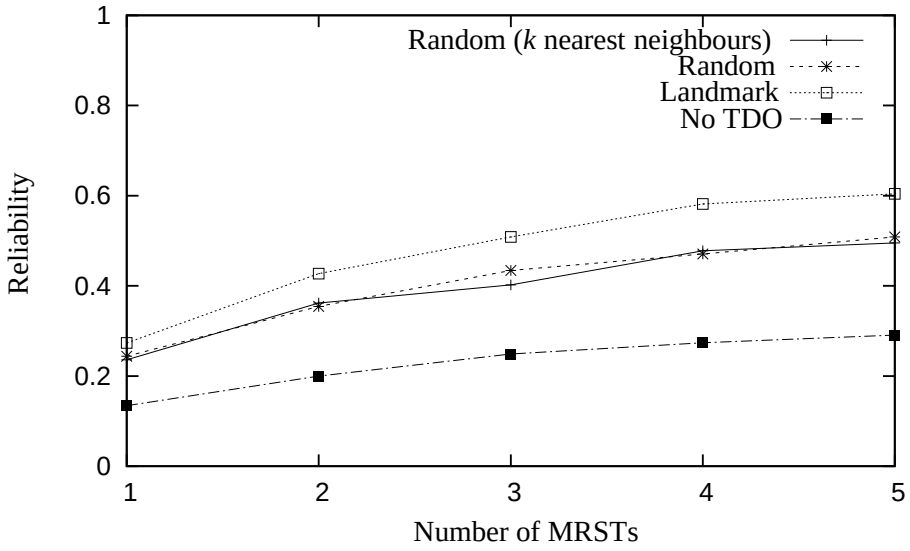
The metric *Reliability* measures the joint reliability of a multipath. If the path diversity is high, adding a new path to a multipath should increase total reliability of the multipath.

The simulation results for *Reliability* measurements are plotted in Figure 7.11. There is a great difference in reliability between Transit-Stub and BRITE topologies. The average reliability of paths in the BRITE topology is less than the one in Transit-Stub topologies. This is the case, because the AS in our BRITE topologies have much more nodes than a stub in the Transit-Stub topologies (82 nodes in BRITE and 10 nodes in Transit-Stub). This results in a larger network diameter (32 hops for BRITE, compared to 10 hops for Transit-Stub) and average path length for BRITE topologies which reduces the reliability. Furthermore, the average path reliability of a topology mostly depends on assigned reliabilities of individual links and nodes. However, the more interesting aspect of this simulation results are the relative differences among the approaches.

We can see that in both topologies **Landmark** performs best. This situation confirms the results in Section 7.2 where the landmark-based approach performed best. **Random** and **Random (*k* nearest neighbours)** both result in similar average reliabilities. As expected simulations without the TDO got worst results. Especially in BRITE topologies – even in the case of only one MRST – the reliability is improved by more than 100% (and more than 37% in Transit-Stub) when comparing **Landmark** to **No TDO**. The improvement of all approaches compared to **No TDO** in percentage can be viewed in Figure 7.12.

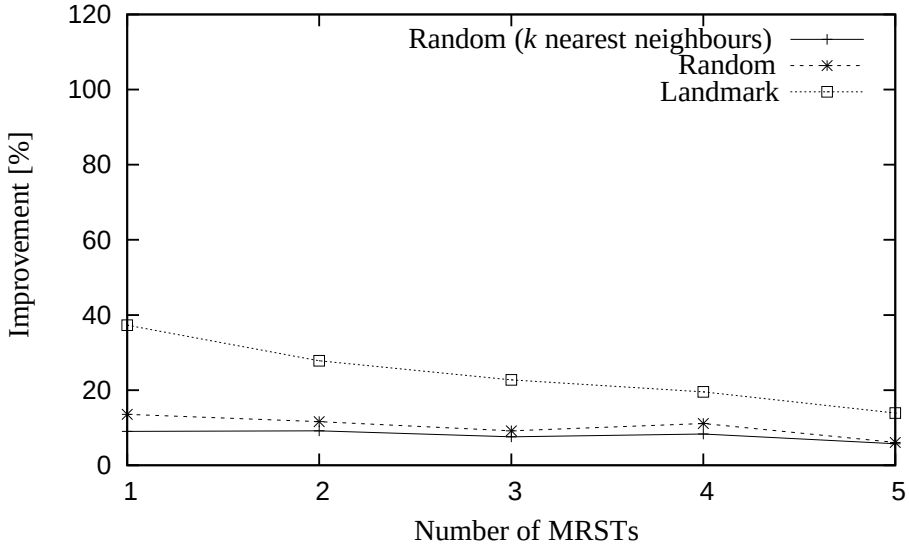


(a) Transit-Stub topology

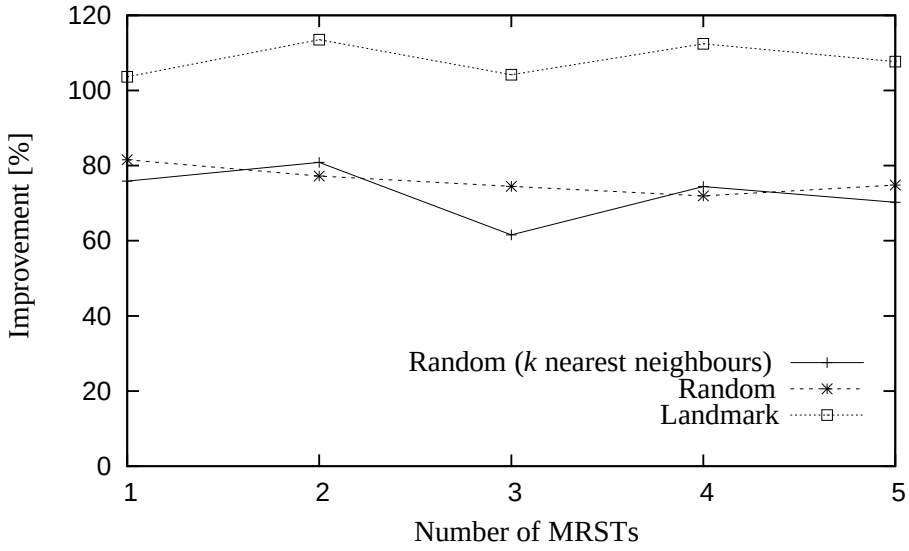


(b) BRITE topology

Figure 7.11: Joint path reliability



(a) Transit-Stub topology

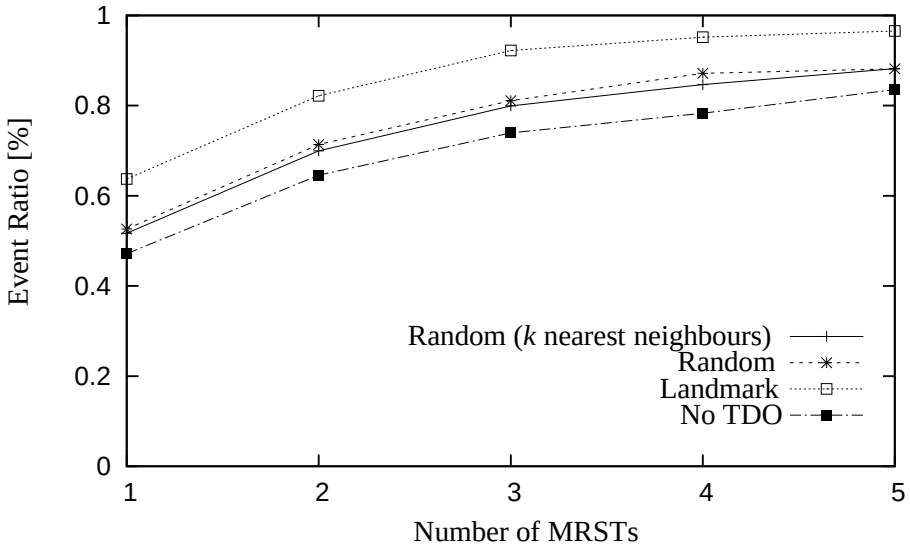


(b) BRITE topology

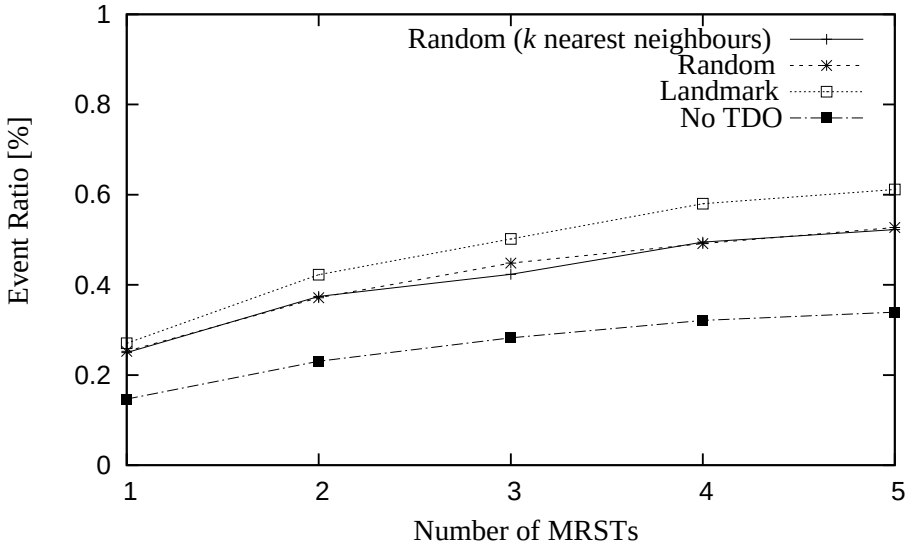
Figure 7.12: Reliability improvement compared to **No TDO**

7.3.3 Event Ratio

The main goal of the proposed system is to increase message delivery reliability. The *Event Ratio* measures the fraction of received events to the total number of expected receives. Because an event has potentially more than one subscriber with a matching subscription the number of expected receives is higher than the number of event publications. In our simulations each event had 16 peers (6,4%) with at least one matching subscription on average.

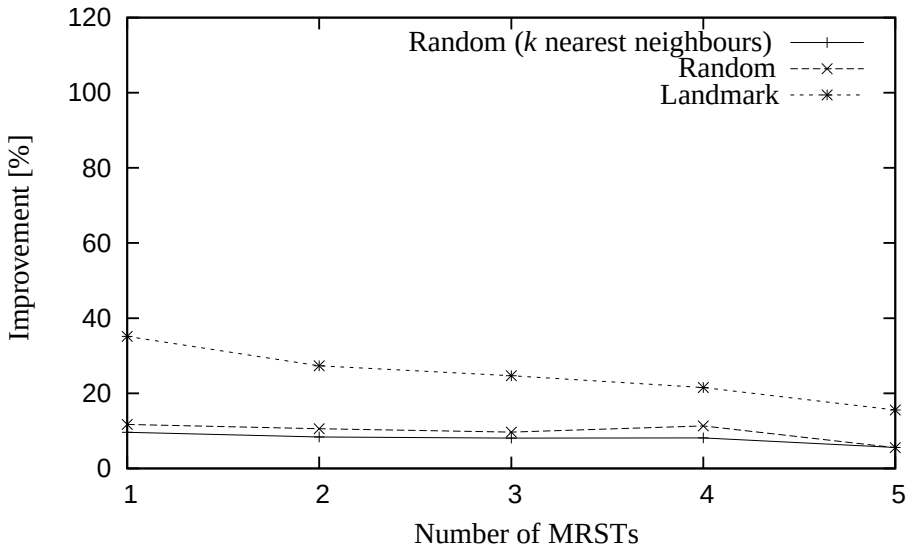


(a) Transit-Stub topology

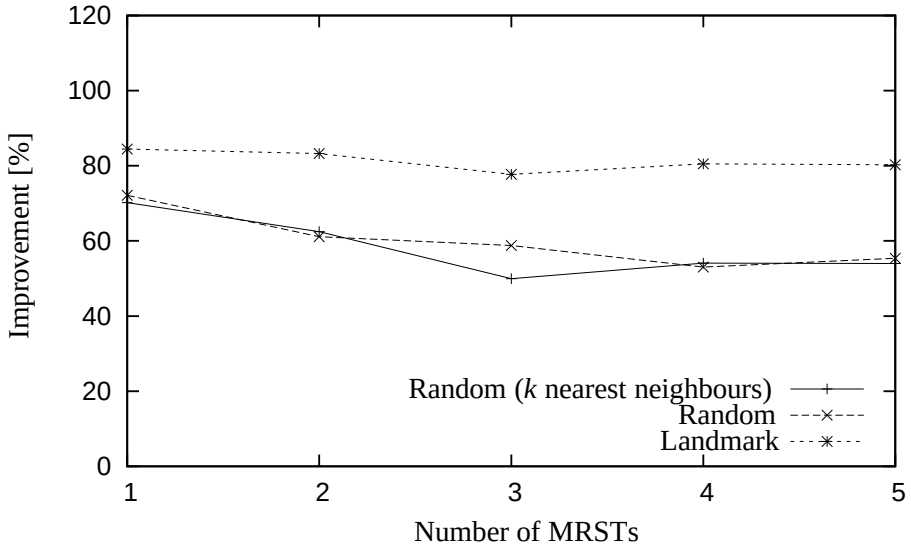


(b) BRITE topology

Figure 7.13: Event Ratio



(a) Transit-Stub topology



(b) BRITE topology

Figure 7.14: Improvement of the Event Ratio compared to **No TDO**

Figure 7.13 plots the simulation results for the Event Ratio. As expected the results reflect the results for path reliability (Section 7.3.2, Figure 7.11). Similar to the previous simulation results, **Landmark** performed best. **Random** and **Random (*k* nearest neighbours)** have similar results in BRITE but **Random** is slightly better in Transit-Stub. Figure 7.14 shows the improvement of the Event Ratio compared to **No TDO**. The **Landmark** approach improves the Event Ratio up to nearly 90% regardless of the number of MRSTs.

7.3.4 Summary

Evaluations show that using the TDO layer makes a significant difference compared to simulations without this layer. All proposed TDO approaches increased the average availability and reliability of paths. The landmark-based approach still seems to produce the best results, but the random approaches also produce considerable results. The *k*-MRST approach produces multiple overlay paths, however, the disjointness of paths is constraint by the underlay topology.

8 Conclusion

In this thesis an approach to improve reliability in a content-based P2P publish-subscribe system has been proposed. In order to increase reliability, several layers have been provided to reduce failure probabilities at the respective failure source.

To solve the problem of underlay failures, the idea of this thesis was to reduce underlay correlations within dissemination trees to a minimum. This has been done by constructing an underlay aware overlay topology which reflects the underlay topology. For this Topology-Discovery-Overlay (TDO) two different instances of a path-matching algorithm have been developed, which use overlap information within traceroute results to find a good location for an overlay peer within the overlay topology. The landmark-based approach discovers underlay paths from a small subset of the available peers to infer most of the underlay topology. The random approach discovers random underlay paths and compares them with each other to leverage overlap information. Evaluations have been made through simulations on generated Internet-like topologies where the developed instances of the algorithm have been compared. Simulation results show that the algorithms discover more than 80% of the underlay links and construct overlay topologies with properties similar to the underlay topology. Shortest overlay paths of overlays constructed by the landmark-based approach have a low stretch of less than 1.5 even in the case where only 2% of the peers are landmarks.

In order to tackle overlay failures, the constructed overlay is then used by a higher level layer to provide multiple overlay edge disjoint paths in the overlay between any two peers. Because underlay diversity of the corresponding paths of overlay links provided by the TDO have high diversity, correlations between these multiple paths are low. To establish multiple paths between any pair of peers, multiple edge disjoint Maximum-Reliability-Spanning-Trees (MRST) are built. To achieve this, a well-known distributed Minimum Spanning Tree algorithm has been adapted. The resulting algorithm is able to create k MRSTs in parallel in a distributed manner. Simulations have been conducted to evaluate the reliability of the established paths. To evaluate the benefits of the TDO, simulations have been performed with both, a plain random overlay topology and an overlay constructed by the TDO layer. Simulation-results show that the reliability of overlay paths is up to 110% higher when using the TDO, compared to simulations without the TDO.

On top of the constructed MRSTs, a simple publish-subscribe layer has been implemented. It uses each MRST separately for subscription flooding and event forwarding. Simulations show that the event delivery probability converges towards a maximum.

8.1 Future Work

The developed approach provides reliable dissemination of events. However, some of the assumptions may not be realistic. The system fully relies on the availability of traceroute techniques. Although traceroute is a good mechanism to obtain underlying paths in IP networks, it has some drawbacks. Due to routers not responding to traceroute packets (anonymous routers), discovered paths may be incomplete. A possible solution would be to integrate additional topology discovery methods like end-to-end measurements [NXTY10].

The k-MRST algorithm constructs cycle free graphs but its time complexity of $O(n \log n)$ is not optimal. The k-MRST layer could be modified to use a faster approach like [KP98], which has a time complexity of $O(\sqrt{n} \log^* n + d)$, where d is the diameter of the network.

Although results show that the landmark-based approach performs best, it has some drawbacks which affect scalability. Simulations of the random approaches, which overcome the scalability problems of the landmark-based approach, show that no clear decision can be made which of the approaches performs best. Therefore, additional simulations for the TDO on real Internet topologies should be made.

Bibliography

[DCKM04] F. Dabek, R. Cox, F. Kaashoek, R. Morris. Vivaldi: a decentralized network coordinate system. *ACM SIGCOMM Computer Communication Review*, 34(4):15, 2004. (Cited on page 27)

[DFC05] B. Donnet, T. Friedman, M. Crovella. Improved Algorithms for Network Topology Discovery. *Network*, pp. 149–162, 2005. (Cited on page 27)

[DRFC04] B. Donnet, P. Raoult, T. Friedman, M. Crovella. Efficient Algorithms for Large-Scale Topology Discovery. *ACM Sigmetrics*, p. 23, 2004. (Cited on page 27)

[DRFC06] B. Donnet, P. Raoult, T. Friedman, M. Crovella. Deployment of an algorithm for large-scale topology discovery. *IEEE Journal on Selected Areas in Communications*, vol(12):24no12pp2210–2220, 2006. (Cited on page 27)

[ECG09] C. Esposito, D. Cotroneo, A. Gokhale. Reliable publish/subscribe middleware for time-sensitive internet-scale applications. *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems - DEBS '09*, p. 1, 2009. doi:10.1145/1619258.1619280. (Cited on pages 14, 16 and 19)

[GHS83] R. G. Gallager, P. A. Humblet, P. M. Spira. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983. doi:10.1145/357195.357200. (Cited on pages 29, 38 and 53)

[GPS04] B. Garbinato, F. Pedone, R. Schmidt. An adaptive algorithm for efficient message diffusion in unreliable environments. In *Dependable Systems and Networks, 2004 International Conference on*, Dsn, pp. 507–516. IEEE, 2004. doi:10.1109/DSN.2004.1311920. (Cited on pages 13, 23 and 24)

[GT-00] Modeling Topology of Large Internetworks, 2000. URL http://www.cc.gatech.edu/projects/gtitm/. (Cited on page 63)

[JTC08] X. Jin, W. Tu, S.-H. G. Chan. Scalable and Efficient End-to-End Network Topology Inference. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):837–850, 2008. doi:10.1109/TPDS.2007.70771. (Cited on pages 26 and 28)

[KB07] M. Kumar S. D, U. Bellur. A Distributed Algorithm for Underlay Aware and Available Overlay Formation in Event Broker Networks for Publish/Subscribe Systems. *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*, pp. 69–69, 2007. doi:10.1109/ICDCSW.2007.9. (Cited on pages 14 and 20)

[KB08] M. Kumar S. D, U. Bellur. Availability models for underlay aware overlay networks. *Proceedings of the second international conference on Distributed event-based systems - DEBS '08*, p. 169, 2008. doi:10.1145/1385989.1386011. (Cited on page 20)

[KF02] M. Kwon, S. Fahmy. Topology-aware overlay networks for group communication. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video - NOSSDAV '02*, p. 127. ACM Press, New York, New York, USA, 2002. doi:10.1145/507686.507688. (Cited on pages 25, 26 and 38)

[KP98] S. Kutten, D. Peleg. Fast Distributed Construction of Smallk-Dominating Sets and Applications. *Journal of Algorithms*, 28(1):40–66, 1998. doi:10.1006/jagm.1998.0929. (Cited on page 84)

[MB07] S. P. Mahambre, U. Bellur. Reliable Routing of Event Notifications over P2P Overlay Routing Substrate in Event Based Middleware. *2007 IEEE International Parallel and Distributed Processing Symposium*, pp. 1–8, 2007. doi:10.1109/IPDPS.2007.370658. (Cited on pages 13, 21 and 23)

[MJ09] A. Montresor, M. Jelasity. PeerSim: A Scalable P2P Simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pp. 99–100. 2009. URL http://peersim.sourceforge.net/. (Cited on page 63)

[MLMB01] a. Medina, A. Lakhina, I. Matta, J. Byers. BRITE: an approach to universal topology generation. *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 346–353, 2001. doi:10.1109/MASCOT.2001.948886. (Cited on page 63)

[NXTY10] J. Ni, H. Xie, S. Tatikonda, Y. R. Yang. Efficient and Dynamic Routing Topology Inference From End-to-End Measurements. *IEEE/ACM Transactions on Networking*, 18(1):123–135, 2010. doi:10.1109/TNET.2009.2022538. (Cited on pages 29 and 84)

[NZ02] T. Ng, H. Zhang. Predicting Internet network distance with coordinates-based approaches. *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, 1:170–179, 2002. doi:10.1109/INFCOM.2002.1019258. (Cited on page 27)

[PQW03] V. Padmanabhan, L. Qiu, H. Wang. Server-based inference of Internet link lossiness. *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, 00(C):145–155, 2003. doi:10.1109/INFCOM.2003.1208667. (Cited on page 22)

[SGF10] A. Singla, P. Godfrey, K. Fall. Scalable routing on flat names. *Proceedings of the*, 2010. (Cited on page 40)

[She10] H. Shen. Content-Based Publish/Subscribe Systems. In X. Shen, H. Yu, J. Buford, M. Akon, editors, *Handbook of Peer-to-Peer Networking*, pp. 1333–1366. Springer US, Boston, MA, 2010. doi:10.1007/978-0-387-09751-0_49. (Cited on pages 11 and 12)

[TKK+11] M. A. Tariq, B. Koldehofe, G. G. Koch, I. Khan, K. Rothermel. Meeting subscriber-defined QoS constraints in publish/subscribe systems. *Concurrency and Computation: Practice and Experience*, pp. 1–15, 2011. (Cited on pages 13, 15 and 16)

[TM05] C. Tang, P. McKinley. Improving multipath reliability in topology-aware overlay networks. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pp. 82–88. IEEE, 2005. (Cited on page 22)

[TSN10] Y. Tian, H. Shen, K.-W. Ng. Improving Reliability for Application-Layer Multicast Overlays. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1103–1116, 2010. (Cited on page 13)

[YKPW04] a. Young, A. Krishnamurthy, L. Peterson, R. Wang. Overlay mesh construction using interleaved spanning trees. *Ieee Infocom 2004*, 1(C):396–407, 2004. doi:10.1109/INFCOM.2004.1354512. (Cited on page 53)

All links were last followed on August 29, 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.

(Tom Quist)