Autonomous Interconnection of Heterogeneous Networks

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurswissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Dipl.-Inform. Sebastian Mies

aus Konstanz

Tag der mündlichen Prüfung: 29. Juni 2012

Erster Gutachter: Prof. Dr. Martina Zitterbart

Zweiter Gutachter: Prof. Dr. Thomas Plagemann

Danksagung

Mein erster Dank für das Vertrauen und die Unterstützung gilt Frau Prof. Dr. Martina Zitterbart. Sie gab mir die Möglichkeit am Institut für Telematik zu forschen, an Konferenzen teilzunehmen, meine Ideen umzusetzen und schließlich zu promovieren. Herrn Prof. Dr. Thomas Plagemann danke ich für die Übernahme des Korreferats und die vielen konstruktiven Kommentare. Herzlich danken möchte ich Herrn Dr. Oliver Waldhorst, der durch wertvolle Kommentare und gemeinsame Arbeiten wesentlich zum Gelingen dieser Arbeit beigetragen hat.

Bei allen Kollegen am Institut für Telematik bedanke ich mich für das außergewöhnlich angenehme Arbeitsklima und für die vielen regelmäßigen Diskussionsrunden von denen ich stark profitiert habe. Bei meinen Teamkollegen Christian Hübsch und Christoph Mayer möchte ich mich hier besonders bedanken. Ein herzlicher Dank geht an Denise Dudek, Bernhard Heep und Christoph Werle für das intensive Korrekturlesen und die Verbesserungsvorschläge!

Diese Arbeit wäre nicht möglich gewesen ohne das Mitwirken vieler Studierender durch Studien- und Diplomarbeiten, aber auch als wissenschaftliche Hilfskräfte: vielen Dank euch allen! Ein besonderer Dank gilt Hans Wippel, der mit seiner Diplomarbeit einen der Grundsteine für diese Arbeit legte.

Viele Menschen haben mich auf meinem Lebensweg bis zur Abgabe dieser Arbeit unterstützt, wofür ich sehr dankbar bin. Dabei möchte ich mich insbesondere bei meiner Familie für die uneingeschränkte Unterstützung bedanken.

Meine langjährige Lebensgefährtin Julia Bohli stand mir immer zur Seite. Während der herausfordernden Schlussphase, in der die gemeinsame Zeit knapp wurde, motivierte sie mich besonders und half mir beim Korrekturlesen: Du bist die Beste! Vielen Dank!

Sebastian Mies
Karlsruhe, den 2. September 2012

Zusammenfassung

Peer-to-Peer (P2P) Anwendungen im Internet erfreuen sich zunehmender Beliebtheit. Sei es zur Telefonie mit Skype, zum Dateiaustausch mittels BitTorrent oder für virtuelle private Netze mittels P2P-VPN. Auch Forschungsprojekte wie das Spontane Virtuelle Netze Projekt (SpoVNet) nutzen die Technologien von P2P-Anwendungen, um einen evolutionären Ansatz für zukünftige Dienste im Internet bereitzustellen. Ein Grund für die Popularität von P2P-Anwendungen sind deren Eigenschaften: sie arbeiten autonom, funktionieren ohne aufwändige Konfiguration, skalieren mit der Anzahl der Nutzer und benötigen keine Änderungen in der Infrastruktur. P2P-Anwendungen verlassen sich allerdings darauf, dass die Netze, in denen sie ausgebracht werden, nahtlose Konnektivität bereitstellen. Das bedeutet, dass jedes Gerät auf dem die P2P-Anwendung gestartet wurde, potentiell mit allen anderen Geräten kommunizieren kann.

Durch die wachsende Heterogenität ist dies aber nicht mehr uneingeschänkt möglich. Wurde im Internet beispielsweise zuvor nur mit dem IPv4 Protokoll kommuniziert, sind es heute durch die Einführung von IPv6 zwei Protokolle. Ein Gerät innerhalb eines reinen IPv6 Netzes kann so nicht mit einem Gerät aus einem reinen IPv4 Netz kommunizieren. Weiterhin wurden an das Internet viele private Netze, wie beispielsweise Heimnetze, angeschlossen welche für weitere Heterogenität sorgen. Technologien wie Bluetooth oder ZigBee verstärken die Heterogenität noch weiter. Diese Umstände machen eine Nutzung gängiger P2P-Anwendungen schwierig oder gar unmöglich.

Diese Arbeit stellt eine Middleware vor, welche nahtlose Konnektivität für P2P-Anwendungen in heterogenen Netzen bereitstellt. Dazu werden Geräte verwendet, die sich in mehreren Netzen gleichzeitig befinden; die Middleware verletzt dabei keine der zuvor genannten Eigenschaften von P2P-Anwendungen.
Die in der Arbeit vorgestellte Middleware besteht aus zwei Teilen:

- Erkennung der Teilnetze mit bestehender nahtloser Konnektivität, und
- Kopplung der heterogenen Netze

Erkennung der Teilnetze mit Konnektivität

Um nahtlose Konnektivität effizient bereitstellen zu können, muss zunächst erkannt werden, in welchen Teilnetzen bereits nahtlose Konnektiviät besteht und wo nicht. Dabei spannt eine Gruppe von Nutzern welche untereinander Nachrichten austauschen können eine sogenannte Konnektivitätsdomäne auf. Nutzer, welche sich in mehreren solcher Konnektivitätsdomänen befinden, können Nachrichten zwischen den Gruppen austauschen und werden Relays genannt. Die Konnektivitätsdomänen spiegeln so die in heterogenen Netzen bereits bestehende Konnektivität wider. Zur Erkennung der Konnektivität müssen diese Konnektivitätsdomänen ermittelt werden.

Die Tatsache, dass durch die Eigenschaften der P2P-Anwendungen kein explizites Wissen über die heterogenen Netze und deren Konnektivität vorausgesetzt werden kann, stellt eine Herausforderung dar. Die Arbeit zeigt, dass dieses Problem dadurch ähnlich zu dem Problem des Aufzählens aller Cliquen in einem Graph ist. Es ist bekannt, dass dieses Problem schwer lösbar ist. Aus diesem Grund wird in der Arbeit zunächst theoretisch ermittelt unter welchen Voraussetzungen und Annahmen eine Erkennung im Rahmen einer P2P-Anwendung möglich ist.

Auf Basis dieser Analyse wird das Connectivity Measurement Protokoll (CMP) vorgestellt. CMP prüft zunächst, ob sich das Gerät eines Nutzers in mehreren Konnektivitätsdomänen befindet. Danach einigt sich CMP auf den Geräten der aller Nutzer, welche sich in derselben Konnektivitätsdomäne befinden, auf einen eindeutigen Identifizierer. Die Identifizierer dienen dann der Adressierung der unterschiedlichen Konnektivitätsdomänen.

Es konnte gezeigt werden, dass CMP die Konnektivität selbst unter widrigen Bedingungen in weniger als 20 Sekunden erkennt, mit der Anzahl der Nutzer skaliert und wenig Bandbreite benötigt. Im besten Fall kann CMP die Konnektivität in wenigen Sekunden erkennen. Weiterhin passt CMP die Identifizierer automatisch an Veränderungen der Konnektivität an. Die Eigenschaften der P2P Anwendungen werden durch CMP nicht beeinträchtigt.

Kopplung der heterogenen Netze

Auf Basis der zuvor ermittelten Identifizierer stellt das Connectivity Domain Interconnection Protocol (CDIP) eine nahtlose Konnektivität zwischen Geräten in unterschiedlichen Konnektivitätsdomänen her. CDIP verbindet dazu zunächst die Relays zu einem logischen Netz, welches alle erkannten Konnektivitätsdomänen überspannt. Dann nutzt es ein Routing-Protokoll in diesem logischen Netz um Nachrichten zwischen Konnektivitätsdomänen anhand der jeweiligen Identifizierer weiterzuleiten. In der Arbeit wird dazu ein geeignetes Routingprotokoll, Tailored Routing (TRout), vorgestellt. Dieses beeinträchtigt keine der Eigenschaften von P2P-Anwendungen. Es skaliert in Bezug auf den Speicherplatzbedarf, die Anzahl der Nutzer und die Veränderungen im Netz.

Die entwickelten Protokolle werden theoretisch und simulativ auf ihre Tauglichkeit im Rahmen von P2P-Anwendungen überprüft. In den betrachteten Szenarien konnte nachgewiesen werden, dass die vorgestellten Protokolle den Anforderungen von P2P-Anwendungen gerecht werden.

Durch die in dieser Arbeit vorgestellte Middleware können existierende P2P-Anwendungen mit wenigen Änderungen auf heterogenen Netzen ausgebracht werden. Es ist davon auszugehen, dass die Netzlandschaft in Zukunft, bespielsweise durch Netzvirtualisierung, immer heterogener wird. Diese Arbeit trägt einen wichtigen Ansatz zur Beherrschbarkeit dieser Netzlandschaft bei.

Contents

Figures

Tables

Chapter 1

Introduction

The Internet has a significant and ongoing impact on society. Emails, the World-Wide-Web and many other services are indispensable for daily life and economics. One of the reasons for the Internet's success is the seamless connectivity the Internet provides, i. e., the potential of a user to reach any other user with Internet access. A category of applications that rely on seamless connectivity are peer-to-peer (P2P) applications. Those applications have recently become popular for file-sharing using BitTorrent [8], telephony using Skype [60], video streaming using StreamSwarm [59], and to provide virtual private networks using P2P-VPN [74]. Furthermore, future Internet initiatives provide P2P frameworks to support new services for today's infrastructure [44]. P2P applications run on personal devices and use the existing network to provide additional services.
Most P2P applications have the following features in common:

- they just work, i. e., a P2P application uses self-organization techniques to reduce the need for manual configuration.
- they scale with an increasing number of users.
- they operate autonomously and do not require any changes in the infrastructure, i. e., no changes in the network are required. Hence, most P2P applications do not have a single point of failure.

A drawback for P2P applications is that networks have become increasingly heterogeneous. The introduction of IPv6 in the Internet, middle boxes in DSL home routers that use network address translation (NAT) [25, 43], Bluetooth [82], and virtual private networks (VPNs) are some examples. Additionally, future Internet design projects, e. g., the 4WARD [18] or the G-Lab project, introduce "virtual networks" that allow the deployment of specialized networks, e. g., for IPTV, or content delivery networks (CDNs). This results in even more heterogeneity.

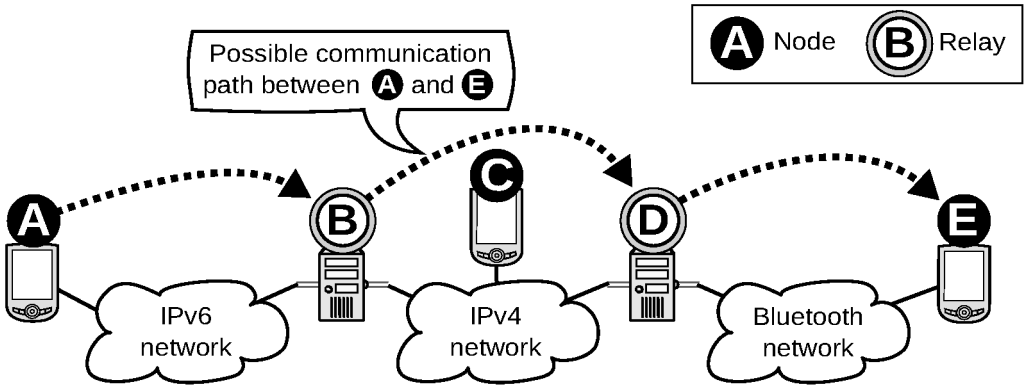


Figure 1.1 – An initial example: nodes with access to heterogeneous networks.

In most cases only dedicated gateways can provide seamless connectivity between such heterogeneous networks since these networks are incompatible to each other. Installing and configuring such gateways is a time-consuming, error-prone task. This makes the use of existing P2P applications in heterogeneous networks tricky or even impossible. Furthermore, users running P2P applications on their devices, e. g., on their home computers, notebooks, or smart phones, usually do not want to install and configure gateways. They often are not aware which networks they are using. Instead, they expect that the P2P application just works even when the communication path needs to cross several heterogeneous networks.

Figure 1.1 illustrates active networked devices, so-called *nodes*, with access to heterogeneous networks. The figure comprises an IPv4, an IPv6, and an Bluetooth network. The nodes have access to at least one of those networks each. Node A has access to the IPv6 network, C has access to the IPv4 network, and node E has access to the Bluetooth network. Nodes B and D have access to more than one network, i. e., they are multi-homed. Node B has access to the IPv4 and IPv6 network, and node D has access to the IPv4 and Bluetooth network. Those nodes are called *relays* since they are able to forward messages between the IPv4/IPv6, and IPv6/Bluetooth networks, respectively. The two relays have the potential to enable communication between all nodes in this setup; as an example, the figure shows a possible communication path between nodes A and E via relays B and D.

This thesis focuses on the deployment of P2P applications in heterogeneous networks. For a better understanding of the barriers of P2P applications in heterogeneous networks, it is necessary to know that they use overlay networks (or just *overlays*) to provide their services. Overlays are logical networks built on top of another network, i. e., briefly the *underlay*. In the overlay, nodes are connected by links, which are established using the transport protocols of the underlay. Many P2P applications construct overlays that have a certain structure. For example, a P2P application can form an overlay with ring topology, comprising nodes ordered by their telephone numbers. This ring could then be used to find a user in order to make a phone call. The service-oriented structure of the overlays of P2P appli-

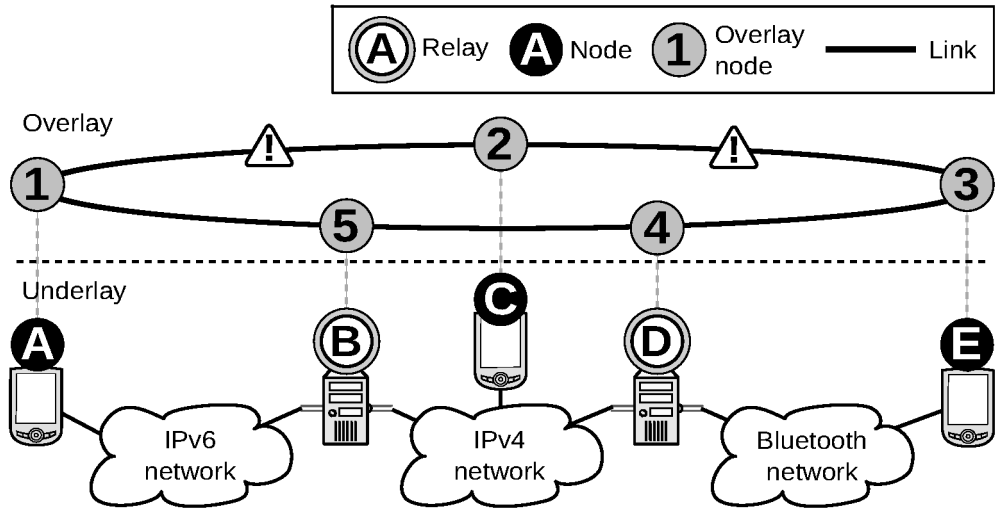


Figure 1.2 – Example of an overlay of a P2P application deployed in heterogeneous networks.

cations mandates that underlay has to provide seamless connectivity between all nodes.

1.1 Problem Statement

A P2P application cannot built a service-oriented overlay because of the lack of seamless connectivity in heterogeneous networks. Figure 1.2 shows an overlay of a P2P application deployed in heterogeneous networks. In the illustrated scenario, the P2P application attempts to build the an overlay with a ring topology. However, due to the heterogeneity, the P2P application cannot establish links between the overlay nodes (1), (2) and (2), (3) to complete the ring. The P2P application cannot operate correctly.

One drawback of P2P applications in heterogeneous networks is the lack of detailed information about the sets of nodes that have seamless connectivity. Furthermore, they do not know which nodes are relays that may help to overcome the lack of seamless connectivity. Additionally, even when the P2P application knows about the provided connectivity, the challenge of working around the lack of seamless connectivity remains. Possible measures are either changing the overlay structure or re-establishing seamless connectivity.

Another difficulty is, that nodes may change their network access anytime—either explicitly, by connecting to a VPN, or implicitly, when migrating from one private home WiFi to another one. Each of those events results in a change of connectivity that needs to be taken into account.

1.2 Claims

The main contribution of the Connectivity Domain middleware (CD middleware) is seamless connectivity for P2P applications deployed in heterogeneous networks. The claims of this thesis are as follows:

Claim 1: *The CD middleware does not weaken the typical strengths of P2P applications.*
P2P applications run decentralized in an unstable environment where nodes may join, fail, or leave at any point in time. P2P applications handle those events and adapt automatically. Furthermore, they are scalable and do not need infrastructure support. This thesis claims that the CD middleware does not weaken any of those properties.

Claim 2: *The CD middleware can detect the connectivity provided by heterogeneous networks.*
Connectivity detection enables the efficient interconnection of heterogeneous networks for P2P applications. Furthermore, the CD middleware allows to tailor P2P protocols for heterogeneous networks. This thesis claims that connectivity detection is possible.

Claim 3: *The CD middleware can provide seamless connectivity for P2P applications and requires only small modifications on existing P2P applications.*
Knowing the existing connectivity it is possible to provide seamless connectivity by applying a routing protocol that uses the detected connectivity efficiently. This thesis claims, that the CD middleware provides seamless connectivity for P2P applications. Existing P2P applications only need to modify the address format.

1.3 Contributions

The main contributions of this thesis are:

Autonomous Detection of Connectivity: A major contribution of this thesis is an in-depth theoretical discussion of the problem and solution space of connectivity detection. It provides insights under which conditions the connectivity detection is possible in the scenario of P2P applications. These theoretical discussions are validated using additional simulations where applicable. Based on the theoretical findings the thesis introduces a protocol, the Connectivity Measurement Protocol (CMP), that detects connectivity in heterogeneous networks. Simulations of CMP are used to confirm the theoretical findings.

Interconnection of Heterogeneous Networks: Based on the detected connectivity this thesis presents a protocol that interconnects heterogeneous networks for P2P applications. This can be boiled down to a typical routing problem. This thesis discusses popular routing protocols available in related work and introduces a protocol for interconnecting heterogeneous networks, called Tailored Routing (TRout).

Moreover, the solutions of the sub-problems in this thesis have a positive impact on applications in other fields. Three aspects are relevant to other fields of applications as well:

Constraints of Relay Detection: This thesis provides theoretical insight on the feasibility of detecting relays in scalable decentralized systems. These findings can be transferred to other applications as well, e. g., the Unmanaged Internet Architecture (UIA) [34].

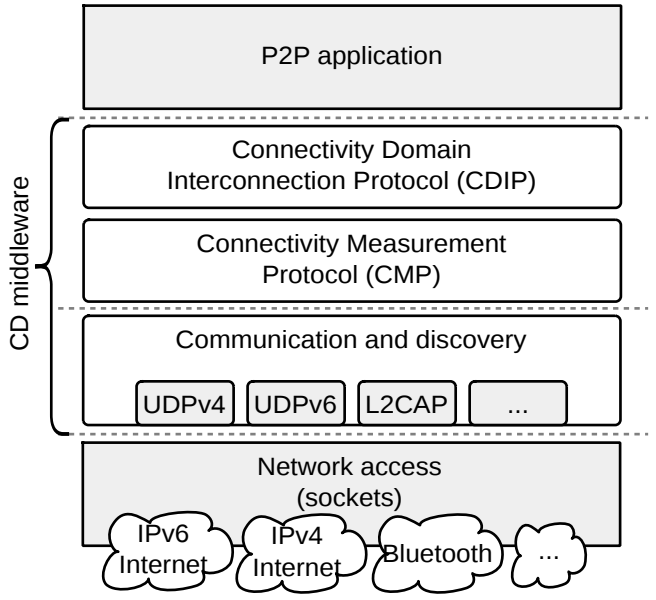


Figure 1.3 – Overview of the P2P application using the connectivity domain (CD) middleware to support heterogeneous networks

Random Number Agreement: This thesis presents an in-depth analysis of random number agreement algorithms. Random number agreements have entanglements with leader election and consensus algorithms which are essential for many decentralized systems.

Scalable Routing: The routing used for interconnecting the heterogeneous networks may also be used to route messages in ad-hoc networks or as a starting-point for further investigations on scalable routing; especially, when anycast is required.

1.4 Approach

As aforementioned, this thesis contributes a middleware that provides seamless connectivity to P2P applications. Figure 1.3 gives an architectural overview of the CD middleware. The bottom layer is the communication and discovery layer that allows finding other nodes that use the CD middleware as well. This layer provides a unified way of exchanging messages with nodes in the same network similar to the Berkeley socket interface [85] provided by operating systems. When considering the example depicted in Figure 1.2, this means that node (A) can discover and exchange messages with node (B). Node (B) can discover and exchange messages with nodes (A), (C), and (D), and so forth.

The communication and discovery layer does not provide detailed information about the connectivity between devices. The Connectivity Measurement Protocol (CMP) above the communication and discovery layer takes care of this problem. CMP detects the sets of nodes, called Connectivity Domains (CDs), that have seamless connectivity. For this purpose, CMP induces agreements on one Connec-

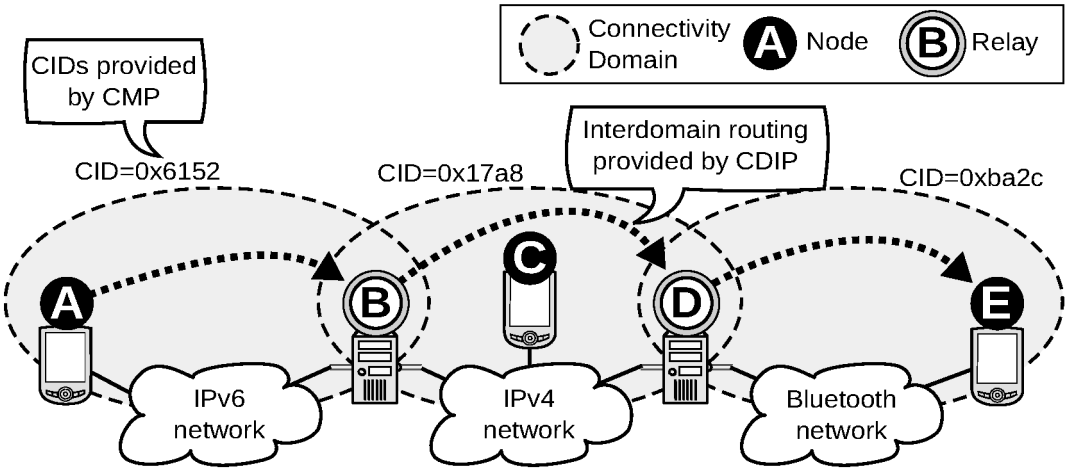


Figure 1.4 – An example of CMP's CID agreement among nodes in the same connectivity domain and a message routed across connectivity domains by CDIP.

tivity Identifier (CID) for each connectivity domain to differentiate them. CMP, furthermore, adapts the CIDs when connectivity changes.

It turns out, that the problem of connectivity detection is similar to the problem of enumerating all maximal cliques in a graph—which is known to be a hard problem. Thus, the theoretical constraints of connectivity detection are studied carefully. The theoretical findings, in conjunction with additional simulations of CMP, show that the CIDs can be determined quickly with low overhead. After CMP has detected connectivity domains, the layers above CMP learn about the CIDs of the connectivity domains including the node.

The layer above CMP, i. e., the Connectivity Domain Interconnection Protocol (CDIP) handles the routing of messages across connectivity domains, and, thus, between heterogeneous networks. Hence, CDIP is responsible for providing seamless connectivity to the P2P application.

CDIP builds an unstructured relay overlay containing all relays running the CD middleware and uses a routing protocol called Tailored Routing (TRout) which is based on the well-known Virtual Ring Routing (VRR) [13]. TRout is designed to run on the relays in the unstructured relay overlay and to route messages to a relay in the respective connectivity domain. For communication across connectivity domains the P2P applications use CDIP-addresses. A CDIP-address comprises the connectivity identifier $\mathsf{CID}(x)$ and the underlay address $addr(x)$ of some node x:

$$\langle \mathsf{CID}(x), addr(x) \rangle.$$

The underlay address denotes the address used for communication in the node's connectivity domain, e. g., a tuple of IPv4 address and UDP-port. The CID is used to determine whether the message needs to be routed across several connectivity domains by the relays. The underlay address of the destination node is used to deliver the message within the destination connectivity domain. From the P2P

application's point of view the communication with CDIP-addresses represents seamless connectivity.

Figure 1.4 illustrates the connectivity domains with the respective CIDs that were detected by CMP. The CIDs in the figure are only 16-bit wide to ensure readability. Node A agreed on a CID of *0x6152*, node C on a CID of *0x17a8*, and node E on a CID of *0xba2c*. The relays B and D know the CIDs of the two connectivity domains they are in. The figure also shows a message CDIP would route from node A to node E using the relays B and D.

1.5 Organisation

This thesis is organized as follows: Chapter 2 provides the fundamentals and notation used in this thesis. Chapter 3 describes the development of an underlay abstraction that resulted in the *ariba* middleware and was a motivation for the effort of this thesis. Chapters 4 and 5 introduces the main elements of the CD middleware, i. e., the CMP and CDIP protocol. Finally, Chapter 6 summarizes the results of this thesis, reviews the claims, and discusses further work.

Chapter 2

Fundamentals and Notations

This chapter introduces fundamentals and notations used in this thesis. It starts with the definition of graphs used for modeling networks in Section 2.1 and natural names for the complexities. Section 2.2 gives an introduction of P2P applications and properties. Finally, Section 2.3 explains basic principles and protocols for routing messages in a network.

2.1 Graphs and Complexities

For modeling network graphs, this thesis uses the notation and definitions derived from [70]. Since the graphs used in this thesis mostly reflect networks, a vertex is equivalent to a node, and an edge is equivalent to a link in an (overlay) network. Furthermore, if not noted differently, all sets in this thesis are *finite*. For convenience, Table 2.1 summarizes the notation of graphs. Directed and undirected graphs are defined as follows:

Definition 2.1 (Directed Graph) – A *directed graph*, or *digraph*, $G := (V, E)$ consists of a set of vertexes (or *nodes*) $V(G)$ and a set of edges (or *links*) $E(G) \subseteq V \times V$. A directed edge $e := (u, v) \in E(G)$ with $u, v \in V(G)$ is a pair of nodes u and v denoting a directed edge from vertex u to vertex v. Let $v \in V(G)$ be a vertex of graph G, then $\Gamma^-(v; G) := \{x | (x, y) \in E(G), y \stackrel{!}{=} v\}$ denotes the set of *in-neighbors*, i. e., vertexes with edges to vertex v. Furthermore, $\deg^-(v; G) := |\Gamma^-(v; G)|$ denotes the *indegree*, i. e., number of in-neighbors, of vertex v. Respectively, $\Gamma^+(v; G)$ denotes the set of *out-neighbors*, i. e., vertexes with edges from vertex v, and $\deg^+(v; G) := |\Gamma^+(v; G)|$ the *outdegree*.

Definition 2.2 (Undirected Graph) – An *undirected graph* (or *graph*) $G := (V, E)$ comprises a set of vertexes/nodes V and a set of edges $E \subseteq \{\{u, v\} | u, v \in V\}$. $V(G)$ denotes the vertex set and $E(G)$ the edge set of G. An edge $e := \{u, v\} \in E(G)$ is a set of two vertexes u and v. Let $v \in V(G)$ be a vertex of graph G, then $\Gamma(v; G) \subseteq$

Notation	Description
$N \in \mathbb{N}$	number of vertexes/nodes in a graph/network.
$M \in \mathbb{N}$	number of edges/links in a graph/network.
$G := (V, E)$	a graph with a set of vertexes/nodes V and edges/links E.
$E(G)$	set of edges/links of graph G.
$V(G)$	set of vertexes/nodes of graph G.
$\Gamma^{-/+}(v; G)$	set of in/out-neighbors of vertex v in digraph G.
$\Gamma(v; G)$	set of neighbors of vertex v in graph G.
$\deg^{-/+}(v, G)$	in/outdegree of vertex v in digraph G.
$\deg(v, G)$	degree of vertex v in graph G.
$p(u, v; G)$	a sequence of edges connecting vertexes u and v.
$\omega(e; G) \in \mathbb{R}$	weight of edge e.
$dist(u, v; G) \in \mathbb{R}$	length of the shortest path connecting vertexes u and v.
$diam(G) \in \mathbb{R}$	diameter of graph G.
$C(v; G) \in \mathbb{R}$	local clustering coefficient of a node v in graph G.
$C(G) \in \mathbb{R}$	global clustering coefficient of graph G.

Table 2.1 – Notation for describing graphs.

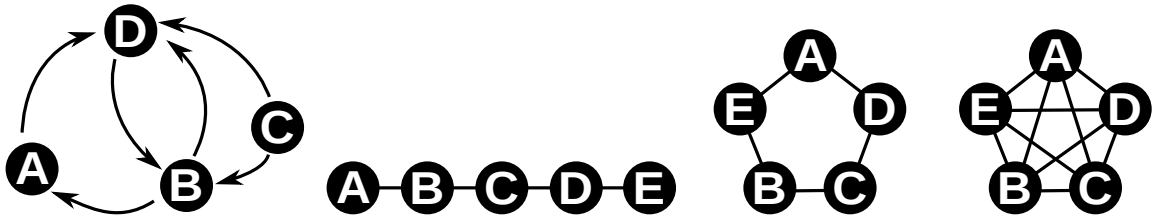
$V(G)$ denotes the set of *neighbors* of vertex v. Furthermore, $\deg(v; G) := |\Gamma(v; G)|$ denotes the *degree*, i. e., the number of neighbors, of vertex v.

If not defined otherwise, $N = |V(G)|$ and $M = |E(G)|$ denote the cardinality of the set of vertexes $V(G)$ and set of edges $E(G)$ of graph G respectively. Basic graph types are, e. g., path graphs, cycle graphs, and complete graphs. Figure 2.1 illustrates some examples. A path graph is a graph comprising vertexes connected to a straight linem and a cycle graph is a graph comprising vertexes connected to form a ring. In a complete graph each vertex is connected to any other vertex in the graph. A graph may contain several sub-graphs which are defined as follows:

Definition 2.3 (Cycle, Clique, and Maximal Clique Sub-Graphs) – A cycle in a graph is a sub-graph, i. e., a subset $G' \subset G$ of a graph G that is a cycle graph. A clique is a complete sub-graph, i. e., a subset $G' \subset G$ where all vertexes are connected to each other. A maximal clique is a clique that cannot be extended by adding an additional vertex from G.

It is important to know paths in a graph for modeling message flows/routes in a network. For this reason, the following definitions describe paths in graphs and specify the distance between vertexes. Furthermore, the diameter of a graph is defined.

Definition 2.4 (Path, Distance, and Diameter) – Let G be a (di-)graph and $u, v \in V(G)$ two vertexes in $V(G)$. Then $p(u, v; G) := (e_1, \ldots, e_n)$ denotes a path in G, i. e., a sequence of edges $e_1, \ldots, e_n \in E(G)$ that connects vertexes u and v. If the edges of a graph are weighted by $\omega(e; G) \in \mathbb{R}, e \in E(G)$, $dist(u, v; G) := \sum_{e \in p(u,v;G)} \omega(e; G)$ denotes the *distance* as sum of all weights on the shortest path connecting vertexes u and v. If the graph is unweighted, $\omega(e; G) := 1$ is assumed



(a) Directed graph (b) Path graph (c) Cycle graph (d) Complete graph

Figure 2.1 – Examples of graphs: a directed, a undirected, a path, a cycle, and a complete graph.

for all edges. The diameter of a graph G is defined as the maximum distance between two nodes, i. e., $diam(G) := \max_{x,y \in V(G)} dist(x, y; G)$

The clustering coefficient [96] is an important metric in graphs. It describes the connectedness of the vertexes in a graph, i. e., how close the graph is to a complete graph:

Definition 2.5 (Clustering Coefficient) – The local clustering coefficient of a vertex $v \in V(G)$ of a graph G denotes how close the neighbors of v, $\Gamma(v; G)$, are to a complete graph (or clique). More precisely, a clustering coefficient of 1.0 denotes that the neighbors of v form a clique, while a value of 0.0 means that the neighbors of v are not connected at all. The clustering coefficient of vertex $v \in V(G)$ in graph G is the quotient of the number of edges between v's neighbors and the number of edges that could possibly be between them. Hence, the local clustering coefficient is calculated as follows:

$$C(v; G) := \frac{2 \cdot |\{\{x, y\} \in E(G) : x, y \in \Gamma(v; G)\}|}{\deg(v; G) \cdot (\deg(v; G) - 1)}$$

The global clustering coefficient is the average of all local clustering coefficients:

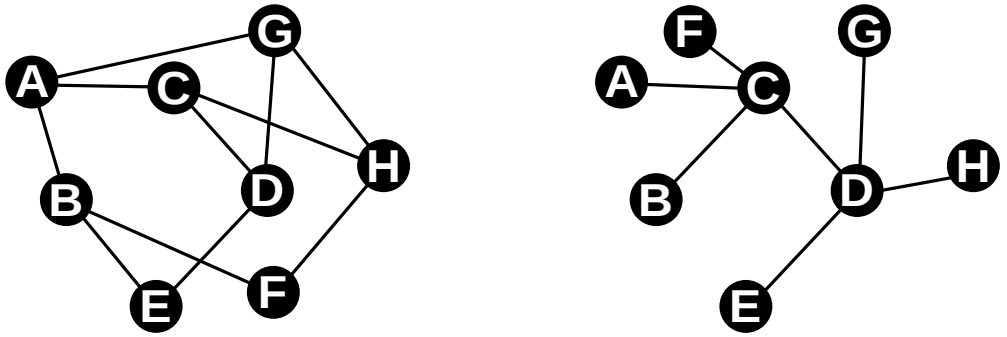
$$C(G) := \frac{1}{|V(G)|} \cdot \sum_{v \in V(G)} C(v; G)$$

2.1.1 Random Graphs

Random graphs are generated using a random graph model. These graphs have a certain structure, e. g., certain clustering coefficient, degree distribution, and diameter. This section introduces two popular models of random graphs: the Erdős–Rényi model and the Barabási-Albert (BA) model.

Erdős–Rényi Model

The Erdős–Rényi model describes two variants of generating random graphs. One variant connects N vertexes with M randomly chosen edges. The other variant connects two vertexes with an edge with probability p. The notation of an Erdős–Rényi graph is either $G(N, M)$ or $G(N, p)$. Random graphs following the Erdős–Rényi model tend to have a clustering coefficient of $C(G(N, p)) \approx p$.



(a) Random graph following the Erdős–Rényi model with $N = 8, M = 11$

(b) Random graph following the Barabási-Albert model with $N = 8, k = 1$

Figure 2.2 – Example of Erdős–Rényi (GNM) and Barabási-Albert (BA) random graph models.

Barabási-Albert (BA) Model

The Barabási-Albert (BA) model generates graphs with power-law distributed degree, i. e., they consist of many vertexes with a small degree and only a few vertexes with a high degree. To achieve this, the BA model uses a preferential attachment strategy. It begins with an initial graph comprising at least two vertexes, then, new vertexes are added to the graph. Each new vertex is connected to k other vertexes with new edges. The vertexes are chosen in favour of their degree, i. e., a vertex v is chosen using probability

$$p(v) := \frac{\deg(v)}{\sum_{x \in V(G)} \deg(x)}.$$

Therefore, it is likely that a vertex with a high degree will become connected to even more vertexes. This results in the power-law distributed vertex degree.

Figure 2.2 shows a exemplary graph generated with the Erdős–Rényi and Barabási-Albert models. While the Erdős–Rényi-based graph does not show any particular structure, the Barabási-Albert-based graph shows an immanent power-law structure.

2.1.2 Complexities

To express complexities of algorithms or protocols, this thesis uses the following natural terms instead of a mathematical big/small O notation where N denotes the dependency of the complexity. The order of the complexities is from *constant*, i. e., low complexity, to *exponential*, i. e., high complexity:

Constant: $O(1)$. Hence, the complexity does *not* grow with N.
Logarithmic: $O(\log N)$
Poly-logarithmic: $O((\log N)^c), c > 1$
Fractional power: $O(N^c), 0 < c < 1$
Sub-linear: $o(N)$
Linear: $O(N)$
Quasi-linear: $O(N \cdot (\log N)^c), c \geq 0$

Polynomial: $O(N^c), c > 1$
Quadratic: $O(N^2)$
Exponential: $O(c^N), c > 1$

2.2 Peer-to-Peer Applications

A peer-to-peer (P2P) application is a software that runs on end-systems *devices*, e. g., personal computers, notebooks, mobile phones, or tablets, and implements a decentralized network service, e. g., file-sharing or telephony. They are very popular because P2P applications tend to "just work"—without time-consuming and error-prone manual configuration—using the existing networks, e. g., the Internet. Furthermore, most P2P applications *scale* because each instance of the P2P application contributes a limited amount of the device's resources for running the network service. Since P2P applications implement a decentralized network service they commonly do not have a single-point of failure, e. g., a central server. Today's P2P applications are widely deployed, e. g., BitTorrent for file-sharing or Skype for Internet telephony. Furthermore, the Spontaneous Virtual Network (SpoVNet) project uses technology behind P2P applications to provide an evolutionary approach to enable future Internet services.

On deployment, instances of the P2P application run on devices attached to one or more networks, e. g., the Internet, VPNs, or LANs. To provide a network service, the P2P application uses a P2P protocol that builds an overlay network, or just *overlay*, using the existing networks. The *underlay* comprises all networks used by the devices. To build an overlay, the P2P protocol uses the transport protocols in the underlay to communicate with other instances of the P2P application using *underlay addresses*, e. g., IP address and UDP port in the Internet.

Overlays

An overlay is a logical network consisting of *nodes* and *links*. The overlay size denotes the total number of nodes in the overlay. A link between two nodes indicates that they can communicate bi-directionally by using *underlay addresses*. A *unidirectional link* between two nodes indicates that a node can sent a message to another node by using an *underlay address*. If a node only knows or has stored the *underlay address* of another node, but has not yet sent or received a message, the *underlay address* represents a *contact* of another node. The nodes connected to a node x via links are the node x's *overlay neighbors*.

A node can establish a link to another node in the overlay by actively sending messages using the node's underlay addresses. A link is removed from the overlay when nodes cannot exchange messages bi-directionally anymore, e. g., when a node failed or one node can't send or return a message. A node may detect a link failure by probing, e. g., using keep-alive messages to check if the node can send and receive messages via a link.

How P2P Protocols Build an Overlay

P2P protocols employ two main mechanisms to build an overlay:

- *Bootstrapping:* First, a new node establishes at least one link to an initial node already in the overlay by discovering the initial node's underlay address. This can be done, for example, by using a central server as rendezvous-point, by using a cache with addresses of nodes recently seen or by broadcasting the local network.
- *Overlay maintenance:* Second, if required, each node exchanges underlay addresses with its overlay neighbors. Subsequently, the node may establish new links to further nodes and may drop existing links to re-structure the overlay until it has a *topology* serving the network service best, or at least best possible. Examples of a network service are group membership, routing, or multicast services. Furthermore, the P2P application adapts the overlay when new nodes join, leave or fail.

Overlay Graph

An *overlay graph* $\mathcal{O} := (\mathcal{V}, \mathcal{E})$ can be used to describe overlay topology theoretically. In this graph a vertex $v \in \mathcal{V}$ of the graph represents a node and an edge $e \in \mathcal{E} \subseteq \{\mathcal{V} \times \mathcal{V}\}$ in the graph represents a link in the overlay. The set $\Gamma(v; \mathcal{O}) \subset \mathcal{V}$ denotes the overlay neighbors of a vertex v. To describe an overlay with unidirectional links, the *directed overlay graph* uses directed edges to represent unidirectional links.

Churn and Overlay Partitioning

Overlays of P2P applications are highly dynamic, i. e., the topology can be changed by the P2P protocol anytime and is naturally changed in cases of *churn*[1], i. e., nodes joining, leaving, and node failures. These dynamics may cause that an overlay graph partitions into several disconnected parts. Disconnected means, that at least two sets A and B with $|A \cup B| \stackrel{!}{=} N$ (N denotes the overlay size) exist, where not a single node in A connects to a node in B via a link. This issue is called *overlay partitioning*. If an overlay is not partitioned, it is *connected*.

Transitive Connectivity and Heterogeneity

Most P2P applications rely on *transitive connectivity* provided by the underlay. This means that a node Y may send an underlay address of an overlay neighbor X to another overlay neighbor Z which subsequently can establish a link with node X using the received underlay address. This can be translated into the communication relation: when nodes X and Y, and, nodes Y and Z can exchange messages, then nodes X and Z can exchange messages (cf. $(X, Y) \wedge (Y, Z) \Rightarrow (X, Z)$). Commonly, nodes attached to the same network in the underlay, e. g., the Internet with IPv4 addressing, have transitive connectivity. The presence of transitive connectivity among a set of nodes implies that

- each node can sent a message to any other node using a valid underlay address,
- all nodes have *direct* connectivity, i. e., can communicate *directly* by only using the connectivity provided by the underlay, and,
- the P2P protocol can build an overlay with an arbitrary topology.

[1] Observations of real churn are provided in [88].

If the underlay does not provide transitive connectivity among all nodes of a P2P application, the underlay is *heterogeneous*. The reasons for non-transitive connectivity are manifold, e. g., the underlay may comprise several networks that use different addresses, address spaces, or, transport protocols. The following sections describe two of the most popular overlay types: unstructured and structured overlays.

2.2.1 Unstructured Overlays

Unstructured overlays provide group membership services by using a connected overlay. For this purpose, the P2P application builds an overlay where all nodes connect via a limited number of links to randomly chosen overlay neighbors. Therefore, the overlay graph $\mathcal{O}$ is a random graph, cf.. Section 2.1.1, and connected with high probability. The random topology usually makes this type of overlay highly resilient against overlay partitioning in cases of churn because each node *and* link is equally important in the overlay. For the same reason, unstructured overlays do not require transitive connectivity provided by the underlay. This is an essential property for the CMP protocol described in Chapter 4.

One of the first P2P file-sharing applications that builds an unstructured overlay is GIA [16]. Furthermore, a complete class of protocols for P2P applications, so-called *gossiping protocols*, build unstructured overlays to efficiently and reliably disseminate data among *all* nodes in the overlay.

For finding a node in an unstructured overlay, e. g., one that stores a certain content, the P2P application needs to flood query messages in the unstructured overlay. In consequence, many nodes not matching the query's criteria receive and process these messages. This suggests that unstructured overlays are not very efficient and do not scale with the overlay size and number of queries. As an alternative, structured overlays, presented in the next section, provide a scalable method to find nodes or store data.

2.2.2 Structured Overlays

In contrast to unstructured overlays, structured overlays have a structured topology to fulfill a certain task, e. g., efficiently finding a node or content stored in the node. The most prominent class of P2P protocols that build structured overlays are key-based routing (KBRs) protocols. They serve as basis for distributed hash-tables (DHTs) which allow, e. g., file-sharing P2P applications to find files efficiently, or a user record for P2P telephony [6]. One of the earliest and most prominent structured overlay protocol is Chord.

Chord

Chord [87] builds an overlay that arranges all nodes on a ring overlay topology. In this ring, each node has a link to a successor and predecessor node. Each node x on the ring is equipped with an node identifier, i. e., $\mathsf{NID}(x)$. The successor $s := succ(x)$ of a node x either has a node identifier that is greater, i. e., $\mathsf{NID}(x) < \mathsf{NID}(s)$ or has the smallest identifier of all nodes. Respectively, the predecessor $p := pred(x)$ of a node either has a node identifier that is smaller, i. e., $\mathsf{NID}(x) > \mathsf{NID}(p)$ or has the greatest identifier of all nodes. The ring forms a convex address space. When a node intends to route a message to an other node y, it just routes it greedily in

Term	Description
Router	A node running a routing protocol
Physical link	Bi-directional communication channel between two nodes on the link layer
Physical neighbors, $\Gamma(x)$	The nodes connected to a node x by physical links
$\omega(x, y)$	Message forwarding costs over a physical link between nodes x and y
Source	Sender node
Destination	Receiver node
Locator	Location-dependent address, e. g., an IP-address
Identifier	Location-independent address, e. g., a node identifier

Table 2.2 – Routing: terms and notations.

the ring based on the distance on the ring towards the destination node identifier $\mathsf{NID}(y)$. Since this would result in a average path length of $O(N)$ hops in the ring, Chord adds a logarithmic number of so-called "fingers" to each node. The $i-th$ finger is a link to the 2^i predecessor on the ring. It has been shown that this reduces the number of required hops to logarithmic complexity.

2.3 Routing

In a network, a node running a routing protocol is a router. Each router in a network has network interfaces which can receive or forward messages from and to the node's physical neighbors over a physical link. A router learns about its physical neighbors using a neighbor discovery mechanism. This mechanism for is different for each network type, e. g., routers in wireless networks may use beacons, routers in fixed networks may use the information provided by the link layer. Forwarding a message over a physical link is associated with *costs*, e. g., bandwidth consumption and/or delay. This cost is expressed by a function $\omega(x, y) \in \mathbb{N}$ for a physical link between the routers x and y. ω does not have to be symmetric, i. e., $\omega(x, y) \neq \omega(y, x)$ is possible. The unit "hop(s)" denotes the number of physical links a message traverses until it reaches the destination. A network of physical links and routers forms an abstract network topology. The network topology is modeled as a graph $G_R := (V, E)$ where routers are represented by vertexes and physical links by edges between them.

The main purpose of routing is the forwarding of messages routers from a sender, the *source*, to a receiver, the *destination*. To route messages, routing protocols use addresses. On the one hand, an address may depend on the location of a router in the network topology. Those addresses are called *locators*. On the other hand, an address may be a arbitrary unique *identifier* which is independent from the location and the network topology.

Most routing algorithms minimize the routing costs for routing a message from the source to the destination. This is equivalent to finding the *shortest-path* in terms of routing costs. It is not surprising that traditional routing algorithms

are in fact decentralized versions of the well-known shortest-path algorithms, e. g., Dijkstra [23], Bellman-Ford [35], and Floyd-Warshall [32].

2.3.1 Distance Vector Routing

The distance vector algorithm (DV algorithm) implements a distributed version of the Bellman-Ford's [35] algorithm. The DV algorithm holds a vector of distances for each physical neighbor. Each vector contains the distances to other routers via the physical neighbor. This ends up in a cost matrix. When the Let x, u, v be routers and $\delta_{u,v} \in \mathbb{R}$ represent the distance-matrix on a router x containing the distance $\delta_{u,v}$ to router u via router v. When DV algorithm starts, each router x knows only its physical neighbors $\Gamma(x)$. In this case, the DV algorithm initializes the matrix on router x with the costs to forward a message to the physical neighbors, i. e., $\delta_{y,y} := \omega(x, y), y \in \Gamma(x)$. Then, the DV sends periodic updates of the routers minimal distance vector, i. e.,

$$\delta'_u := \min \delta_{u,v}$$

to all its physical neighbors. If a router x receives an update from router y, it adds the costs $\omega(y, x)$ to all entries of the vector and updates his distance-matrix

$$\delta_{u,y} := \delta'_u + \omega(y, x)$$

accordingly.

Figure 2.3 illustrates the distance matrix of four routers (Ⓐ,...,Ⓓ). Values printed in bold have been changed. The gray table cells denote the shortest distance to the destination, i. e., the contents of δ'. A new shortest distance is transferred to the neighbors of a router. At T=1, the DV algorithm initializes the distance matrices on each router with the forwarding costs to the router's neighbors. Subsequently, the DV algorithm sends distance vector updates to all neighbors of each router. At T=2, the DV algorithm on router Ⓐ receives the distance vector comprising the distance to router Ⓒ, $\delta'_C = 9$ and to router Ⓓ, $\delta'_D = 1$. The DV algorithm on router Ⓐ adds the costs $\omega(A, B)$ to this distance vector and assigns these values to its distance matrix, i. e., $\delta_{C,B} := 12$, and $\delta_{D,B} := 4$. The DV algorithm does the same on all other routers with the respective distance vector updates. Then, the updates in the distance vector δ' are again sent to the neighbors of each router. T=3 and T=4 perform the same operations. In T=4 the DV algorithm has reached convergence.

Count-To-Infinity

The distance vector algorithm has a performance penalty when routers fail, i. e., it takes a long time until all routers set the according distance to infinity. Consider T=4 in the example of Figure 2.3. If router Ⓐ fails, router Ⓑ will set $\delta_{A,A} := \infty$. Router Ⓑ still knows a route to router Ⓐ via Ⓒ or Ⓓ. Therefore, the DV algorithm on router Ⓑ will just update the minimal distance vector $\delta'_A := 15$ and send it to its neighbors. Then, the DV algorithm sets $\delta_{A,B}$ on router Ⓒ to $15 + 9 = 24$ and on router Ⓓ to $15 + 1 = 16$. The subsequent updates sent to router Ⓑ increase the minimal distance vector δ'_A by 2 to $16 + 1 = 17$. The DV algorithm on router Ⓑ will send his updated minimal distance vector to the

router's neighbors and so forth. The DV algorithm is in an infinite loop counting the distances to router Ⓐ to infinity.

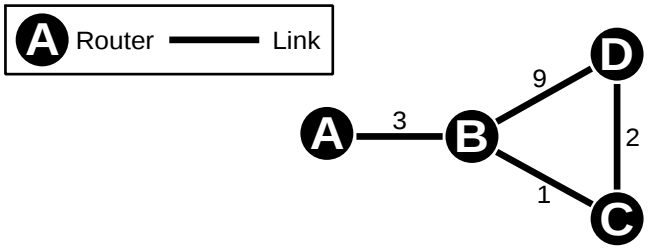
2.3.2 Destination Sequenced Distance Vector Routing

The Destination-Sequenced Distance Vector (DSDV) algorithm was introduced by Perkins et al. in 1994 [72]. Later, Babel [61] refined DSDV to make it more powerful and more efficient. DSDV uses a routing-table where each entry comprises

- the destination's address, e. g., locator or identifier,
- the costs for routing,
- the destination sequence number, and,
- the *next-hop*, e. g., the physical link over which the routing protocol will forward a received message addressed to the destination.

The main idea behind DSDV is, that each router announces its address in conjunction with a sequence number. This sequence number increases on any routing update by the destination (or originate) node. When a node receives routing updates from its neighbors it adapts the distance of an routing-table entry, if the distance in the update is smaller, or the sequence number is greater. This principle solves the count-to-infinity problem that comes with distance vector (DV) protocols. This is the case, because a worse route is only accepted by other nodes when the origin node has chosen a new sequence number. Consequently, nodes have to choose a new sequence number until the protocol re-converges to a stable state in case of dynamic networks.

Figure 2.4 depicts the same network as shown in Figure 2.3 using the DSDV algorithm. First, DSDV fills the routing-table with the distances to the physical neighbors and itself. Then, DSDV periodically notifies the physical neighbors about changes in its routing-table. Therefore, in T=2, the routers learn about new routes. The DSDV algorithm only accepts routes whose routes whose distance is shorter or sequence number is greater. In step, T=3, router Ⓒ replaces the route to router Ⓐ with a distance of $\delta = 6$ via the next-hop over router Ⓒ. The DSDV algorithm has converged.



T=1:

Router A

to	via A	via B	via C	via D
A				
B		**3**		
C		∞		
D		∞		

Router B

to	via A	via B	via C	via D
A	**3**		∞	∞
B				
C	∞		**9**	∞
D	∞		∞	**1**

Router C

to	via A	via B	via C	via D
A		∞		∞
B		**9**		∞
C				
D		∞		**2**

Router D

to	via A	via B	via C	via D
A		∞	∞	
B		**1**	∞	
C		∞	**2**	
D				

T=2:

Router A

to	via A	via B	via C	via D
A				
B		3		
C		**12**		
D		**4**		

Router B

to	via A	via B	via C	via D
A	3		∞	∞
B				
C	∞		9	**3**
D	∞		**11**	1

Router C

to	via A	via B	via C	via D
A		**12**		∞
B		9		**3**
C				
D		**10**		2

Router D

to	via A	via B	via C	via D
A		**4**	**14**	
B		1	**11**	
C		**10**	2	
D				

T=3:

Router A

to	via A	via B	via C	via D
A				
B		3		
C		**6**		
D		4		

Router B

to	via A	via B	via C	via D
A	3		**21**	**15**
B				
C	**9**		9	3
D	**7**		11	1

Router C

to	via A	via B	via C	via D
A		12		**6**
B		9		**3**
C				
D		10		2

Router D

to	via A	via B	via C	via D
A		4	14	
B		1	11	
C		10	2	
D				

T=4:

Router A

to	via A	via B	via C	via D
A				
B		3		
C		6		
D		4		

Router B

to	via A	via B	via C	via D
A	3		**15**	15
B				
C	9		9	3
D	7		11	1

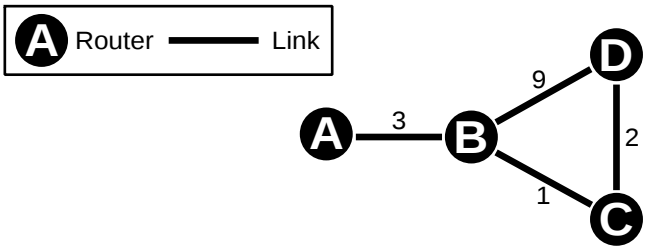
Router C

to	via A	via B	via C	via D
A		12		6
B		9		3
C				
D		10		2

Router D

to	via A	via B	via C	via D
A		4	14	
B		1	11	
C		10	2	
D				

Figure 2.3 – Example of the distance vector algorithm. Values printed in **bold** have been changed. The gray table cells denote the shortest distance to the destination. A new shortest distance is transferred to the neighbors of a router.



T=1:

Router A

to	next	δ	seq
A	A	0	1
B	B	3	1
C	-	-	-
D	-	-	-

Router B

to	next	δ	seq
A	A	3	1
B	B	0	1
C	C	9	1
D	D	1	1

Router C

to	next	δ	seq
A	-	-	-
B	B	9	1
C	C	0	1
D	D	2	1

Router D

to	next	δ	seq
A	-	-	-
B	B	1	1
C	C	2	1
D	D	0	1

T=2:

Router A

to	next	δ	seq
A	A	0	1
B	B	3	1
C	**B**	**12**	**1**
D	**B**	**4**	**1**

Router B

to	next	δ	seq
A	A	3	1
B	B	0	1
C	**D**	**3**	1
D	D	1	1

Router C

to	next	δ	seq
A	**B**	**12**	1
B	**D**	**3**	1
C	C	0	1
D	D	2	1

Router D

to	next	δ	seq
A	**B**	**4**	1
B	B	1	1
C	C	2	1
D	D	0	1

T=3:

Router A

to	next	δ	seq
A	A	0	1
B	B	3	1
C	B	**6**	1
D	B	4	1

Router B

to	next	δ	seq
A	A	3	1
B	B	0	1
C	D	3	1
D	D	1	1

Router C

to	next	δ	seq
A	**D**	**6**	1
B	D	3	1
C	C	0	1
D	D	2	1

Router D

to	next	δ	seq
A	B	4	1
B	B	1	1
C	C	2	1
D	D	0	1

Figure 2.4 – Example of the destination sequenced distance vector algorithm. Values printed in **bold** have been changed. The table headers have the follwing meaning: to denotes the destination, next denotes the next-hop towards the destination, δ denotes the distance to the destination, and seq denotes the current sequence number.

Chapter 3

Underlay Abstraction

The idea of abstracting from the underlay is one of the research topics in Spontaneous Virtual Networks project (SpoVNet). SpoVNet has two major objectives [17]:

1. Provide communication services flexibly, adaptively and spontaneously in heterogeneous networks.
2. Enable seamless transition from current to future networks.

To achieve these objectives, SpoVNet proposes a two-tier abstraction architecture consisting of an underlay and a service abstraction. The underlay abstraction has the goal to abstract from heterogeneous networks in order to provide a solid base for implementing advanced services, e. g., multicast or event services. The service abstraction has the goal to abstract from different implementations of advanced services. Both tiers provide this abstraction with a well-defined interface which stays valid even when the underlying network evolves. Using this architecture, SpoVNet can emulate advanced services today and replace them with native services when the Internet evolves eventually in the future. Therefore, SpoVNet provides an evolutionary approach towards the future Internet.

The first objective states that the approach should work flexibly, adaptively and spontaneously on top of heterogeneous networks. Hence, SpoVNet can take advantage of recent developments in the area of P2P applications which have a similar objective. Additionally, the techniques used with P2P applications usually scale—an additional feature SpoVNet seeks. The protocols used by P2P applications serve as starting-point for emulating advanced services. One drawback is that P2P applications do not support heterogeneous networks, though.

The main motivation behind the concepts presented in this thesis is the need of an proper underlay abstraction that enables an easy creation of new P2P-based

services on top of heterogeneous networks. The first concepts towards this goal are presented in [9]. The paper contains several concepts to deal with heterogeneous networks, e. g., different addressing, quality-of-service, and mobility. It states that the underlay abstraction needs the following features to ease the development of new P2P-based services:

Seamless connectivity: Seamless connectivity enables the transfer of current P2P-based services to heterogeneous networks.

Identifier-based addressing: Identifier-based addressing hides mobility and multi-homing from the P2P-based service. In contrast to locator-based addressing, i. e., using IP-addresses, identifiers do not change, when devices change networks or move. Furthermore, in future networks, devices are most likely multi-homed, i. e., they have access to more than one network and, consequently, more than one locator. This makes the selection of an appropriate locator for communication difficult.

Requirement-based protocol selection: The SpoVNet underlay abstraction proposes to select transport protocols by requirements, e. g., quality of service, latency, reliability, and not by their types.

SpoVNet uses a two layer design to implement these features in the underlay abstraction, namely the

- Base Communication (BC) and
- Base Overlay (BO).

The Base Communication runs once per device and abstracts from heterogeneous networks in terms of different protocols and technologies. This also includes the provisioning of seamless connectivity on top of heterogeneous networks by using so-called relay paths. A Base Overlay runs once per application and uses the Base Communication to provide an isolated identifier-based address space for communication between *SpoVNet-nodes*. Each SpoVNet-node has an unique node identifier inside a spontaneous virtual network, i. e., a SpoVNet, formed by the Base Overlay. Each SpoVNet has an unqiue identifier as well, i. e., the *SpoVNet-identifier*.

Figure 3.1 illustrates two SpoVNets deployed on top of heterogeneous networks. The figure depicts 6 networked devices, i. e., nodes (A,...,E) and 3 heterogeneous networks, i. e., an IPv6, an IPv4, and a Bluetooth network. Node A has access to the IPv6 network, node C has access to the IPv4 network, and node E has access ot the Bluetooth network. Two nodes have access to two networks simultaneously and are therefore denoted relays. Relay B has access to the IPv4 and IPv6 network and relay D has access to the IPv4 and Bluetooth network. On each node runs the SpoVNet underlay abstraction consisting of the Base Communication and the Base Overlay. The Base Communication runs exactly once per node, the Base Overlay once per SpoVNet-application. Each SpoVNet has an identfier, i. e., SpoVNet1 with SpoVNet-identifier 0x2608 and SpoVNet2 with SpoVNet-identifier 0x1006. SpoVNet1 consists of SpoVNet-nodes with NID 1 running on node A, NID 2 running on node C, and NID 3 running on node B. SpoVNet2 consists of SpoVNet-nodes with NID 4 running on node C, NID 5 running on node E, and NID 6 running on node D. Since node C accomodates two SpoVNet nodes, two

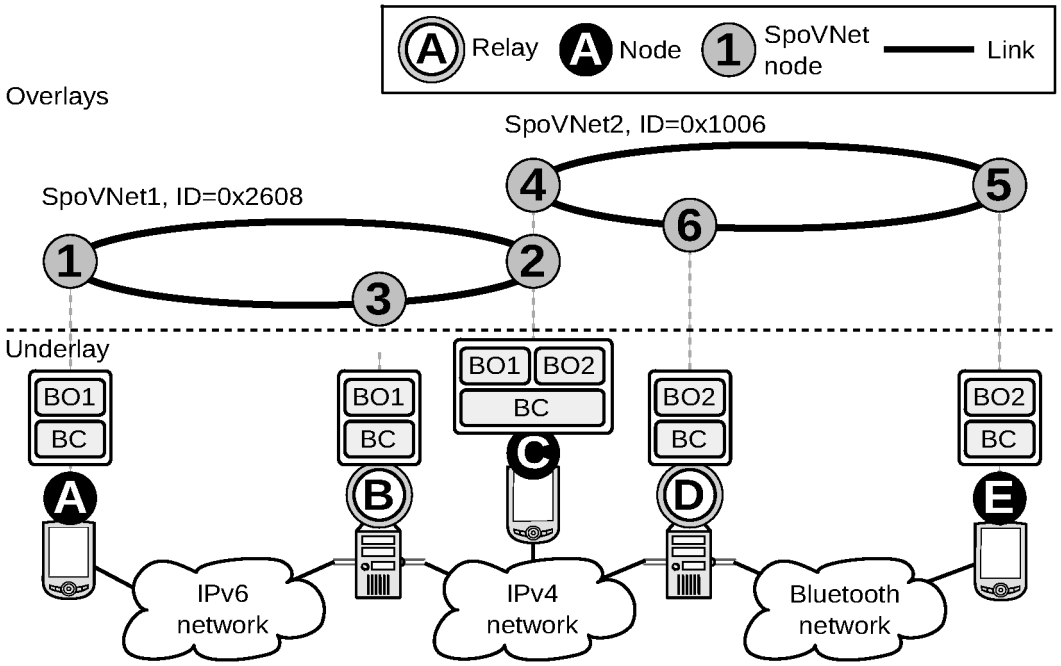


Figure 3.1 – Two spontaneous virtual networks (SpoVNets).

instances of the Base Overlay run on one instance of the Base Communication. The application running on each SpoVNet-node can communciate with SpoVNet-nodes in the same SpoVNet using the node identifiers in their respective SpoVNets.

The *ariba*-underlay [64], in development at the Karlsruhe Institute of Technology (KIT), implements a part of the SpoVNet underlay abstraction. From the service and application perspective, *ariba* provides a virtual network substrate that features identifier-based addressing and a unified developer interface. *ariba* hides the complex handling of heterogeneous networks and mobility from the developer. To achieve this, *ariba* routes messages across heterogeneous network borders and hides changing underlay addresses when nodes move or join other networks. This eases the implementation of new services and applications significantly.

Figure 3.2 shows an overview of the *ariba* architecture. At the top layer, applications and services use the *ariba* developer and legacy interface. The latter allows existing applications to benefit from *ariba*'s features without any changes. To achieve this, *ariba* emulates an IP-based network [45]. Each *ariba*-based application, i. e., using the developer interface presented in Section 3.1, may use several *ariba*-based services or use *ariba* directly. *ariba* provides an isolated virtual network with identifier-based addressing between all SpoVNet-nodes for each application and its services, the Base Overlay, described in Section 3.2. The Base Overlay itself builds its virtual network using links from the Base Communication.

The Base Communication runs once per device and handles all links requested from all Base Overlays of the applications, heterogeneous addresses, and provides relay-based communication between heterogeneous networks. For this purpose, it implements several modules for different underlay protocols, e. g., TCP with IPv4

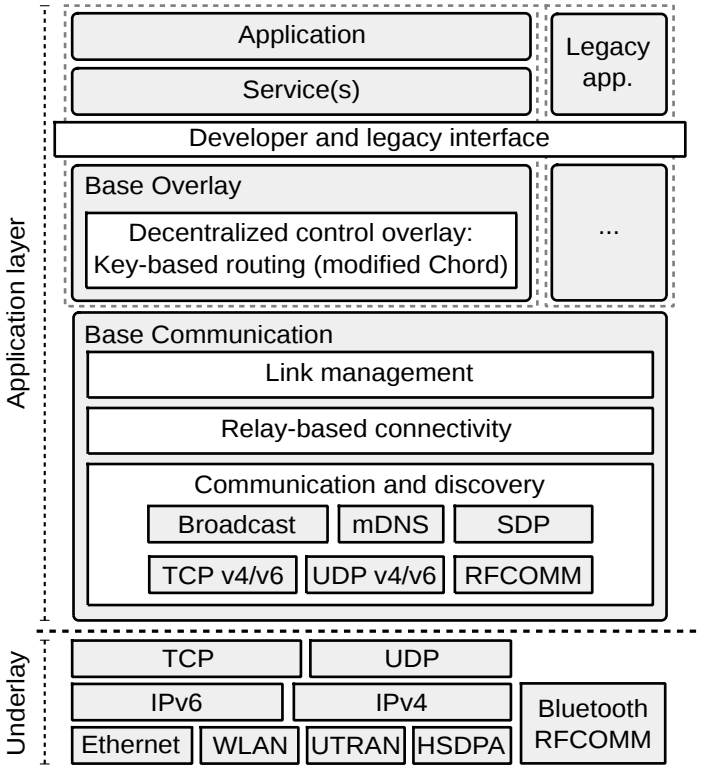


Figure 3.2 – The *ariba*-underlay abstraction's architecture overview.

and IPv6, and modules to discover other SpoVNet-nodes that are already running the application for bootstrapping, e. g., using broadcast, or Bluetooth SDP. The base communication interfaces directly with the (heterogeneous) underlay that comprises many different protocols, e. g., TCP (Layer 4) or Bluetooth RFCOMM (Layer 2). Section 3.3 describes the base communication in further detail.

ariba has proven itself in many applications, e. g., demos shown at conferences. Section 3.4 presents some of the highlights. The concepts developed in this thesis go beyond some of the implemented features in *ariba*. Section 3.5 describes the motivation and contributions for the CD middleware based on *ariba*.

3.1 Developer Interface

The *ariba* developer interface eases the creation of new decentralized applications and services. Table 3.1 summarizes the main functionality of the *ariba* developer interface. It comprises methods to initiate a new spontaneous network (SpoVNet) for the application, join/leave an a SpoVNet. *ariba* uses an event-based and asynchronous processing model. This model implies, that all methods immediately return. Events are emitted by *ariba* as soon as it has completed the respective operation of the method. Thus, a service or application needs to handle the events, when *ariba* has completed the join or leave of a SpoVNet-node. It also informs the service or application in case of failure of an operation.

When a new node has joined the SpoVNet an application or service is able to send messages to other SpoVNet-nodes. For this purpose, an application or service

Function	Description
Methods for Base Overlay control	
initiate	Creates an application-specific SpoVNet
join	Join an existing SpoVNet
leave	Leave the SpoVNet
Events emitted by the Base Overlay	
onJoinCompleted	Join succeeded
onJoinFailed	Join failed
onLeaveCompleted	Node left the SpoVNet
onLeaveFailed	Node could not leave the SpoVNet
Methods for communication	
bind	Binds a service ID
unbind	Unbinds a service ID
establishLink	Establishes a link to another SpoVNet-node
dropLink	Drops a link
sendMessage	Sends a message over a link or to a SpoVNet-node
Events emitted during communication	
onLinkUp	Link has been established
onLinkDown	Link has been dropped
onLinkChanged	Link has been changed
onLinkFailed	Link failed
onLinkRequest	Incoming link request
onMessage	Incoming message on a link

Table 3.1 – *ariba*'s developer interface.

can bind service identifiers, a 16-bit numeric value that identifies the application or a service. This identifier should be well-known by all SpoVNet-nodes running the same service or application. To exchange messages between SpoVNet-nodes, *ariba* provides two ways of communication.

First, it is possible to establish a link to another SpoVNet-node using a node identifier and service identifier, so messages can be sent over the link to the other SpoVNet-node. Second, it is possible to send a message without setting up a link first using a SpoVNet-node and service ID—*ariba* will then establish a link and deliver the message over this link internally. The difference between these two modes is, that links may have a certain context (i. e., different link properties) and are identified by a link ID.

ariba emits several events during communication. The major part of these events inform the service or application about successful link establishment, link failure, link change, or incoming link requests from other SpoVNet-nodes. Furthermore, a callback informs the service or application about incoming messages from another SpoVNet-nodes.

The developer interface also hides all issues related to heterogeneous networks and mobility using a homogeneous identifier-based addressing. The main module

providing this addressing and the virtual network substrate for the application is the Base Overlay.

3.2 Base Overlay

The Base Overlay provides a virtual network substrate for services and applications. The Base Overlay itself uses links from the Base Communication to build a key-based routing (KBR) overlay. When an SpoVNet-node joins a SpoVNet, it actually joins this KBR overlay. The KBR overlay serves two purposes. First, it provides a identifier addressing space, so each SpoVNet-node in the KBR overlay can be reached using a SpoVNet-node identifier. Second, it provides the isolated virtual network substrate to the services or application. The implementation of the KBR overlay depends on the service's or application's requirements. In general, any KBR overlay may be implemented in *ariba* because of its modular design. The current release of *ariba* implements two KBR overlays services and applications may choose from:

- *Full mesh:* The full-mesh KBR overlay connects each SpoVNet-node with any other SpoVNet-node. Thus, this approach is not scalable.
- *Modified Chord:* The modified Chord KBR overlay builds a ring overlay topology with $\log N$ fingers (or "short-cuts"). Thus, the modified Chord, on the one hand, scales well with the network size N, because each SpoVNet-node connects to $O(\log N)$ other SpoVNet-nodes only. On the other hand, to reach other SpoVNet-nodes the modified Chord needs $O(\log N)$ hops. The main difference to the original Chord is, that the modified Chord uses bi-directional successors, predecessors, and fingers. Furthermore, it uses discovery messages to build the ring similar to those used in TRout (cf. Section 5.3). These modifications allow the modified Chord to recover from network partitioning.

These implementations are rudimentary, but demonstrate the general flexibility of *ariba*'s design. In further work, *ariba* may be extended by a variety of different KBR overlays depending on the requirements of the service or application. Examples for other KBR overlays are Kademlia [65], or Bamboo [76].

3.3 Base Communication

As mentioned before, the KBR overlay is built using links from the Base Communication. The Base Communication has the following properties:

- The Base Communication *abstracts* from heterogeneous addressing and networks. To this end, it uses *end-point descriptors* that comprise all underlay addresses of a node. The KBR overlay can exchange these end-point descriptors between SpoVNet-nodes without knowing its exact contents. Subsequently, the KBR overlay can use the end-point descriptors to build new links between nodes using the Base Communication. Section 3.3.1 describes the communication and discovery sub-module of the Base Communication that provides this abstraction.
- Furthermore, the Base Communication provides *relay-based* end-to-end connectivity across heterogeneous networks using the relay-based connectivity sub-module described in Section 3.3.2.

ISO/OSI-Layer	Address
4	udp{14632};tcp{1976|12987}
3	ip{141.3.71.46|2a00:1398:9:fb00:a11:96ff:fe16:396c}
2	bluetooth{C8:84:47:02:CD:A1};rfcomm{10}

Table 3.2 – An exemplary end-point descriptor.

– Finally, the Base Communication monitors and manages links from all KBR overlays in the link management sub-module described in Section 3.3.3.

The following describes each of the sub-modules in further detail.

3.3.1 Communication and Discovery

The communication and discovery sub-module provides the following functionality:

– Discovery of end-point descriptors of other SpoVNet-nodes.
– Establishment of links for the Base Overlay.

To provide this functionality the Base Communication uses end-point descriptors. The communication and discovery comprises additional components, the discovery components and the communication components.

End-Point Descriptors

An end-point descriptor comprises all underlay addresses a node has, e. g., TCP/UDP ports, IPv4/v6, Bluetooth medium access addresses (MAC), or RFCOMM channels. The end-point descriptor holds these underlay addresses loosely-coupled, e. g., a TCP port is stored independently from the IPv4/IPv6 addresses. Table 3.2 shows an exemplary end-point descriptor. It contains a UDP port, two TCP ports, an IPv4 address, an IPv6 address, a Bluetooth MAC and a RFCOMM channel. The sub-modules of the Base Communication will combine the addresses of each layer to communicate with other nodes.

Discovery of End-Point Descriptors of Other Nodes

To bootstrap the Base Overlay, the Base Communication provides discovery mechanisms that announce and collect end-point descriptors from other SpoVNet-nodes also running the *ariba*-underlay. The discovery components of the Discovery and Communication sub-module implement this functionality for different network technologies. For example, the Base Communication announces a node's end-point descriptor using the IPv4/v6 one hop multicast functionality, the multicast DNS (mDNS) service, or the service discovery protocol (SDP) of the Bluetooth network stack. Each announcement contains the SpoVNet-identifiers, so an application can determine if the node is in the same SpoVNet. The Base Communication collects these announcements to provide them to the Base Overlay for bootstrapping, i. e., joining a SpoVNet.

Establishment of Links

The interface to establish new links for the KBR overlay is similar to what the Base Overlay provides to the services and application (cf., methods for communication

in Table 3.1). The main difference is that the Base Communication uses end-point descriptors for addressing other nodes.

When the Base Overlay asks the Base Communication to establish a link, the Base Communication tries to reach the other node using *all* addresses from the end-point descriptor. For this purpose, it tries to send the message using all communication components of the communication and discovery sub-module, e. g., TCP/UDP using IPv4/v6, or Bluetooth RFCOMM. Each communication component extracts the underlay addresses it understands from the end-point descriptor and tries to send link request to the other node.

When the Base Communication on a node receives a link request from another node, it immediately responses with an acknowledgement message. Hence, a node receives potentially multiple link requests and responds with multiple acknowledgement messages. As the Base Communication only needs to establish one link, it sets up the link using the underlay address for which it receives the first acknowledgement message.

The mechanism above works reliably for nodes using the same protocols and being in the same network. In case of network heterogeneity, when nodes cannot communicate with each other *directly*, the next section describes the relay-based connectivity sub-module that handles this problem.

3.3.2 Relay-based Connectivity

Relay-based connectivity is required when the KBR overlay attempts to establish a link to a SpoVNet-node in a different network, so that a link cannot be established with the end-point descriptor. As mentioned in Chapter 2.2, KBR overlays are built incrementally, starting with links to overlay neighbors and subsequently refining the overlay by exchanging addresses. In case of *ariba*, end-point descriptors are exchanged. For finding an *indirect* path between SpoVNet-nodes, the relay-based connectivity sub-module monitors the path of the end-point descriptor in the Base Overlay. If a link cannot be established directly between two SpoVNet-nodes, the Base Communication uses this "relay path" to communicate indirectly. Each node on the relay path tries to shorten the path by trying to establish a direct link to one of the next hops. This induces a lot of overhead subject to the relay path length as each SpoVNet-node will greedily try to shorten the path concurrently. Bryan Ford's "Unmananged Internet Architecture" (UIA) [34] uses similar mechanisms.

3.3.3 Link Management

When a link has been established, the Base Communication handles the link management. This includes

- partial hiding of mobility, i. e., the internal re-establishment of failed links in case of mobility using an updated end-point descriptor and
- monitoring if the link is operational by sending keep-alive messages.

Link management increases the reliability of links and, thus, the Base Overlay. Furthermore, the link management sub-module may be extended by native mobility support, e. g., mobile IP, and additional link failure detection mechanisms, e. g., when a network interface or access fails.

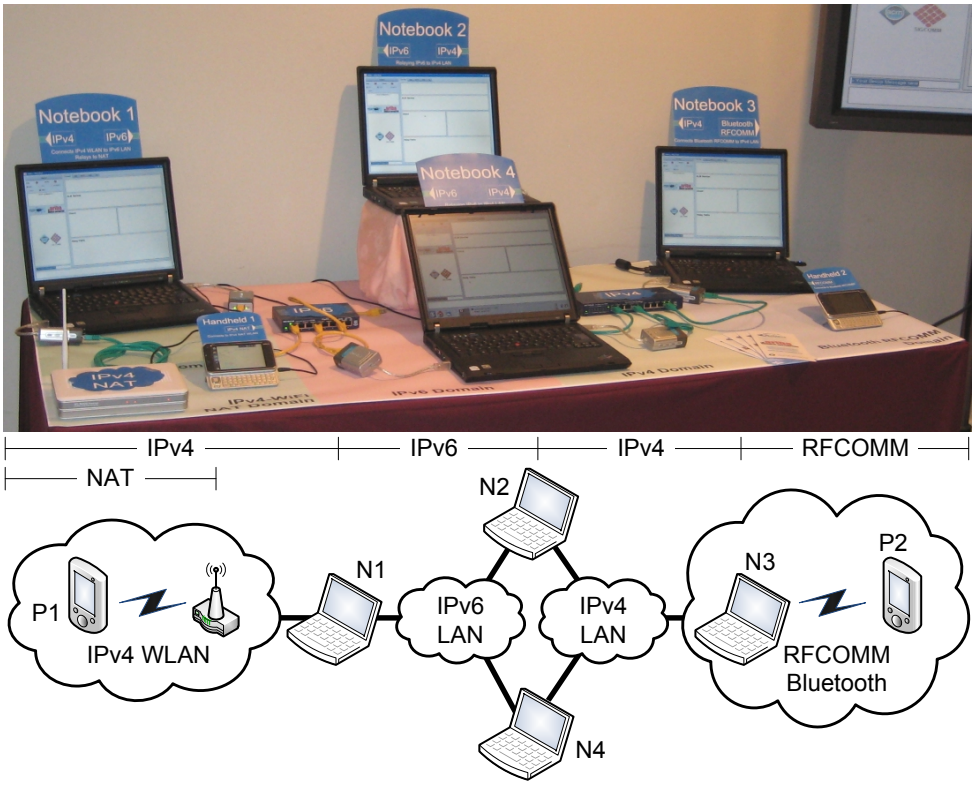


Figure 3.3 – The *ariba* demo setup. Source: [46]

3.4 Demos and Applications

ariba has proven itself at many conferences, demos and a programming contest in small-scale scenarios. The basic demonstration setup of *ariba* comprises 4 heterogeneous networks: an IPv4 WLAN behind a NAT WLAN router, an IPv6 LAN, an IPv4 LAN and a Bluetooth network. None of these networks are connected; however, 6 devices connect to these heterogeneous networks, four of them are multi-homed, i. e., connect to more than one network.

Figure 3.3 shows the real-world *ariba* demonstration setup for two demos. One of the two *ariba*-demos shows the self-organizing provisioning of end-to-end connectivity [46]. The demo has been awarded with the 2nd place "best-demo" at SIGCOMM 2009 in Barcelona, Spain and received an honourable mention. The other one shows unmodified legacy applications running with *ariba* [45].

The main feature of both demos is self-organization. The network configuration in the demos can be changed interactively by unplugging Ethernet cables from the switches and re-plugging them. In each case, *ariba* adapts to the changed network and re-establishes end-to-end connectivity, if possible.

3.5 Motivation and Contribution of the CD Middleware

The concepts of the CD middleware contribute to the relay-based connectivity sub-module of the Base Communication. The main motivation are the following problems that came up during the development of *ariba*:

- *Coupling between Base Overlay and Base Communication:* To find relay-paths the Base Overlay and Base Communication modules need to be tightly coupled. This means, the Base Overlay must inform the Base Communication about the overlay paths to find an appropriate relay path for the link in the Base Overlay. This results in circular dependency which is not desired in a layered architecture.
- *"Highway"-effect and length of relay paths:* Relay-paths are concatenated overlay paths which tend to be very long. Furthermore, for the same reason, it is likely, that the same relay is used most of the time even when more relays exist. *ariba*'s current approach makes it nearly impossible to control, which and how many nodes are involved when routing messages across heterogeneous networks.
- *Path-shortening overhead:* In the current approach each relay-path needs to be shortened separately. This means, each node on the relay-path must probe if it can reach a node on the path directly. Depending on the number of links in the Base Overlay this overhead is significant.

In the process of handling these problems, [98] introduces the concept of connectivity domains. Further work, i. e., [69, 66, 67] refines this concept and is the base of the CD middleware presented in this thesis. The CD middleware solves the following difficulties:

- The CD middleware gives *ariba* more control over the message forwarding between heterogeneous networks, since the CD middleware detects connectivity explicitly. Furthermore, relays are detected by the CD middleware. Therefore, relay paths can be easily shortened, without additional probing.
- The CD middleware uses a dedicated routing protocol to forward messages between Connectivity Domains, which reduces the "Highway"-effects. Furthermore, the length of the relay paths can be optimal (shortest-paths), if required, or grows only logarithmically with the number of nodes. The dedicated routing protocol of the CD middleware decouples the Base Overlay from the Base Communication module.

The communication and discovery layer introduced in Appendix A is almost equivalent to the one used in *ariba*. The differences are results of the lessons learned from small technical issues in *ariba*, but do not differ in the basic functionality.

Chapter 4

Autonomous Detection of Connectivity

This chapter explores the detection of connectivity in the context of P2P applications. Parts of this work have already been presented at the International Conference on Peer-to-Peer Computing 2011 [68].

As mentioned in Section 2.2 a P2P application builds an overlay on existing networks, the *underlay*. Most P2P applications[1] rely on seamless connectivity, more technically "transitive connectivity", provided by the underlay. Transitive connectivity means that when nodes X and Y, and, nodes Y and Z can exchange messages, then, nodes X and Z can exchange messages as well. In today's networks, the assumption of transitive connectivity does not hold in many cases. First, the introduction of IPv6 compromises the transitive connectivity in the Internet as nodes supporting IPv6 only cannot communicate with nodes that support IPv4 only. Second, scenarios including more heterogeneous networks, e. g., private networks behind firewalls, virtual private networks (VPNs), point-to-point connections (e. g., Bluetooth RFCOMM), do not have transitive connectivity.

Related work discusses the problem of non-transitive connectivity in two ways. First, workarounds to P2P applications exist to handle non-transitive connectivity, e. g., [38]. Second, protocols exist that do not need transitive connectivity at all [34, 13]. Both approaches compensate the impact of non-transitivity by tolerating long and non-deterministic detours, overload of nodes compensating non-transitive connectivity, and the bypass of non-transitive connectivity introduced on purpose (e.g., for limiting traffic to a specific domain). The user faces the latter, for example, when Skype uses another client to traverse a firewall. Furthermore, concepts

[1] For example, all P2P applications using structured overlays

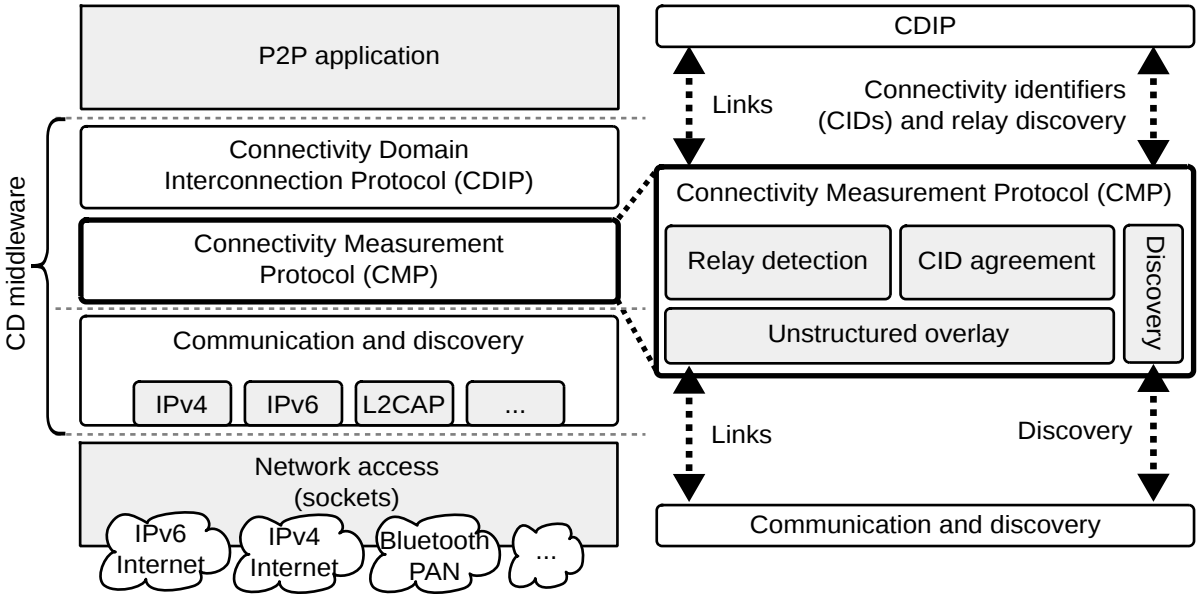


Figure 4.1 – Connectivity detection in the P2P middleware

exist to deal with heterogeneous networks or "network pluralism" [36, 20]. These concepts present architectures that support and promote the interconnection of different networks using interfaces at the border of each network. These concepts provide prior information about transitive connectivity. This allows to tailor P2P applications for each heterogeneous network, and enable traffic-engineering, load-balancing, and, efficient routing. The main problem is that these concepts require manual configuration and infrastructure support. This is hardly an option for P2P applications as they should just work without manual configuration if possible.

This chapter introduces a third concept which has not yet been considered by related work. First, it provides insight under which conditions subsets of nodes that have transitive connectivity can be detected. Second, it contributes the *Connectivity Measurement Protocol* (CMP). CMP proactively detects sets of nodes, so-called *Connectivity Domains* (CDs), that have transitive connectivity. Then, CMP agrees on a unique *Connectivity Identifier* (CID) with nodes in each connectivity domain to differentiate them. Using the CIDs it is known if a node that is in several connectivity domains and, therefore, is a *relay*. CMP adapts the CIDs when connectivity between nodes changes, i. e., when nodes are detached from a network, or attached to other networks.

CMP itself comprises three key mechanisms shown in Figure 4.1. For communication in each homogeneous network, the communication and discovery layer described in Appendix A enables discovery of and communication between instances of the CD middleware running on the nodes. However, the layer provides neither transitive connectivity nor information about nodes that share transitive connectivity.

Using the communication and discovery layer, CMP first builds an unstructured overlay (cf. Section 2.2.1). This overlay comprises all nodes running the CD

middleware and does *not* rely on transitive connectivity. Then, CMP checks for transitive connectivity using a mechanism called *triangle check* to detect if a node is a relay. After detecting all relays, CMP assumes that non-relays have transitive connectivity, i. e., are in the same connectivity domain. Then, CMP uses a random number agreement on the non-relays to agree on an CID for each CD. Simulations of CMP show that CMP agrees on CIDs in less than 20 seconds from a cold-start in the considered scenarios, i. e., a worst-case of CMP. The observed average bandwidth overhead was below 4 kbyte/s per node till convergence. For more results refer to Section 4.6.

The outline of this chapter is as follows: Section 4.1 discusses the problem and solution space under a set of assumptions and requirements of P2P applications. Section 4.2 presents a solution how to autonomously detect connectivity. This solution requires relay detection and random number agreement algorithms. Sections 4.3 and 4.4 provide a feasibility study of different relay detection and random number agreement algorithms and build the fundament for the CMP design. Subsequently, Section 4.5 presents the Connectivity Measurement Protocol (CMP). Section 4.6 shows results of simulations using CMP in heterogeneous networks. Section 4.7 discusses further work and concepts to CMP. Finally, Section 4.8 summarizes the contribution and results of this chapter.

4.1 Problem and Solution Spaces

This section discusses the problem and solution space of detecting the connectivity provided by heterogeneous underlays. To get an intuitive understanding of connectivity, Section 4.1.1 provides an example of a P2P application deployed on a heterogeneous underlay. Using the insight from this example, Section 4.1.2 introduces a formal problem statement. Based on this statement, Section 4.1.3 discusses the problem and solution space of connectivity detection.

4.1.1 Introductory Example

Figure 4.2 illustrates a scenario of an unstructured overlay built by a P2P application on a heterogeneous underlay. The underlay layer shows nodes attached to several networks. Nodes B, C, D are connected to a private home WLAN. The private home WLAN network accesses the Internet using a NAT/Firewall-WLAN-Router. Nodes A and B have a Bluetooth module, thus, they can communicate inside a Bluetooth network. Nodes D, E, F, G have public Internet access, and nodes G and H have access to a virtual private network (VPN).

The overlay layer in the figure shows an exemplary unstructured overlay built by the P2P application. This is possible because unstructured overlays do not require transitive connectivity. In the example, it is assumed that nodes can establish links when they have access to the same network. For example, nodes D and E establish links using the underlay addresses from the public Internet, e. g., IP address and UDP-port. Other pairs of nodes that do not have access to the same network, e. g., nodes C and E, must use a *multi-homed* node, e. g., a *relay* like node D, to communicate *indirectly*.

The heterogeneous underlay does not provide transitive connectivity between all nodes. The figure illustrates (non-)transitive connectivity. On the one hand,

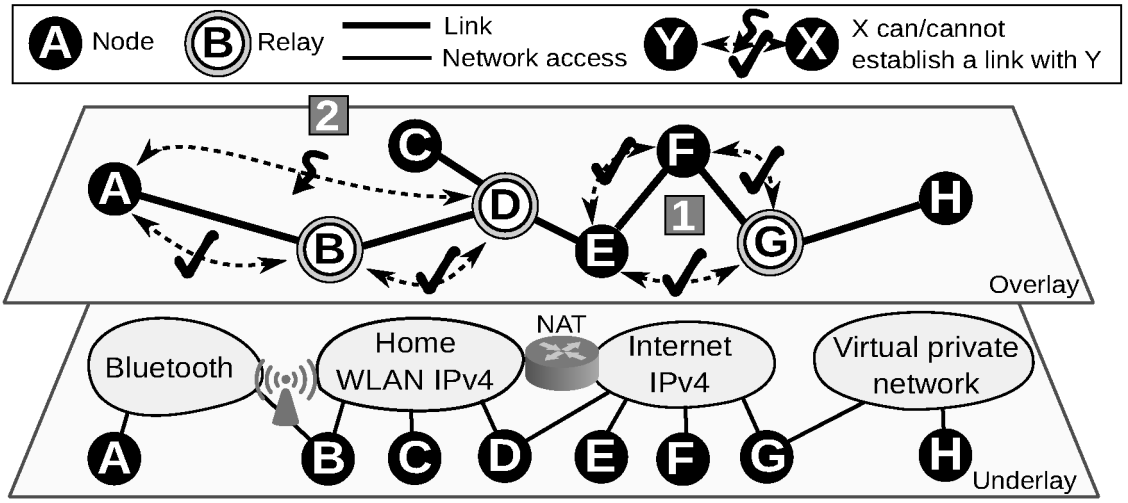


Figure 4.2 – Exemplary scenario of an unstructured overlay built by a P2P application on a heterogeneous underlay with transitive 1 and non-transitive 2 connectivity.

at 1, node F can send the underlay address of node G to node E. Node E can use this address to establish a new link to node G. Thus, when node G can communicate with node F and node F with node E, then node E can also communicate with G because they all have access to the same network. On the other hand, at 2, nodes E, D, and C do not have transitive connectivity, because, if D transfers the underlay address of node C to node E, the underlay address from the Home WLAN is not valid in the public Internet, this means, the nodes cannot establish a link. The nodes are not aware of that à-priori because both networks use the same address types, i. e., IP addresses, and, thus, look valid to the P2P application. However, the addresses belong to networks with disjoint address spaces which make the establishment of a link impossible.

For handling a heterogeneous underlay properly the nodes need to detect sets of nodes that have transitive connectivity, i. e., can communicate directly. Furthermore, nodes that do not have transitive connectivity need to know about appropriate relays to communicate indirectly. The next section formalizes the problem of detecting connectivity.

4.1.2 Problem Statement

The formal problem statement comprises four steps. First, the connectivity function provides insights about potential links two nodes can establish in the overlay. Second, the connectivity graph defines the graph of all potential links between the nodes. Third, the connectivity graph allows the definition sets of nodes with transitive connectivity. Finally, connectivity domains and relays are defined.

Connectivity function: Let $\mathcal{N}$ denote the set of *nodes* in an overlay of size $N = |\mathcal{N}|$. Then, the *connectivity function* defines the ability of establishing a link between two nodes:

$$c : \mathcal{N} \times \mathcal{N} \rightarrow \{true, false\} .$$

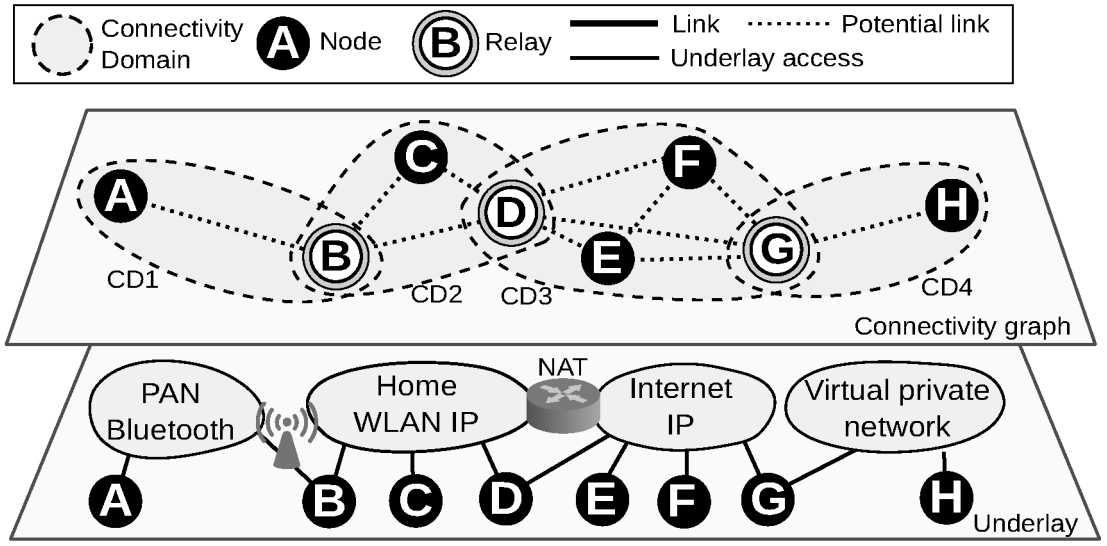


Figure 4.3 – Example of connectivity domains/cliques and relays in the connectivity graph of the scenario presented in Section 4.1.1.

It holds $c(n,m) = \mathit{true}$, if and only if nodes $n, m \in \mathcal{N}$ can establish a link, and $c(n,m) = \mathit{false}$ otherwise. Since the the links in an overlay are bi-directional, $c(n,m)$ is symmetric, i. e., $c(n,m) = \mathit{true} \Leftrightarrow c(m,n) = \mathit{true}$.

Connectivity graph: The connectivity function allows the definition of the *connectivity graph* $\mathcal{G} := (\mathcal{N}, \mathcal{E})$ with the set of nodes $\mathcal{N}$ and the set of undirected edges

$$\mathcal{E} := \{\{n,m\} : c(n,m) = \mathit{true}, n, m \in \mathcal{N}\}\,.$$

Transitive connectivity in the connectivity graph: Using the connectivity graph, a definition of nodes having transitive connectivity is possible. The nodes in set $\mathcal{C} \subset \mathcal{N}$ have transitive connectivity when all nodes can establish links to each other:

$$\forall x, y, z \in \mathcal{C} : c(x,y) \wedge c(y,z) \Rightarrow c(x,z)$$

Hence, in a subset of nodes that have transitive connectivity, there exists an edge between every pair of nodes in the connectivity graph. In other words, the subset is one *clique* in the connectivity graph.

Connectivity domains (CDs) and relays: A maximal set of nodes $\mathcal{C}$ that have transitive connectivity is called *connectivity domain*. In this case, maximal means that the set $\mathcal{C}$ cannot be extended by including one more adjacent node from the connectivity graph without violating the condition of transitive connectivity. $\mathbf{C}$ denotes the set of all connectivity domains. For two connectivity domains $\mathcal{C}_1, \mathcal{C}_2 \in \mathbf{C}$, a node $r \in \mathcal{C}_1 \cap \mathcal{C}_2$ is a *relay* between $\mathcal{C}_1$ and $\mathcal{C}_2$. Relays may also form cliques. It is assumed, however, that $\mathbf{C}$ does not contain any Connectivity Domains comprising relays only.

Figure 4.3 shows the connectivity domains/cliques and relays in the connectivity graph of the scenario presented in Section 4.1.1. The formal problem is to detect

connectivity domains efficiently and find the relays between them. This is similar to enumerating all maximal cliques in a graph, which is a NP-hard problem in general. Therefore, it is necessary to narrow the problem down using the scenario of P2P applications. The next section discusses the problem and solution space in this scenario.

4.1.3 Related Work, Requirements, and Assumptions

Section 4.1.2 shows that determining all connectivity domains and relays is similar to listing all maximal cliques in the connectivity graph of an overlay with size N. The Bron-Kerbosch algorithm [12] would be an intuitive, central solution to this problem. It uses recursive backtracking to find all maximal cliques with $O(3^{N/3})$ complexity in the worst-case. Additionally, [62] provides several new and optimized algorithms for solving the problem, and [26] provides an algorithm for sparse graphs in near-optimal time; however, all these algorithms have a complexity within $O(3^{N/3})$ proven in [92]. There also exist many variants of these algorithms, e. g., variants supporting dynamics [86], or algorithms that distribute the task using a central server [51].

Reducing the Solution Space

The main issue of the algorithms and their variants is, that they need global knowledge of the connectivity graph and are not decentralized. Thus, those algorithms are not viable solutions for a P2P application. A viable solution must comply two main requirements of P2P applications:

Requirement 4.1 (Decentralized and Autonomous) – Since P2P applications try not to rely on a central entity, e. g., a central server, or, other special hardware, P2P applications require a solution that works decentralized and autonomously on each node in the overlay.

Most P2P applications are scalable because they use local resources provided by the nodes itself. Thus, it is important that the solution detecting connectivity domains does not reduce the scalability of an existing P2P application:

Requirement 4.2 (Scalability) – For reasons of scalability the solution can only use a limited amount of local resources of a node, e. g., memory usage, bandwidth consumption, or, computational resources. Limited means, the usage must either be limited by a constant or grow moderately with the number of nodes, e. g., within $O(\log N)$ when N is the number of nodes.

These requirements reduce the solution space. However, possible real-world scenarios allow several assumptions that reduce the problem space as well.

Reducing the Problem Space

The scenario of P2P applications allows to state several assumptions that reduce the problem space. First, it is obviously highly improbable that the connectivity graph of an overlay of size N comprises the theoretically maximum number of cliques $3^{N/3}$. This assumption reduces the theoretical worst-case complexity of finding all maximal cliques magnitudes below $O(3^{N/3})$.

Second, in real-life networks usually only a few routers connect a larger number of end-systems. This means, real-life networks have a low *relay ratio* and allows to claim the following assumption:

Assumption 4.3 (Relay ratio) – The number of nodes attached to a single network is larger than the number of relays connected to several networks. In the formalism of the connectivity domains, this means that for two connectivity domains $\mathcal{C}_1, \mathcal{C}_2 \in \mathbf{C}$, it holds $|\mathcal{C}_i \backslash (\mathcal{C}_1 \cap \mathcal{C}_2)| \gg |\mathcal{C}_1 \cap \mathcal{C}_2|$, for $i \in \{1, 2\}$. Furthermore, the number of connectivity domains is finite and small compared to the number of all nodes, i. e., $|\mathbf{C}| \ll |\mathcal{N}|$.

In the analysis in Section 4.3, this assumption is used to show the feasibility of relay detection.

Finally, if nodes migrate from one network to another one, for example, by detaching a node from an IPv4 and re-attaching it to an IPv6 network, it usually includes a change of underlay addresses. Consequently, the node loses all overlay neighbors that are attached to the previous network. This allows to use the events of link failures on a node as an indicator for connectivity changes under the following assumption:

Assumption 4.4 (Node Migration) – If nodes migrate from one network to an other, links in the previous network fail and new links are established in the new network.

This assumption allows the Connectivity Measurement Protocol (CMP), cf. Section 4.5 to detect connectivity changes.

Summary

The previous requirements and assumption reduced both, the solution and the problem space. The most challenging question is, if the problem space is simple enough so a solution can be found. The next sections will discuss one feasible solution that fits the requirements and uses the assumptions to determine its feasibility.

4.2 Designing a Scalable and Decentralized Solution

Using the formal problem description, i. e., the requirements and assumptions of Section 4.1, a possible solution comprises the following steps:

1. **Build an unstructured overlay** $\mathcal{O} = (\mathcal{N}, \mathcal{E}')$ as sub-graph of the connectivity graph $\mathcal{G}$ with node degree bound by some $k \in \mathbb{N}$, i. e., $\mathcal{E}' \subset \mathcal{E}$ and $|\{e \in \mathcal{E}' : n \in e\}| \leq k$ for all $n \in \mathcal{N}$. Overlay neighbors in $\mathcal{O}$ can be selected by random sampling from the connectivity graph. In practice this is done using peer sampling [50] or random walks [4] in a scalable, and self-organizing manner, i. e., fulfilling the requirements in Section 4.1.3. Furthermore, a unstructured overlay commonly does not require the underlay to have transitive connectivity, cf., Section 2.2. The unstructured overlay serves as helper to detect connectivity.
2. **Detect relays** within the unstructured overlay $\mathcal{O}$. Thus, a mechanism is needed to detect relays in the overlay. This step is motivated from the key insight that, if all relays are detected, and non-relay nodes know which of their

overlay neighbors are relays, they know that all remaining overlay neighbors are in the same connectivity domain.

3. **Reach an agreement on a globally unique connectivity domain identifier (CID)** among the non-relay nodes implying that these nodes are in the same connectivity domain.

With this solution each node knows if it is a relay and CIDs of the connectivity domain(s). Furthermore, each non-relay can check by comparison of the CID if it is in the same connectivity domain.

This solution is not yet a protocol that solves the problem. Before the Connectivity Measurement Protocol (CMP) is derived, appropriate algorithms for the two main sub-problems, namely, relay detection and agreeing on a CID per connectivity domain, need to be found. The following sections present algorithms that solve these issues under the given requirements and assumptions. Additionally, the sections provide worst-case bounds for each sub-problem where applicable.

4.3 Relay Detection

Informally, if a node wants to determine if it is a relay, the node must check if one pair of nodes it can communicate directly with that cannot communicate directly with each other. Formally, a node x has to check if there is a pair of potential overlay neighbors a, b in the connectivity graph $\mathcal{G}$ that cannot establish a link, i. e., check if

$$\exists a, b \in \Gamma(x; \mathcal{G}) : c(a, x) \wedge c(x, b) \not\Rightarrow c(a, b)$$

as this would violate the transitive connectivity condition. Hence, the node is a relay for at least one pair of nodes. Depending on the number of nodes in the overlay, a node may have millions of nodes as potential overlay neighbors in the connectivity graph. As every pair needs to be considered this leads to a quadratic complexity with subject to the number of nodes in the overlay per node. This makes relay detection challenging.

The solution introduced in Section 4.2 already narrows the problem down by only considering a limited set of overlay neighbors $\Gamma(x; \mathcal{O})$ in the unstructured overlay $\mathcal{O}$ for relay detection. Thus, each node x needs only $O(k'^2) = O(1)$ checks because the number of actual overlay neighbors $k' = |\Gamma(x; \mathcal{O})|$ is limited; however, this also reduces the accuracy of the detection, i. e., a node may not be able to detect if it is a relay.

To shed light on this issue, the following sections study the feasibility of detecting relays when considering the assumptions and requirements from Section 4.1, in conjunction with the solution in Section 4.2. The study is threefold. First, Section 4.3.1 presents a technical tool called "triangle check" to probe if a pair of overlay neighbors of a node can establish a link. Section 4.3.3 describes the worst-case for relay detection. Then, Section 4.3.4 and Section 4.3.5 discuss two approaches to determine the required number of triangle checks by considering the worst case:

– First, the *random sampling approach* considers triangle checks between a randomly sampled pair of overlay neighbors, i. e., $a, b \in \Gamma(x; \mathcal{G})$ for each triangle

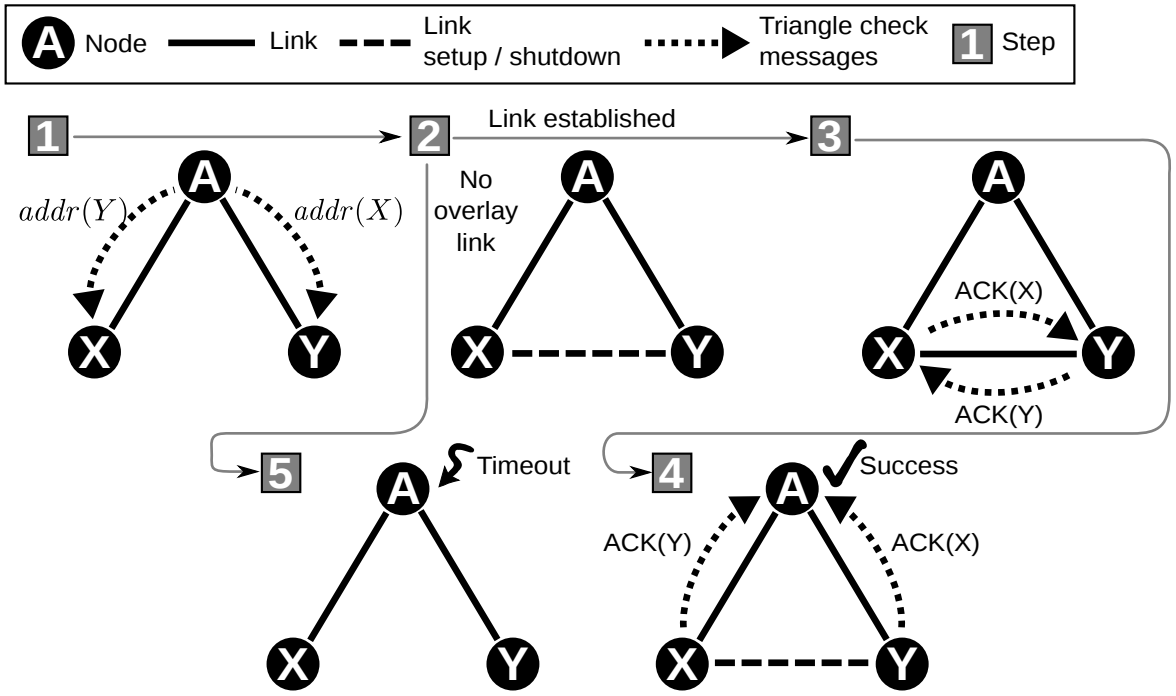


Figure 4.4 – Triangle check of node A between overlay neighbors X and Y

check. Thus, this approach is independent from the unstructured overlay $\mathcal{O}$ and discusses the general relay detection in the connectivity graph $\mathcal{G}$. This is important, since the unstructured overlay is built using random sampling.

– Second, the *unstructured overlay approach* considers the actual set $A \subset \Gamma(x; \mathcal{G})$, $|A| \leq k$ of overlay neighbors sampled randomly from the connectivity graph $\mathcal{G}$ for relay detection.

Finally, Section 4.3.6 summarizes the results of this study.

4.3.1 Triangle Checks

It is essential to check if a pair of overlay neighbors of a node can communicate *directly* to enable relay detection on a node. When considering the formal problem statement presented in Section 4.1.2, this is equivalent to the evaluation of the connectivity function $c(a, b)$ between two overlay neighbors $a, b \in \Gamma(x; \mathcal{O})$ of node x. The feasibility of this solution relies on one idea of an implementation. This section sketches such an implementation called *triangle check*. Whenever a triangle check fails a node knows it is a relay for at least two overlay neighbors.

Figure 4.4 shows the steps of a triangle check issued by node A between the overlay neighbors X and Y in detail. The figure shows a sequence of 5 steps 1 to 5. In the first step 1, node A sends messages containing the other overlay neighbors' underlay addresses to node X and Y, i. e., underlay address $addr(Y)$ to node X and $addr(X)$ to node Y respectively. Then, node A waits for acknowledgements from node X and Y to indicate that the check was successful.

If nodes X and Y receive node A's message in step 2 both try to establish a link using the underlay addresses contained in the messages. If the link has been

Notation	Description
$\mathcal{G} := (\mathcal{N}, \mathcal{E})$	Connectivity graph
$\mathcal{O} := (\mathcal{N}, \mathcal{E}')$	Unstructured overlay (graph)
$N := \lvert\mathcal{N}\rvert$	Overlay size
k	Max. no. of overlay neighbors
$A := \Gamma(x; \mathcal{O}), k' := \lvert A\rvert$	Set of overlay neighbors A of node x in the unstructured overlay k' denotes the actual number of overlay neighbors of node x
N_C	Number of connectivity domains
$c(a, b) \rightarrow \{true, false\}$	Connectivity function/triangle check between nodes a, b
$\mathcal{C}_1, \ldots, \mathcal{C}_{N_C} \subseteq \mathcal{N}$	Connectivity domains
$\mathbf{C} := \{\mathcal{C}_1, \ldots, \mathcal{C}_{N_C}\}$	Set of all connectivity domains
$\mathbf{C}(x) := (\mathcal{C}_1, \ldots, \mathcal{C}_n)$	Sequence of connectivity domains comprising node x
$\Theta(x) := (\theta_1, \ldots, \theta_n)$	Probability θ_i that a node x samples an overlay neighbor from the connectivity domain $\mathcal{C}_i$ with $i = 1 \ldots n$ and $(\mathcal{C}_i)_{i=1\ldots n} = \mathbf{C}(x)$
d	Distribution of non-relay nodes
r	Relay ratio

Table 4.1 – Summary of notations.

established successfully, nodes X and Y exchange acknowledgement messages ACK(X) and ACK(Y) in step 3. Finally, nodes X and Y forward these acknowledgements to A to inform A that the triangle check was successful in step 4. If the overlay link cannot be established in step 2, node A runs into a timeout and assumes that the triangle check has failed in step 5. To ensure that the triangle check did not fail due to of messages loss, congestion, or other reasons, node A repeats the check several times. Only when all repetitions have failed, node A assumes that it is in fact a relay for overlay neighbors X and Y.

4.3.2 Notations and Model for Random Sampling

The solution sketched in Section 4.2 uses random sampling to find overlay neighbors for a node. Hence, for the discussion of the feasibility of relay detection a definition of random sampling is necessary. For this reason, let

$$\mathbf{C}(x) := (\mathcal{C}_i)_{i=1\ldots n} \text{ with } n \in \mathbb{N}, 1 \leq n \leq N_C \tag{4.1}$$

define the sequence of connectivity domains comprising node x. Then, each θ_i of

$$\Theta(x) := (\theta_i)_{i=1\ldots n} \text{ with } \theta_i \in (0, 1], n := \lvert\mathbf{C}(x)\rvert, \sum_{i=1}^{n} \theta_i \stackrel{!}{=} 1 \tag{4.2}$$

denotes the probability that a node x samples an overlay neighbor from connectivity domain $\mathcal{C}_i$ of $\mathbf{C}(x)$. Thus, $\Theta(x)$ characterizes the random sampling of node x's overlay neighbors from the connectivity graph. Two distributions of $\Theta(x) = (\theta_i)_{i=1\ldots n}$ for random sampling are considered:

- On the one hand, the *uniform*-Θ-distribution describes the *uniform* random sampling of nodes from all connectivity domains $\mathbf{C}(x)$ of a node x, i. e., $\theta_1 = \ldots = \theta_n = \frac{1}{n}$.
- On the other hand, the *non-uniform*-Θ-distribution describes the *non-uniform* random sampling with a probability proportional to the number of nodes in a connectivity domain, i. e., $\theta_1 = \frac{|\mathcal{C}_1|}{N_x}, \ldots, \theta_n = \frac{|\mathcal{C}_n|}{N_x}$, with $N_x = \sum_{X \in \mathbf{C}(x)} |X|$ being sum of the number of nodes of all connectivity domains comprising node x. This includes relays, thus, N_x can be greater than the number of nodes N.

The following describes the worst-case scenario and sheds light on the number of triangle checks required using the random sampling approach. For convenience, Table 4.1 summarizes the notations this section uses.

4.3.3 Worst-Case Scenario

Intuitively, relay detection on a node gets harder in terms of the required number of triangle checks the less connectivity domains comprise the node. This makes relay detection on a node in two connectivity domains the worst case. The following backs this intuition formally.

Let x be the only node present in $n \geq 2$ connectivity domains $\mathbf{C}(x)$ and $\Theta(x) := (\theta_i)_{i=1\ldots n}$ an arbitrary Θ-distribution. Furthermore, let $p := \max \Theta(x)$ be the maximum probability of random-sampling an overlay neighbor from a connectivity domain. Then, the maximum probability of random-sampling a pair of overlay neighbors from disjoint connectivity domains is $(1 - p^2)$. A triangle check fails with this probability and the node detects it is a relay. p decreases monotonically with a growing number of connectivity domains n because of $\sum_{i=1}^{n} \theta_i \stackrel{!}{=} 1$, and, $\theta_i \neq 0$, cf., term (4.2). Thus $(1 - p^2)$ grows monotonically with growing number of connectivity domains n and makes relay detection on the node more likely. Consequently, relay detection on a node in two connectivity domains is the worst case.

Scenario with Two Connectivity Domains

Motivated by the worst-case of relay detection, the following describes a scenario comprising a set of nodes $\mathcal{N}$ of size $N := |\mathcal{N}|$ which are at least in one of the two connectivity domains $\mathcal{C}_1$ and $\mathcal{C}_2$. The set of relays $\mathcal{R} := \mathcal{C}_1 \cap \mathcal{C}_2$ denotes the nodes in the intersection of both connectivity domains. For further discussion the goal is to describe this scenario with relative parameters. These relative parameters describe the node distribution d among $\mathcal{C}_1, \mathcal{C}_2$ and the number of relays in $\mathcal{R}$ using the *relay ratio* r. The *relay ratio* defines the fraction of relays in connectivity domains $\mathcal{C}_1$ and $\mathcal{C}_2$:

$$|\mathcal{R}| = \lceil r \cdot N \rceil, r \in (0, 1), 1 \leq |\mathcal{R}| \leq (N - 2) \tag{4.3}$$

The *node distribution*[2] $d \in \left(0, \frac{1}{2}\right]$ defines the distribution of non-relay nodes among the connectivity domains $\mathcal{C}_1$ and $\mathcal{C}_2$:

$$|\mathcal{C}_1 \backslash \mathcal{R}| = \lceil d \cdot (1-r) \cdot N \rceil \tag{4.4}$$
$$|\mathcal{C}_2 \backslash \mathcal{R}| = \lceil (1-d) \cdot (1-r) \cdot N \rceil \tag{4.5}$$

This means, e. g., if d equals $\frac{1}{2}$ the two connectivity domains comprise an equal number of non-relay nodes. The smaller d, the less non-relay nodes are in $\mathcal{C}_1$ and the more are in $\mathcal{C}_2$. Adding up all relay and non-relay nodes yields the number of all nodes N, more precisely:

$$\begin{aligned} & |\mathcal{C}_1 \backslash \mathcal{R}| + |\mathcal{C}_2 \backslash \mathcal{R}| + |\mathcal{R}| && = N \\ \Leftrightarrow \quad & \lceil d \cdot (1-r) \cdot N \rceil + \lceil (1-d) \cdot (1-r) \cdot N \rceil + \lceil r \cdot N \rceil && = N \\ \Leftrightarrow \quad & N && = N \end{aligned}$$

This confirms that the node distribution d and relay ratio r are in fact *relative* parameters to describe the scenario of two connectivity domains independently from the *absolute* number of nodes N.

According to Assumption 4.3 in Section 4.1.3 there must be more non-relay nodes than relays. This means, the number of relays in $\mathcal{R}$ must be *significantly* lower than the number of non-relay nodes in $\mathcal{C}_1 \backslash \mathcal{R}$. More precisely,

$$|\mathcal{R}| \ll |\mathcal{C}_1 \backslash \mathcal{R}| \Leftrightarrow r \cdot N \ll d \cdot (1-r) \cdot N \Leftrightarrow r \ll d \cdot (1-r)$$

which allows the derivation of the constraints

$$r \leq \frac{d}{1+d} \quad \text{and} \quad d \geq \frac{r}{1-r} \tag{4.6}$$

of the node distribution d and relay ratio r.

Figure 4.5 illustrates the relay detection on node A in a scenario with a node distribution of $d = \frac{1}{2}$ in the connectivity domains $\mathcal{C}_1$ and $\mathcal{C}_2$, and an extraordinary high relay ratio of $r = \frac{1}{3}$. Node A performs four triangle checks between pairs of its overlay neighbors, X, U, V, and Y. The problem of relay detection on node A is, that some of the node's overlay neighbors are relays. Therefore, triangle checks succeed across connectivity domain boundaries. This makes the relay detection complex. Only the triangle check between nodes X and Y fails in the example. In this case, both overlay neighbors are not relays and in different connectivity domains.

4.3.4 Relay Detection using Random Sampling

The example in Figure 4.5 shows that a triangle check initiated by a node $x \in \mathcal{R}$ fails if the node chooses two overlay neighbors in different connectivity domains which are not in the set of relays. Without loss of generality, let the random

[2] Without loss of generality, let connectivity domain $\mathcal{C}_1$ comprise less nodes than $\mathcal{C}_2$, i. e., $|\mathcal{C}_1| < |\mathcal{C}_2|$.

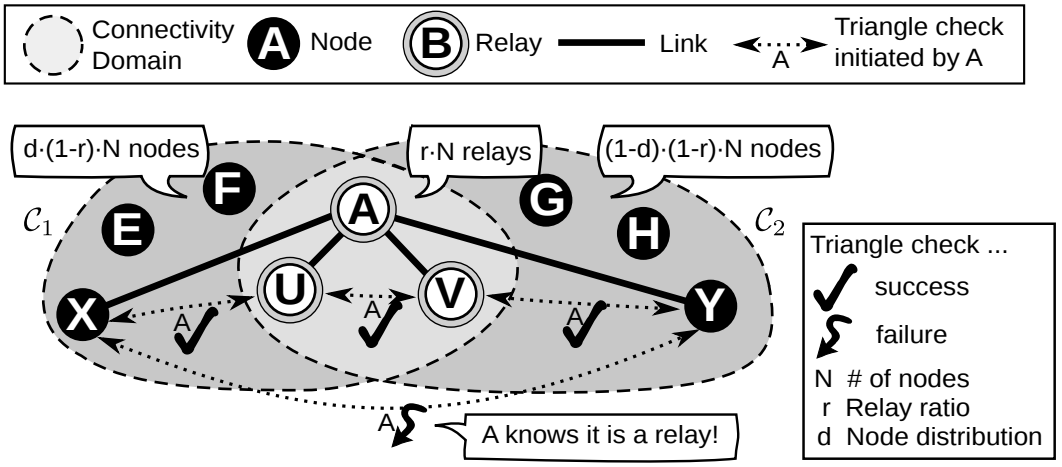


Figure 4.5 – Exemplary scenario for relay detection on node A with an extraordinary high relay ratio of $r = \frac{1}{3}$ and a node distribution of $d = \frac{1}{2}$

sampling of a node $x \in \mathcal{R}$ obey the non-uniform-Θ-distribution $\Theta(x) = (\theta_1, \theta_2)$ for the connectivity domains $\mathbf{C}(x) = (\mathcal{C}_1, \mathcal{C}_2)$ parametrized with d and r:

$$\theta_1 := \frac{|\mathcal{C}_1|}{N_x} = \frac{N \cdot (d \cdot (1-r) + r)}{N_x}$$
$$\theta_2 := \frac{|\mathcal{C}_2|}{N_x} = \frac{N \cdot ((1-d) \cdot (1-r) + r)}{N_x}$$
$$N_x := |\mathcal{C}_1| + |\mathcal{C}_2| = N \cdot (1+r)$$

Then, the probability that a triangle check between two randomly sampled overlay neighbors of node x fails can be approximated as function of r and d:

$$P_{\text{fail}}(r,d) := 2 \cdot \frac{|\mathcal{C}_1 \backslash \mathcal{R}|}{N} \cdot \frac{|\mathcal{C}_2 \backslash \mathcal{R}|}{N} = 2 \cdot d \cdot (1-d) \cdot (1-r)^2. \tag{4.7}$$

The term comprises the probability that two randomly sampled overlay neighbors are not in the set of relays $(1-r)^2$ and in different domains $2 \cdot (1-d) \cdot d$. Additionally, the term allows the derivation of the probability that *at least* one of Δ_{min} triangle checks will fail. The term represents only a approximation. The real probability could only be calculated using the absolute number of nodes:

$$P'_{\text{fail}} := \frac{|\mathcal{C}_1 \backslash \mathcal{R}| \cdot |\mathcal{C}_2 \backslash \mathcal{R}|}{\binom{N-1}{2}}.$$

However, $P_{\text{fail}}(r,d)$ is a lower bound as $P'_{\text{fail}}(r,d) := \epsilon \cdot P_{\text{fail}}(r,d)$ with $\epsilon := \frac{N^2}{2 \cdot \binom{N-1}{2}} \to 1$ when $N \to \infty$ and $\epsilon \leq 1$.

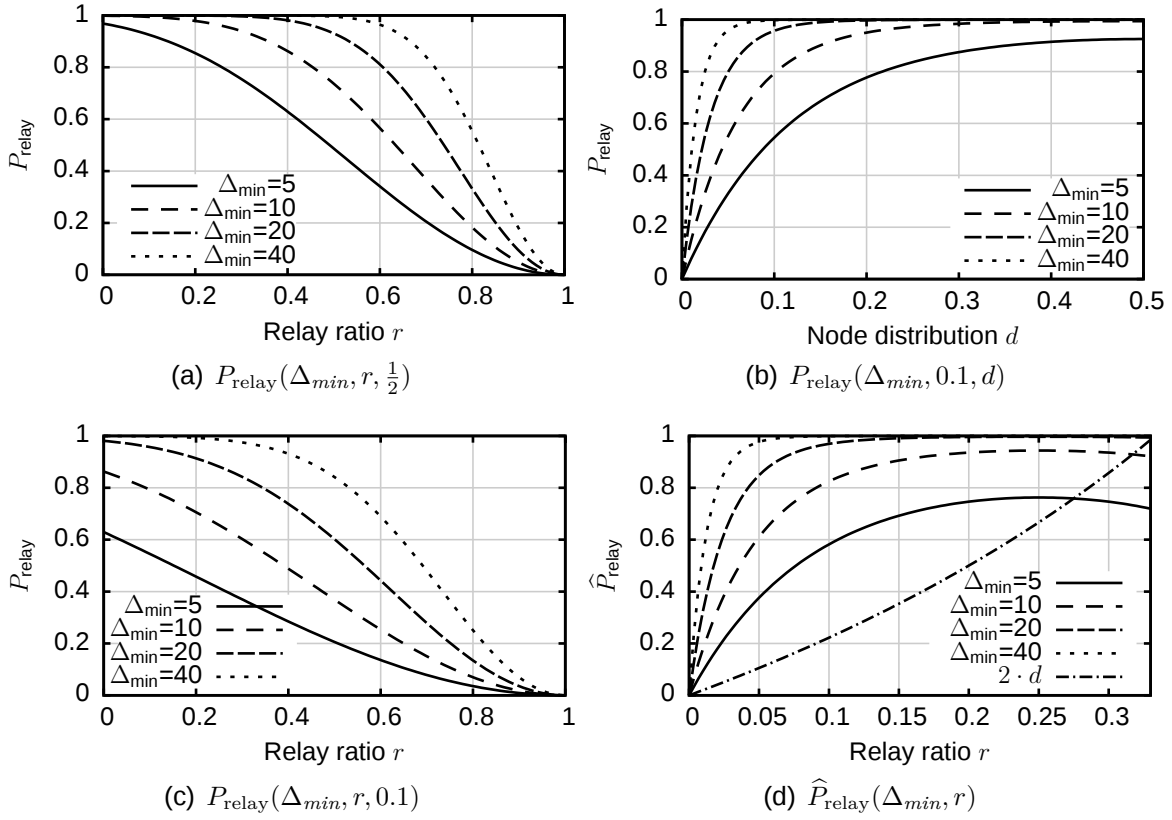


Figure 4.6 – Probability of a valid relay detection on a node: P_{relay} and $\widehat{P}_{\text{relay}}$.

$P_{\text{fail}}(r, d)$ allows to calculate the probability of relay detection a node:

$$P_{\text{relay}}(\Delta_{min}, r, d) := 1 - (1 - P_{\text{fail}}(r, d))^{\Delta_{min}} \tag{4.8}$$

By applying the constraints of (4.6), derived from Assumption 4.3, to (4.7) leads to a lower bound of P_{relay}:

$$\widehat{P}_{\text{relay}}(\Delta_{min}, r) := P_{\text{relay}}(\Delta_{min}, r, \frac{r}{1-r}) \tag{4.9}$$

$$\Leftrightarrow \quad \widehat{P}_{\text{relay}}(\Delta_{min}, r) := 1 - \left(4 \cdot r^2 - 2 \cdot r + 1\right)^{\Delta_{min}} \text{ with } r \in (0, \frac{1}{3}) \tag{4.10}$$

$\widehat{P}_{\text{relay}}$ represents the worst-case probability in terms of the node distribution d and Assumption 4.3. More precisely, the set of relays $\mathcal{R}$ comprises the same number of relays as there are non-relay nodes in the smallest of the connectivity domains $\mathcal{C}_1$ or $\mathcal{C}_2$. This results in an imbalance in the number of non-relay nodes in the connectivity domains $\mathcal{C}_1$ and $\mathcal{C}_2$.

Figure 4.6 shows several curves of the probabilities of relay detection as a function of the relay ratio r and node distribution d for an increasing amount of triangle checks $\Delta_{min} \in \{5, 10, 20, 40\}$ using P_{relay} and $\widehat{P}_{\text{relay}}$.

Figure 4.6(a) shows P_{relay} as function of the relay ratio r with an even node distribution of $d = \frac{1}{2}$. This is equivalent to a random sampling that obeys the

uniform-Θ-distribution. The plot suggests that $10 \leq \Delta_{min} \leq 20$ triangle checks are sufficient for relay detection on a node in a scenario with $r \leq 20\%$ relays with high probability. This relay ratio is way beyond what was expected in Assumption 4.3.

A difficulty of relay detection using random sampling, is the impact of a small node distribution d which results in a non-uniform-Θ-distribution. Figure 4.6(b) shows P_{relay} as function of the node distribution d with a relay ratio $r = 0.1$. Even with a high number of triangle checks the probability of relay detection drops to zero when the node distribution is unbalanced. This is natural, an unbalanced node distribution makes it more likely that a node samples two overlay neighbors from the larger connectivity domain $\mathcal{C}_2$. Figure 4.6(c) suggests the same behaviour. It shows P_{relay} as function of the relay ratio r with a fixed node distribution $d = 0.1$. Only a high number of triangle checks $\Delta_{min} \geq 40$ makes the relay detection feasible given a node distribution of 0.1.

Finally, figure 4.6(d) shows the worst-case assumption on the node distribution by illustrating $\widehat{P}_{\text{relay}}$ as function of the relay ratio r. Only a relay ratio r above 0.1, a resulting worst-case node distribution $d \geq 0.1$, and at least $\Delta_{min} \geq 20$ triangle checks, make the relay detection feasible.

In summary, the study of the random-sampling approach, gave the following insight:

- The worst-case scenario is the relay detection on a node in two connectivity domains. The two connectivity domains can be modelled using two parameters, the relay ratio r and the node distribution d which are independent of the absolute number of nodes N.
- The higher the relay ratio $r > 0$ and/or the more unbalanced the node distribution $d < 0.5$, the harder the relay detection.
- A large number of triangle checks $\Delta_{min} \geq 10$ can compensate a high relay ratio $r > 0$, and an unbalanced node distribution $d < 0.5$.

The random-sampling approach does not consider the unstructured overlay. It just uses random-sampling for relay detection. Thus, the approach is more generic.

4.3.5 Relay Detection using the Unstructured Overlay

This section discusses relay detection by using the overlay neighbors in the unstructured overlay. For this reason, let $\mathcal{A} := (a_0, \ldots, a_{k'-1})^T, k' \leq k$ be the vector of random-sampled overlay neighbors $a_i \in \Gamma(x; \mathcal{O})$ of a node x in the unstructured overlay. Furthermore, using the scenario of two connectivity domains, let $\mathcal{A}$ comprise $c_j \in \mathbb{N}$ overlay neighbors from $\mathcal{C}_j \backslash \mathcal{R}, j \in \{1, 2\}$ and c_r overlay neighbors from $\mathcal{R}$ such that $c_1 + c_2 + c_r = k'$.

Assume that node $x \in \mathcal{R}$ is a relay. Then, to describe the worst case, i. e., maximal imbalance *and* maximal relay ratio, c_2 and c_r are calculated as follows: first, let c_1 denote the number of overlay neighbors in $\mathcal{C}_1 \backslash \mathcal{R}$. Second, assume that the number of relays equal the number of non-relays in $\mathcal{C}_1$. According to Assumption 4.3 this is unlikely, but with respect to the previous insight makes relay detection difficult. Consequently, since node x is a relay, let $c_r := c_1 - 1$

denote the number of overlay neighbors in $\mathcal{R}$. Finally, the rest of the overlay neighbors must be in $\mathcal{C}_2 \backslash \mathcal{R}$, thus, $c_2 := k' - 2 \cdot c_1 + 1$.

For relay detection on node x three strategies are considered:

Random strategy:
The node performs Δ_{min} triangle checks between randomly chosen overlay neighbors a_i, a_j with $i \neq j$.

Overlapping strategy:
The node performs $\Delta_{min} = k'$ triangle checks between a_i, and $a_{(i+1) \bmod k'}$, $i = 0 \ldots (\Delta_{min} - 1)$.

Non-overlapping strategy:
The node performs $\Delta_{min} = \lfloor \frac{k'}{2} \rfloor$ triangle checks between $a_{(2 \cdot i)}$, and $a_{(2 \cdot i+1) \bmod k'}$, $i = 0 \ldots (\Delta_{min} - 1)$.

For evaluation of those strategies a simulation sheds light on the probability of relay detection. The simulation determines the probability of the overlapping and non-overlapping strategy by counting the permutations of $\mathcal{A}$ that lead to a triangle check failure divided by the number of all permutations Ω. The probability when using the random strategy is calculated using

$$P'_{\text{relay}} := 1 - \left(1 - 2 \cdot \frac{c_1}{k'} \cdot \frac{c_2}{k'}\right)^{\Delta_{min}}$$

according to (4.8). Figure 4.7 shows the simulation results for the probability of a relay detection on a node in the unstructured overlay as function of varying parameters $k' \in \{10, 16, 20\}$ and c_1. The reason for $k' \leq 20$ is because the number of permutations grows with the factorial of k'; more than $k' > 20$ overlay neighbors takes too much time to simulate.

Figures 4.7(a), and 4.7(b) show the probability of relay detection for $k' = 10$ and $k' = 16$ as function of c_1. Both figures show the superiority of the overlapping strategy. It is close to 1.0 even when the number of relays c_r exceeds the number of non-relays c_2. The random and non-overlapping strategies show an nearly proportional probability; however, the non-overlapping strategy uses half of the triangle checks, thus, has a slightly lower probability. Both hardly come close to 1.0 with a low number of nodes c_1 while the overlapping strategy easily reaches and sustains a probability of 1.0. A noteworthy abnormality is the initial sharp drop in the curve of the non-overlapping strategy. This follows when $c_1 = 1$, then, c_r is zero, and $c_2 = k' - 1$, hence, in any permutation of $\mathcal{A}$ a non-overlapping pair of overlay neighbors $a_{2 \cdot i} \in \mathcal{C}_1 \backslash \mathcal{R}$, and $a_{(2 \cdot i+1) \bmod k'} \in \mathcal{C}_2 \backslash \mathcal{R}$, $i = 0 \ldots (\Delta_{min} - 1)$ can be found which causes the triangle check to fail.

Figure 4.7(c) shows the probability of relay detection for $k' = 20$ and Figure 4.7(d) illustrates the values of c_r and c_2 as function of c_1. Figure 4.7(c) acknowledges the trend shown by the figures 4.7(a) and 4.7(b). The overlapping strategy provides the best probability of relay detection. Even when Assumption 4.3 is violated by a larger number of relays than non-relay nodes, i. e., c_r exceeds c_2, the overlapping strategy allows relay detection with a probability higher than 75%. This is also the case with a lower number of overlay neighbors k' when comparing

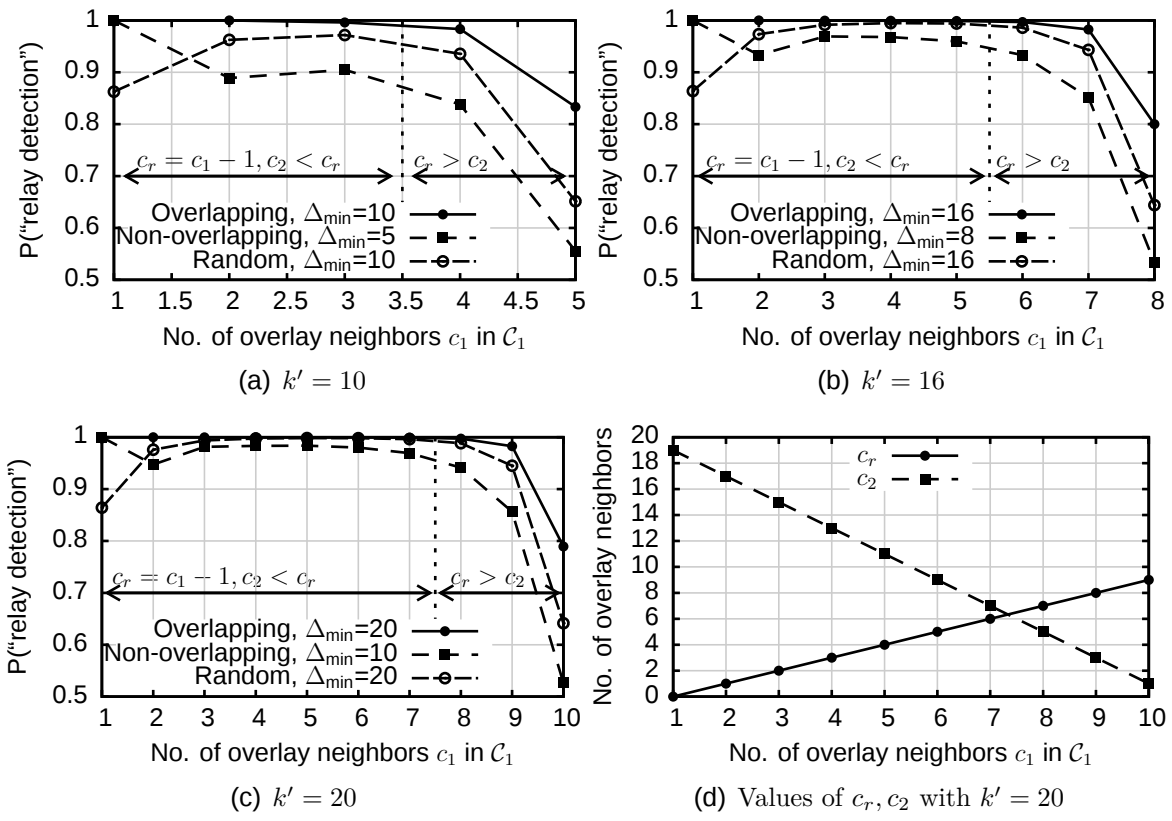


Figure 4.7 – Probability of a relay detection on a node in the unstructured overlay.

the curves from the other figures. In conclusion, among the considered strategies, the overlapping strategy outperforms, both, random, and pairwise non-overlapping strategy.

Naturally, relay detection on nodes in the unstructured overlay can only detect relays that have *at least* one overlay neighbor in each connectivity domain $\mathcal{C}_i \backslash \mathcal{R}, i \in \{1, 2\}$. Additionally, the number of overlay neighbors k' of a node x also limits the number of detectable connectivity domains. Hence, the unstructured overlay approach only provides a weak relay detection—limited to the unstructured overlay. In contrast, the random-sampling approach provides a statement that reflects the feasibility of relay detection in the connectivity graph $\mathcal{G}$ in general.

4.3.6 Summary

This section discussed the feasibility of detecting relays with the constraints given in Section 4.1. For this purpose, worst-case scenarios have been considered for different relay detection approaches. This results in the insights as follows:

1. A node in two connectivity domains is a worst-case for relay detection (cf. Section 4.3.3).
2. Unbalanced connectivity domains and a high relay ratio make relay detection difficult in the random-sampling and unstructured overlay approach.
3. Yet, relay detection in the unstructured overlay is feasible under the assumptions from Section 4.1 and even beyond (cf. Section 4.3.5).

4. A minimum of $10 \leq \Delta_{min} \leq 20$ using the overlapping triangle check strategy is adequate for a relay detection in the unstructured overlay. This requires a node to have $10 \leq k' \leq 20$ overlay neighbors if possible. Furthermore, the choice of Δ_{min} and k' also provides a good chance of relay detection in the connectivity graph $\mathcal{G}$ (cf. Section 4.3.4).

After the relay detection, non-relays must agree on an identifier for each domain. The next section discusses several random number agreement schemes for this purpose.

4.4 Connectivity Identifier (CID) Agreement

After the nodes performed the relay detection, the non-relay nodes know that they are in a connectivity domain with their overlay neighbors. To identify a connectivity domain, non-relays agree on a random number, the *connectivity identifier* (*CID*), with their overlay neighbors. At the same time relays wait until their non-relay overlay neighbors have agreed on CIDs. This way, they become aware of the set of connectivity domains they are relay for.

This section discusses the requirements, a possible random number agreement to agree on a CID and provides a feasibility study. For this purpose, the following considers an unstructured overlay graph comprising the non-relays in a connectivity domain. Formally, let $\mathcal{O}' := (\mathcal{N}', \mathcal{E}'')$ comprise the non-relay nodes of the unstructured overlay that are in a connectivity domain $\mathcal{N}' := \mathcal{C} \backslash \mathcal{R}$ and $\mathcal{E}'' := \{\{a, b\} \in \mathcal{E}' : a, b \in \mathcal{N}'\}$. The goal is to reach an CID agreement on all nodes in $\mathcal{O}'$.

4.4.1 Requirements and Related Work

Naturally, the random number agreement algorithm must comply with the requirements from Section 4.1.3. Hence, the CID agreement must be scalable and decentralized. Furthermore, it must use the unstructured overlay of the solution in Section 4.2. In addition to these requirements, the CID agreement has two additional requirements:

Requirement 4.5 (Leak robustness) – The CID agreement must be tolerant when weak relay detection fails on some nodes. In this case, nodes might agree on a CID which are in fact in several connectivity domains. Thus, the CID agreement has *leaks* to several connectivity domains. It is desirable that the CID agreement is robust against at least some of those leaks.

Requirement 4.6 (Agreement stability) – After the non-relay nodes agreed on a CID this agreement must be *stable*. This means, a single joining, leaving, or failing node must not be able to change the CID agreement. Ideally, only the majority of nodes should decide on a new CID agreement.

For finding a feasible random number agreement algorithm to agree on a CID several related work is considered. First, one way of agreeing on a CID is to elect a leader that chooses a CID. The very first point of reference is [41], which elects the node (or process) with the greatest identifier. Janson et. al [48] provides a discussion of the convergence of general leader election algorithms. Most of the

leader election algorithms assume a full mesh, i. e., all nodes link to all other nodes. This makes these approaches unattractive. Furthermore, electing a leader would introduce a single point of failure, i. e., the leader may select a CID that is not unique.

Since CMP uses a unstructured overlay, gossip-based consensus protocols become interesting in the design process. The basic idea of gossiping is inspired by the spread of epidemics discussed by Demers et. al [22] and rumor spreading theory [73, 54]. This led to a variety of gossip algorithms [11].

The random agreement is a consensus on a random number among a set of nodes. Generally, a consensus has certain bounds: Fischer et. al [31] show the impossibility of a consensus with only one single faulty process (or node). Newer work, i. e., [77], shed light on the impossibility of consensus when links fail between nodes. Thus, in CMP, if a node does not want to adapt a certain CID, there is no consensus on a CID among all nodes. This is a typical boundary.

4.4.2 Gossip-based Random Number Agreement

To fulfill the requirements from Section 4.4.1, the most appropriate group of protocols are gossip-based consensus algorithms [11]. These algorithms assume a unstructured overlay and use a gossiping protocol to achieve a consensus between nodes. It is known that gossiping protocols are scalable, decentralized, have good convergence time, and are well-understood theoretically [73]. Thus, fit the requirements from Section 4.1.3 with one exception: the most gossip-based consensus algorithms find a consensus on an average value. The CID agreement needs to agree on a random number. However, these algorithms provide an starting point for a gossip-based random number agreement algorithm.

The approach behind gossip-based algorithms is as follows: Each T_{round} seconds, or each *round*, a node exchanges a message of limited size comprising *gossip* with a set of randomly chosen overlay neighbors. This results in a constant "fan-out", i. e., number of messages sent, and constant "fan-in", i. e., number of messages received per node as the number of overlay neighbors $k' \leq k$ is limited. The contents of the gossip depends on the intention, e. g., finding a consensus or random agreement, of the gossip-based algorithm.

Algorithm 4.1 describes a gossip-based random number agreement on a node x with $k' := |\Gamma(x; \mathcal{O}')|$ overlay neighbors. In this case, the gossip exchanged by the algorithm is a random number. The algorithm first chooses a random number $\phi \in \mathbb{N}_p$. This random number is the node's first *proposal* for the random number agreement. To ensure that the proposal is unique with high probability, *rand*() returns a random number within the interval $\mathbb{N}_p := [1, 2^{128} - 1] \subset \mathbb{N}$, i. e., a 128-bit value. The value of zero denotes an *invalid* proposal. After that, the k'-tuple of received proposals $\mathcal{P} := (p_1, \ldots, p_{k'})$ is initialized with zeros. A specific element $p_i \in \mathbb{N}_p \cup \{0\}$ of the k'-tuple stores a received proposal from an overlay neighbor or has an invalid value. Then, the algorithm sets the running flag and proceeds with the first round in the while-loop.

In the loop, node x sends his proposal ϕ to all of its overlay neighbors. Consequently[3], in the next step, the node receives proposals and refreshes the proposals of the overlay neighbors in the k'-tuple $\mathcal{P}$ while waiting T_{round} seconds. After waiting, node x chooses a new proposal ϕ from the k'-tuple $\mathcal{P}$ using a selection function D and starts over.

The analysis of the algorithm's properties requires an global view on the proposals of the nodes $V(\mathcal{O}')$ in $\mathcal{O}'$. For this purpose, let $\Phi_i := \bigcup_{x \in V(\mathcal{O}')} \phi_x$ be the set of proposals of the nodes in $\mathcal{O}'$. ϕ_x denotes the proposal ϕ chosen by Algorithm 4.1 on node $x \in V(\mathcal{O}')$ after a maximum of $i \in \mathbb{N}$ rounds. Before the algorithm enters the main loop Φ_0 comprises $|V(\mathcal{O}'|)$ distinct proposals with high probability. When the nodes have found an agreement, $\Phi_{R_a} = \{p'\}$ contains exactly one random number p' after $R_a \in \mathbb{N}$ rounds. R_a or $R_a \cdot T_{round}$ are metrics of the *convergence time* of the random number agreement.

To reach a random number agreement, Algorithm 4.1 must use a suitable selection function $D : \mathbb{N}_p^{k'} \to \mathbb{N}_p$ which reduces the overlay-wide proposals in Φ_i to a single proposal after a finite number of rounds R_a. The next section discusses the properties of several selection functions.

4.4.3 Discussion of Selection Functions

The selection function $D : \mathbb{N}_p^{k'} \to \mathbb{N}_p$ selects a new proposal from proposals $\mathcal{P} \in {\mathbb{N}_p}^{k'}$ received from k' overlay neighbors of node x. To ease the understanding of the selection functions the following sections use the exemplary values of $\mathcal{P}$:

$$\mathcal{P}' := (75, 12, 45, 12, 45, 98).$$

Additionally, the definition of the selection functions uses the following notation:

```
/* Initialization: choose a random number proposal */
φ ← rand();
P ← (0, ..., 0);
running ← true;
/* Main loop: exchange proposals periodically */
while running = true :
    /* Send proposal to overlay neighbors */
    foreach overlay neighbor y ∈ Γ(x; O'): send(φ, y);
    /* Receive proposals from overlay neighbors during wait of T_round seconds */
    receive_and_wait(P,T_round);
    /* Select new proposal */
    φ ← D(P)
```

Algorithm 4.1 – Gossip-based random number agreement on node x with $k' := |\Gamma(x; \mathcal{O}')|$ overlay neighbors

[3]The overlay neighbors run the same algorithm.

Distinct proposals: The n_u-tuple $U^{\mathcal{P}} := (u_1, \ldots, u_{n_u}) \in \mathbb{N}_p^{n_u}$ with $n_u := \left|\bigcup_{i=1}^{k'} p_i\right|$ and $u_i \in \bigcup_{i=1}^{k'} p_i$ denotes the distinct proposals from $\mathcal{P}$ ordered decendingly by the frequency and by the value of the proposal to break ties. Thus, u_1 is one of the *most frequent* and u_{n_u} one of the *least frequent* proposals in $\mathcal{P}$. Example:

$$U^{\mathcal{P}'} := (45, 12, 98, 75).$$

Proposal frequency: $f^{\mathcal{P}}(p) \in \mathbb{N}, p \in \mathbb{N}_p$ denotes the frequency of a proposal p in $\mathcal{P}$. Example:

p	45	12	98	75
$f^{\mathcal{P}'}(p)$	2	2	1	1

Using these notations the following sections discuss several selection functions.

Maximum Selection Functions

A very obvious possibility to agree on a proposal, is to select the largest number[4] in $\mathcal{P}$. The maximum selection function D_{M} implements this behaviour:

$$D_{\mathrm{M}}(\mathcal{P}) := \max_{p \in \mathcal{P}} p \tag{4.11}$$

The algorithm's convergence time using this selection function is equivalent to rumor spreading [73] which is within $n_r \in O(\log N)$ (N is the number of nodes in $\mathcal{O}'$) rounds[5]. The disadvantage is, that the random number agreement always converges to the largest proposal. In consequence, the agreement loses its random quality.

To fix this, the maximum hash (MH) selection function D_{MH} selects the random number proposal having the maximum cryptographic hash value. This ensures that the resulting random number agreement does not lose its random quality:

$$D_{\mathrm{MH}}(\mathcal{P}) := \arg\max_{p \in \mathcal{P}} H(p) \tag{4.12}$$

However, if one node proposes a random number that has a greater hash value than the current agreement, all nodes will agree on this proposal. Therefore, a single node can change the agreement. This violates the stability requirement.

To introduce stability after the nodes have agreed on a proposal, the majority maximum hash (MMH) selection function D_{MMH} uses an absolute majority vote. It selects the random number that 50% of overlay neighbors propose or the random number with the greatest hash to break ties:

$$D_{\mathrm{MMH}}(\mathcal{P}) := \begin{cases} u_1, & \text{if } f^{\mathcal{P}}(u_1) \geq \frac{1}{2} \cdot k' \\ \arg\max_{x \in \mathcal{P}} H(x), & \text{otherwise} \end{cases} \tag{4.13}$$

[4] This is similar to the leader election algorithm [41] or bully algorithm

[5] Actually, this complexity can also be deduced from considering flooding in a random graph. Because random graphs have a logarithmic diameter, gossip-based flooding takes a logarithmic number of rounds as well.

As defined before, u_1 is the most frequent proposal in $\mathcal{P}$. Thus, random number u_1 is selected if at least $\frac{k'}{2}$ overlay neighbors propose u_1.

Using D_{MMH} the agreement on a CID can be realized fast due to the expected logarithmic number of rounds needed to converge, and stable due to the consideration of the absolute majority vote in the selection function. However, D_{MMH} is not robust against leaks. If one relay is not detected, all nodes in all connectivity domains comprising the undetected relay will likely agree on the same CID.

The main reason for D_{MMH}'s failure in the presence of leaks is that it selects a proposal by interpreting its value, i. e., comparing that the hash. The next section presents the random selection functions which change this behaviour.

Random Selection Functions

In contrast to the maximum selection functions the random selection functions do not interpret the value of a proposal. This makes it less likely that a single proposal has an influence on the random number agreement. The simplest random selection function is given by D_{R} which selects one proposal from $\mathcal{P}$ randomly. Thus, it does not make any decision based on the values of the proposals. This prevents that a particular proposal has a global impact on the agreement—each proposal is treated equally:

$$D_{\text{R}}(\mathcal{P}) := p_x \in \mathcal{P}, x \in \{1, \ldots, k'\} \text{ chosen randomly} \tag{4.14}$$

Intuitively, this function will cause that nodes agree extremely slow, or, worse, never on a single random number. To accelerate the agreement and to provide stability for the agreement the relative majority random (RMR) selection function D_{RMR} combines the random selection function with a relative majority vote:

$$D_{\text{RMR}}(\mathcal{P}) := \begin{cases} u_1, & \text{if } |U^{\mathcal{P}}| \geq 2 \wedge f^{\mathcal{P}}(u_1) > f^{\mathcal{P}}(u_2) \\ D_{\text{R}}(\mathcal{P}), & \text{otherwise} \end{cases} \tag{4.15}$$

If one proposal in $\mathcal{P}$ is more frequent than others, i. e., $f^{\mathcal{P}}(u_1) > f^{\mathcal{P}}(u_2)$, then D_{RMR} selects this proposal. Otherwise, D_{RMR} selects a random number proposal randomly. Like in D_{MMH} the majority vote provides the required stability for the agreement. Furthermore, if one proposal has the relative majority, it is likely that other nodes choose this proposal which accelerates the agreement.

The goal of D_{R} is reducing the impact of leaks in the random number agreement—not preventing it. Furthermore, the relative majority vote of D_{RMR} makes an agreement more likely and stable. The main problem is that the number of rounds R_a needed to agree on a random number is unclear. Related work does not give a answer to this task. In the next section Algorithm 4.1 using D_{R} is analysed to back the intuition with facts.

4.4.4 Analysis of the Random Selection Function

The goal in this section is to gain insight of the number of rounds R_a needed for Algorithm 4.1 with selection function D_{R} on N nodes in an graph $\mathcal{O}'$ until an agreement on a single random number is reached. The analysis is twofold. First,

Lemma 4.7 provides insights if $\mathcal{O}'$ is a cycle or path. Second, Theorem 4.8 extends the insight of Lemma 4.7 to random graphs.

All following considerations assume that Algorithm 4.1 operates synchronously on all nodes in $\mathcal{O}'$. This means all nodes initially propose a random number ϕ at the same time, therefore, $|\Phi_0|$ comprises N different proposals. Then, in each round, all nodes send their proposals ϕ to their overlay neighbors instantly and wait T_{round} seconds. Next, all nodes choose a new proposal ϕ randomly applying D_{R} on the received proposals and start over.

Lemma 4.7 (D_{R} on cycle and path graphs) – Let $\mathcal{O}'$ be a cycle or path graph comprising $N \geq 3$ nodes. Then at least a quarter of the initially distinct proposals are expected to get lost in the first round of Algorithm 4.1 using D_{R}, i. e., $|\Phi_1| \leq \frac{3}{4} \cdot |\Phi_0|$. Furthermore, the recurrence relation

$$\begin{aligned} T_0 &:= N \\ T_{i+1} &:= T_i - \tfrac{1}{4} \cdot \left(\tfrac{T_i}{N}\right)^3 \cdot N \end{aligned}$$

describes a upper bound of the expected number of distinct proposals in $\mathcal{O}'$ after $i \in \mathbb{N}_0$ rounds. Hence, $|\Phi_i| \leq \epsilon \cdot T_i$, for a fixed $\epsilon \geq 1$.

Proof. Let $\mathcal{O}'$ be a cycle or path comprising N nodes and $G := (V, E)$ the *directed* graph with $V := V(\mathcal{O}')$ that describes the proposal selections. An edge $e := (a, b) \in E := V \times V$ in G denotes that node a selects the proposal of node b, i. e., (a) $\rightarrow$ (b). Since all nodes choose a proposal the outdegree $\deg^+(x)$ of a node $x \in V$ is exactly 1. The indegree $\deg^-(x) \leq 2$ depends on the number of node x's neighbors $\Gamma(x; V)$ which select node x's proposal. If $\deg^-(x)$ is zero the proposal is lost because no neighbor selects the proposal. Assume node x is added to $\mathcal{O}'$ between nodes a and b in the cycle or path, then the potential edges in G are as follows:

1	(a) $\leftrightharpoons$ (x) (b) $\rightarrow$	5	$\leftarrow$ (a) $\leftarrow$ (x) (b) $\rightarrow$	$\Rightarrow \phi_x$ lost
2	(a) $\rightarrow$ (x) $\rightarrow$ (b) $\rightarrow$	6	$\leftarrow$ (a) (x) $\rightarrow$ (b) $\rightarrow$	$\Rightarrow \phi_x$ lost
3	(a) $\leftrightharpoons$ (x) $\leftarrow$ (b)	7	$\leftarrow$ (a) $\leftarrow$ (x) $\leftarrow$ (b)	
4	(a) $\rightarrow$ (x) $\leftrightharpoons$ (b)	8	$\leftarrow$ (a) (x) $\leftrightharpoons$ (b)	

Table 4.3 – Proposal reduction when using D_{R}: node x added between nodes a and b

Table 4.3 shows that exactly 2 out of 8 possibilities lead to loss of node x's proposal. By induction, this applies to all nodes added to a cycle $\mathcal{O}'$ with $N \geq 3$ and $\frac{1}{4} \cdot N$ proposals are expected to get lost. When $\mathcal{O}'$ is a path, then nodes at the ends of the path have no choice but to select the proposal of their only neighbor. This case has to be investigated separately.

1	a ⇋ b ← c	⇒ ϕ_c lost
2	a → b ⇋ c	⇒ ϕ_a lost

1	a → b → c ⇋ d	⇒ ϕ_a lost
2	a → b ⇋ c ← d	⇒ ϕ_a, ϕ_d lost
3	a ⇋ b c ⇋ d	
4	a ⇋ b ← c ← d	⇒ ϕ_d lost

Table 4.4 – Possible selections in a path comprising $N = 3, 4$ nodes

Table 4.4 considers the possible selections in a path comprising $N = 3, 4$ nodes. The selections show that with $N = 3$ nodes one proposal gets lost for sure. With $N = 4$ nodes, one proposal gets lost with at least probability $\frac{3}{4}$. This satisfies the induction start—also for paths.

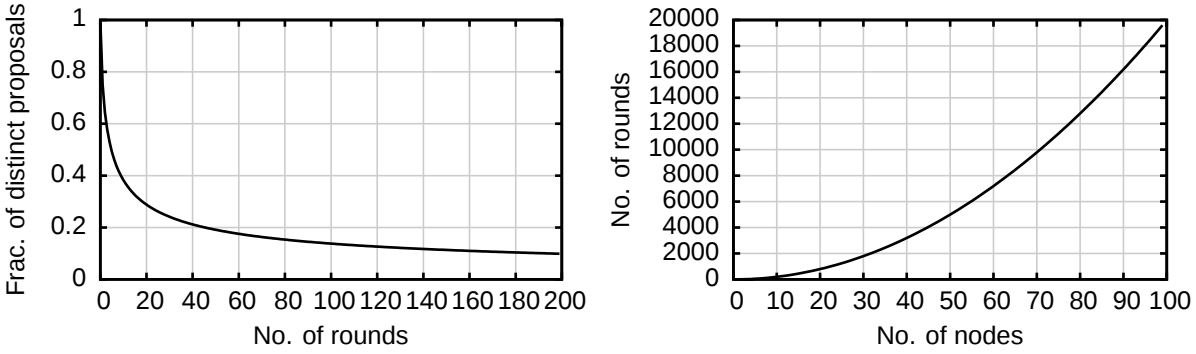
The induction in Table 4.3 allows to derive the recurrence relation T_i of the lemma. A node proposes a unique random number, i. e., no other node in $\mathcal{O}'$ proposes the same random number, with probability $\frac{T_i}{N}$. The induction only holds, if nodes a, x, and b, all propose unique random numbers. The probability for this is $\left(\frac{T_i}{N}\right)^3$. Thus, the probability that node x's unique proposal gets lost is $\frac{1}{4} \cdot \left(\frac{T_i}{N}\right)^3$. Applying this probability to all nodes leads to the recurrence relation of T_i. □

This lemma allows to proof that the behaviour applies to random graphs as well:

Theorem 4.8 (D_R on connected random graphs) – Lemma 4.7 applies also for connected random graphs G with at least 3 nodes and a minimum node degree of $\widehat{k} \geq 2$, i. e., $V(G) \geq 3$ and $\forall v \in V(G) : \deg(v; G) \geq \widehat{k}$, with high probability.

Proof. Let $\mathcal{O}'$ be a connected random graph with at least 3 nodes and a minimum node degree of $\widehat{k}$. Then, in comparison to the path and cycle graph, each node has to choose a proposal from least $\widehat{k}$ neighbors. Hence, the probability that a no neighbor chooses a node's proposal is $f(\widehat{k}) := \left(1 - \frac{1}{\widehat{k}}\right)^{\widehat{k}}$. $f(x)$ increases monotonically from $f(2) = \frac{1}{4}$ to $f(x) \to e^{-1}$ for $x \to \infty$. Hence, at least $\frac{1}{4}$ of all proposals are lost and Lemma 4.7 applies for G as well. □

The recurrence relation in Lemma 4.7 is not solvable by using a well-known methodology (e. g., master-method), i. e., no closed-form is available. Thus, the values are calculated to gain further insights on the behaviour on the random number agreement using D_R. Figure 4.8(a) shows the fraction of distinct proposals as function of rounds of the gossip-based random number agreement algorithm. The Figure shows that convergence to a single proposal value x needs many rounds and gets improbable as the number of rounds increases. However, in the first rounds, the slope is very steep, therefore, the number of distinct proposals reduce massively. Figure 4.8(b) shows the number of rounds needed to converge to a single unique proposal as a function of the amount of nodes. It also shows that D_R is hardly scalable, since convergence time is not growing sub-linearly.



(a) Fraction of distinct proposals (b) Convergence time

Figure 4.8 – Fraction of unique proposals and convergence speed using the recurrence relation of D_{R}

4.4.5 Feasibility Study

Section 4.4.3 presented the selection functions D_{R}, $D_{\mathrm{RMR}}D_{\mathrm{MH}}$, D_{MMH} for the gossip-based random number agreement. The main goal of this section is to confirm the theoretic findings of Section 4.4.2 with experiments. Those experiments should use an unstructured overlay graph as proposed by Section 4.2.

To achieve this goal, a custom simulation has been developed. This simulation allows to simulate Algorithm 4.1 on a large-scale unstructured overlay graph with different selection functions. For reasons of scalability, the simulation is limited to the algorithmic aspects. Therefore, neither message loss nor a network communication channel is simulated between the nodes. One simulation run works as follows:

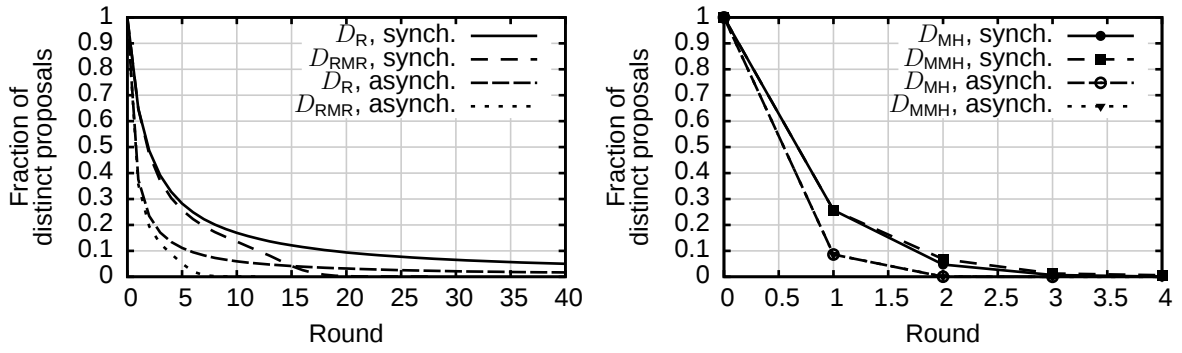
- First, an unstructured overlay graph comprising N nodes is constructed with a minimum degree of $\frac{1}{2} \cdot k$ and maximum degree of k. The simulator builds this graph as follows. Initially, the simulator adds N nodes to the graph. Then, the simulator iteratively connects nodes with less than $\frac{1}{2} \cdot k$ neighbors with another randomly chosen node. The result is an unstructured overlay graph as proposed in Section 4.2.
- Second, in each simulation round the simulator performs one round of the algorithm on all nodes.

The simulation stops until the number of distinct proposals of all nodes reaches 1 or the number of simulation rounds reaches 100.

The choices of random and maximum selection functions used for Algorithm 4.1 are simulation parameters. Additionally, the simulation distinguishes between two simulation modes:

Synchronous mode: In synchronous mode, the simulation runs Algorithm 4.1 absolutely synchronous. This means, in each simulation round, the simulator lets all nodes select their proposal at the same time. Then, all nodes send their proposal to their neighbors at the same time as well[6].

[6]The analysis of D_{R} in Section 4.4.4 assumes this mode.



(a) Distinct proposals when using D_{R} or D_{RMR}.

(b) Distinct proposals when using D_{MH} or D_{MMH}.

Figure 4.9 – Distinct propoals during the random number agreement.

Asynchronous mode: In asynchronous mode, the simulation runs Algorithm 4.1 asynchronously. This means, in each simulation round, each node selects an proposal and sends the selected proposal to its neighbors immediately. This has an influence on the proposal selection in the same round. The influence depends on the order in which the algorithm is simulated on the nodes in each simulation round. To simulate asynchronous behaviour, the simulation shuffles the order in each simulation round. The asynchronous mode results in a more realistic simulation because message delays and asynchronous clocks in real networks would have a similar effect.

The simulator records the performance of the random number agreement in terms of the following metrics:

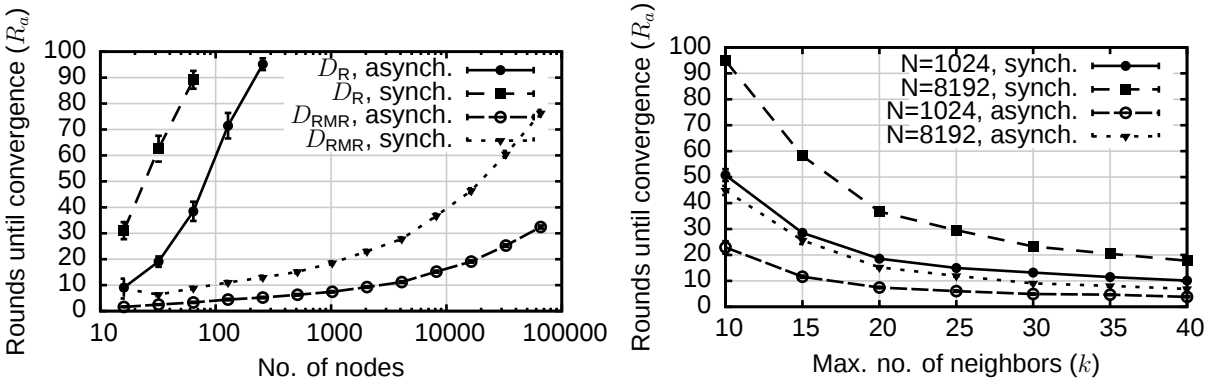
- fraction of distinct proposals among the nodes $\frac{|\Phi_i|}{N}$ after simulation round i,
- convergence time, i. e., required number of rounds R_a for the random number agreement, and
- fraction of successful agreements in the presence of leaks.

Each simulation run is repeated 100 times with different initialization seeds for the pseudo random number generator[7] which is used to build the unstructured overlay graph and initial random number proposals on the nodes. The simulator provides means and 95% confidence intervals for each performance metric. The following sections describe the experiments and discuss the results.

Distinct Proposals

The first experiment evaluates the fraction of distinct proposals, i. e., $\frac{|\Phi_i|}{N}$ in each round of Algorithm 4.1using the four selection functions in asynchronous and synchronous mode. Figure 4.9(a) shows the results for the random selection functions D_{R} and D_{RMR}. The curves of D_{R}, both in asynchronous and synchronous mode, reflect the behavior already predicted and proven by Theorem 4.8. D_{R} reduces the number of distinct proposals tremendously in the first rounds indicated by a steep slope between rounds 0 and 5. This makes it more likely that nodes may agree on

[7] Mersennne twister



(a) Convergence time with growing no. of nodes ($k = 20$)

(b) Convergence time with growing no. of neighbors ($D_{\mathrm{RMR}}, N = 1024$)

Figure 4.10 – Convergence time of the random number agreement.

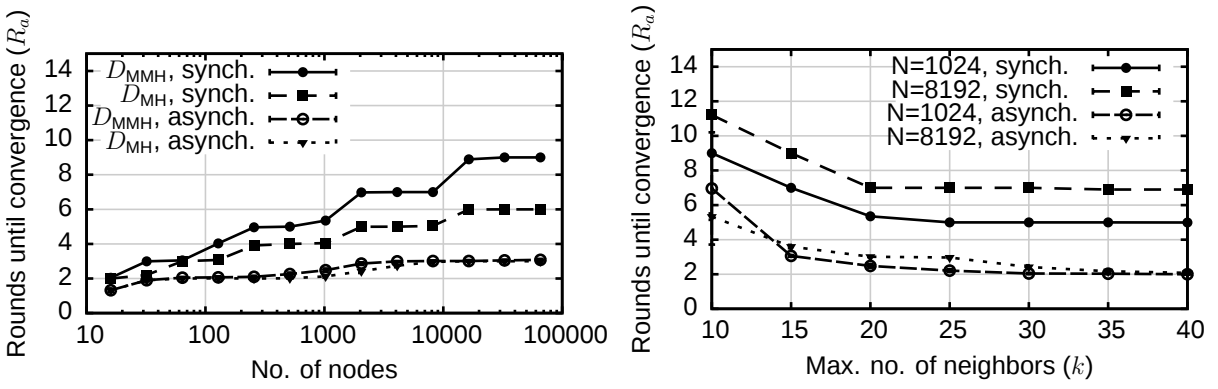
a distinct proposals that has a relative majority among all proposals when using D_{RMR}. The curves of D_{RMR} illustrate this behaviour. During the first rounds, D_{RMR} is almost equal to D_{R} in terms of distinct proposals in the first few rounds, i. e., 0 to 10 in the synchronous mode, and, 0 to 5 in the asynchronous mode. When the fraction of distinct proposals falls below 0.1 nodes recognize the relative majority of a certain proposal and accelerate the convergence of the agreement, i. e., the fraction of distinct proposals drops to $\frac{1}{N}$. In summary, the experiments confirm the theoretical insights gained in Section 4.4.4 and Figure 4.8. Furthermore, they show the impact of the relative majority vote in D_{RMR}.

Figure 4.9(b) shows the results for the maximum selection functions D_{MH} and D_{MMH}. In contrast to the random selection functions, the fraction of distinct proposals decreases rapidly, i. e., under 0.3 after the first round. The reason for this is the flooding character of the maximum selection functions. A node just has to receive the proposal with the global maximum hash and has converged already. Since the majority vote only provides stability, it does not have much influence on the fraction of distinct proposals. That is why the curves of D_{MH} and D_{MMH} are almost identical.

In all cases the asynchronous mode decreases the fraction of distinct proposals much faster than the synchronous mode. This is because in asynchronous mode, it less likely that two adjacent nodes just exchange their proposal and do not reduce the fraction of distinct proposals.

Convergence Time

The next experiment sheds light on the number of rounds needed for convergence, i. e., the convergence time as function of the number of nodes N and maximum number of neighbors k. Figure 4.10(a) shows the convergence time as function of the number of nodes N with the random selection functions D_{R} and D_{RMR} with a maximum number of neighbors $k = 20$. The curves show the bad convergence time of D_{R} and D_{RMR}. D_{R} exceeds the simulation round limit of 100 when the number of nodes exceeds 64 in synchronous, and 128 in asynchronous mode. This behavior has been predicted using the recurrence relation in Figure 4.8(b). D_{RMR}



(a) Convergence time with growing no. of nodes ($k = 20$)

(b) Convergence time with growing no. of neighbors ($D_{\text{RMR}}, N = 1024$)

Figure 4.11 – Convergence time of the random number agreement.

performs better, i. e., has a much lower convergence time with a growing number of nodes. This is because the relative majority vote of D_{RMR} accelerates the agreement. However, the curves, when using D_{RMR} show a super linear trend with a growing number of nodes. This makes D_{RMR}—although it is better than D_{R}—a bad candidate for an random number agreement with a large number of nodes as well.

Figure 4.10(b) shows the convergence time as function of the maximum number of neighbors k and with fixed numbers of nodes $N = 1024$ and $N = 8192$ when using D_{RMR}. One insight from this Figure is that the convergence time decreases when the maximum number of neighbors grow. This is because the maximum number of neighbors decides on the ability of a node to recognize that the proposal is having the network-wide relative majority. Informally, the view of a node on the network-wide relative majority situation of proposals becomes sharper the more neighbors a node can consider. Another insight is, that the impact of a growing k is decreasing. In asynchronous mode, a maximum number of neighbors above $k > 20$ does not decrease convergence time significantly in case of constant number of nodes $N = 1024$ and $N = 8192$.

Figure 4.11(a) shows the convergence time as function of the number of nodes N with the maximum selection functions D_{MH} and D_{MMH} and a maximum number of neighbors $k = 20$. In contrast to the random selection functions, the maximum selection functions converge very fast and in a scalable manner. The curves show that the number of rounds needed to agree on a random number increases within logarithmic complexity. An interesting point is, that D_{MMH} makes the agreement slightly faster than D_{MH}—at least in synchronous mode. The reason for this is that the absolute majority vote is not stable on all nodes. Thus, the proposal representing the absolute majority may flip several times and delay the agreement slightly. Obviously, in asynchronous mode, this behavior is less likely. To understand the steps in the curves, one must understand that the maximum selection functions basically flood the network limited times for an agreement. Flooding correlates with the diameter of the unstructured overlay graph. The diameter itself is an

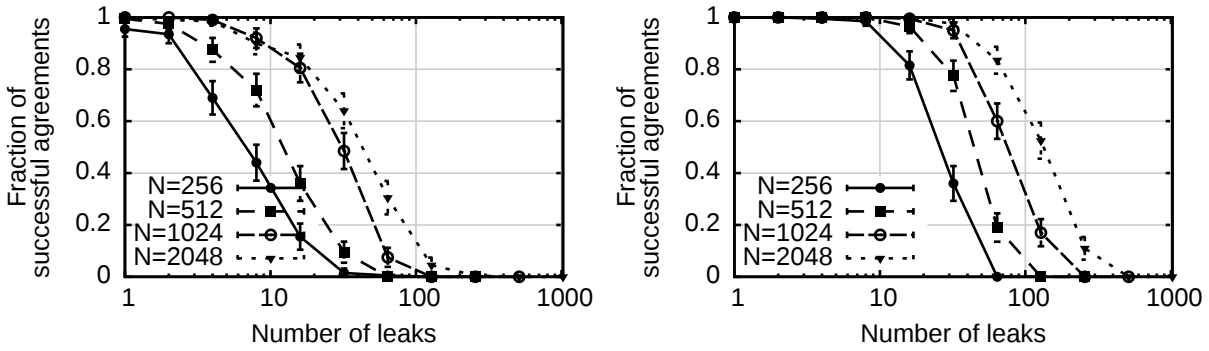


Figure 4.12 – Impact of leaks on the CID agreement with growing number of nodes N with $k = 20$.

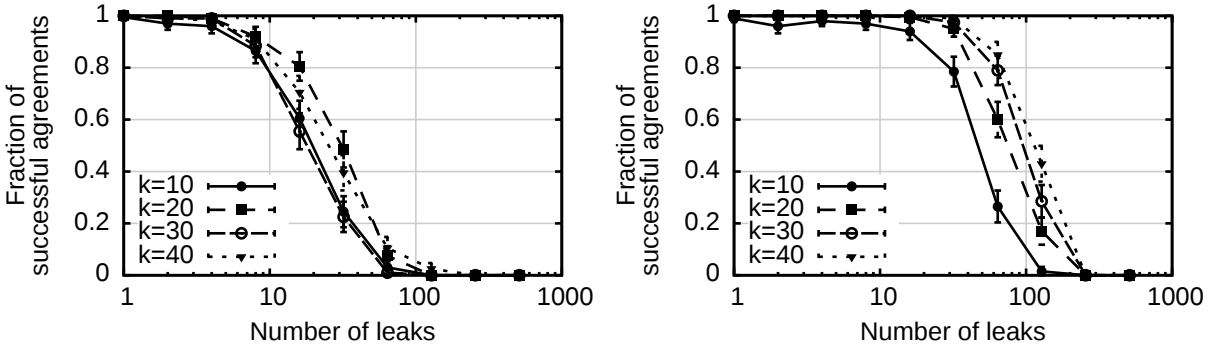


Figure 4.13 – Impact of leaks on the CID agreement with growing number of max. overlay neighbors k with $N = 1024$.

integer because each link has a weight of 1 and suffers from quantization effects, i. e., several graphs have the same diameter. This explains the steps in the curves.

Figure 4.10(b) shows the convergence time as function of the maximum number of neighbors k and with fixed numbers of nodes $N = 1024$ and $N = 8192$ when using D_{MMH}. As already observed in the random selection functions, a larger number of maximum overlay neighbors does not significantly improve the convergence time for $k > 20$.

Impact of Leaks

Finally, the last experiment addresses the random number agreement in the presence of leaks. For this purpose, the experiment uses a setup similar to the worst-case presented in Section 4.3.3. It comprises two unstructured overlay graphs connected by "leaking" nodes connected to both graphs. Figures 4.12 and 4.13 show the experiments comparing D_{MMH} and D_{RMR} in scenarios of growing number of nodes, neighbors, and leaks. When comparing Figures 4.12(b) and 4.12(a) D_{RMR} is more robust against leaks than D_{MMH} with a growing number of nodes. Surprising is, however, that D_{MMH} is robust against leaks to some extend. The reason for this behaviour is as follows: when 50 percent of the nodes agree on the proposal with the maximum hash in each unstructured overlay graph fast enough, leaks

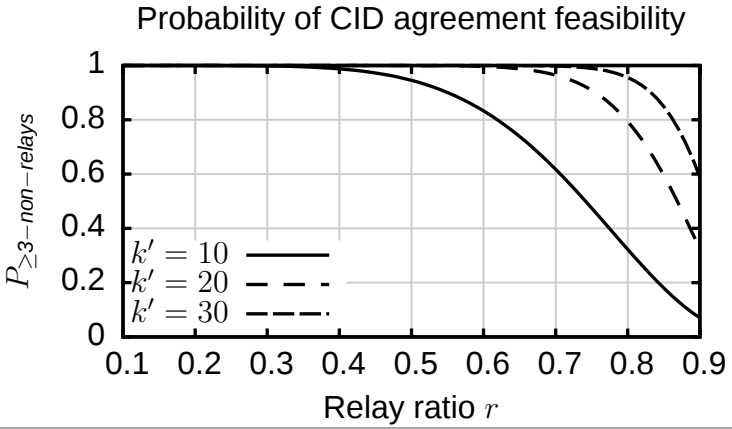


Figure 4.14 – Feasibility of the CID agreement subject to the relay ratio r and number of overlay neighbors k'

can not change the agreement. For both selection functions, D_{RMR} and D_{MMH}, Figures 4.13(b) and 4.13(a) show the low impact of a growing maximum number of overlay neighbors k on the robustness against leaks.

4.4.6 Feasibility of the CID Agreement in a Connectivity Domain

One problem not yet been considered is the necessity of the non-relay having non-relay overlay neighbors in order to agree on a CID. More precisely, $\mathcal{O}'$ needs to be connected and comprise all non-relay nodes of a connectivity domain. When taking the worst-case scenario into account, cf. Section 4.3.3, it is possible that all overlay neighbors of a non-relay node are relays if the relay ratio r is high. In this case, a connectivity domain-wide CID agreement is impossible when $\mathcal{O}'$ is not connected.

Intuitively, each non-relay node must have at least one non-relay overlay neighbor to agree on a CID. Furthermore, $\mathcal{O}'$ is only connected with high probability, if each non-relay node has at least 3 non-relay overlay neighbors. In this case, $\mathcal{O}'$ is a random graph and $\forall v \in V(\mathcal{O}') : \deg(v) \geq 3$. For the worst-case scenario, the probability that a non-relay node has 3 non-relay overlay neighbors is calculated as follows with subject to the relay ratio r and the node's number of overlay neighbors k':

$$P_{\geq 3-non-relays}(k', r) := \sum_{i=0}^{k'-3} \binom{k'}{i} \cdot r^i \cdot (1-r)^{k'-i}$$

The $P_{\geq 3-non-relays}$ calculates the probability that each non-relay node has at least 3 non-relay neighbors. Figure 4.14 shows a plot of $P_{\geq 3-non-relays}$ as function of the relay ratio r and for different numbers of overlay neighbors $k' \in \{10, 20, 30\}$. One can see that the probability that the CID agreement is feasible decreases with a growing relay ratio r. A larger number of overlay neighbors k', however, can compensate a higher relay ratio. This is because the more overlay neighbors a non-relay node has the higher the probability of one of these neighbors being a non-relay. With the maximum number of neighbors $k = 20$ the non-relay nodes

have at least $k' \geq 10$ overlay neighbors[8]. Thus, a relay ratio of ≈ 0.4 is still acceptable.

4.4.7 Summary

This section discussed the properties of selection functions for a gossip-based random number agreement algorithm. Two proposal selection functions D are promising candidates for use with the gossip-based random agreement algorithm: on the one hand, D_{RMR} which is robust against leaks, but has moderate convergence time. On the other hand, D_{MMH}, which is vulnerable against leaks, but has a fast convergence time, i. e., uses a number of rounds depending on the logarithmic diameter of the unstructured overlay graph. The majority selections used in both selection functions advise a maximum of overlay neighbors of about $k \approx 20$. The following table summarizes the results:

Requirement	D_{RMR}	D_{MMH}
Convergence time	Bad	Very good
Leak robustness	Very good	Moderate/Good
Stability	Very good	Very good

To use the best of both worlds, i. e., D_{RMR} and D_{MMH}, Algorithm 4.1may use D_{RMR} when the leaks are still probable, i. e., during the first R_{RMR} rounds, and switch to D_{MMH} to finally agree on a CID in a timely manner when leaks are unlikely. Formally, D_{HYBRID} implements this behaviour:

$$D_{\text{HYBRID}}(\mathcal{P}) := \begin{cases} D_{\text{RMR}}(\mathcal{P}), & \mathit{round} \leq R_{\text{RMR}} \\ D_{\text{MMH}}(\mathcal{P}), & \text{otherwise} \end{cases} \tag{4.16}$$

The next section uses the discussed mechanisms and algorithms to design the *Connectivity Measurement Protocol* (*CMP*).

4.5 Connectivity Measurement Protocol (CMP)

The Connectivity Measurement Protocol (CMP) determines accurate connectivity identifiers (CIDs) for each connectivity domain and provides information about relays. For this purpose, CMP implements the solution proposed in Section 4.2 in conjunction with the relay detection studied in Section 4.3, and CID agreement algorithms of Section 4.4.

Figure 4.15 shows a brief overview of CMP's placement in the CD middleware and its internal structure. It is placed upon the communication and discovery layer that provides *direct* communication functionality between nodes in homogeneous networks and functionality to discover nodes already running CMP. CMP uses the communication layer to build an unstructured overlay, as proposed in Section 4.2. Furthermore, CMP detects whether a node is a relay using the triangle checks discussed in Section 4.3. If a node is not a relay, CMP uses the CID agreement algorithm as discussed in Section 4.4 to agree on a CID for each connectivity domain with the node's overlay neighbors. CMP provides the determined CIDs to

[8] Assuming the number of nodes is greater than k

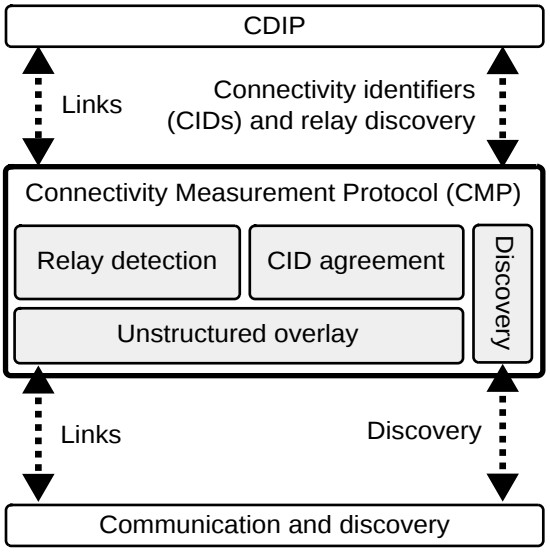


Figure 4.15 – CMP's placement and its internals

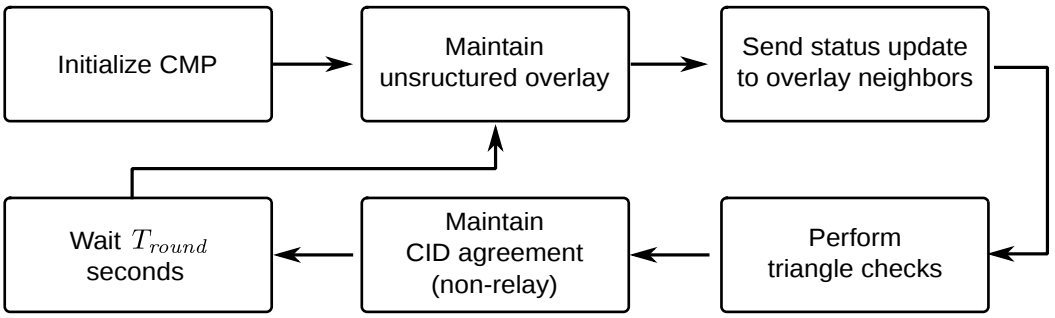


Figure 4.16 – CMP's main loop.

the *Connectivity Domain Interconnection Protocol* (*CDIP*), so it can use them to forward messages between different connectivity domains. The discovery module provides peer sampling and random walks functionality to build the unstructured overlay and exchange information about relays in each connectivity domain.

4.5.1 CMP's Main-Loop, State Machine, and Node Status

This section gives a general overview how CMP works. This includes the CMP's main loop, the node's status, the tasks in each of CMP's states described in a finite state machine, and CMP's parameters with a preliminary discussion.

Main Loop

To understand CMP, it is important to get a general idea how CMP works. CMP is gossip-based protocol and operates in rounds implemented by a single loop. A round is started every T_{round} seconds and comprises the handling of CMP's mechanisms. Figure 4.16 gives a brief overview of CMP's loop consisting of the following steps:

Status	Description
– Status	
$state(x)$	Current state of the node:HOLD, AGREEING, or, STABLE.
$isRelay(x)$	*true* if node detected it is a relay.
$\phi(x)$	Current CID proposal.
– Additional Information	
$\Delta_{checked}(x)$	*true* if overlay neighbor x has been checked using a triangle check.

Table 4.5 – CMP's status information and additional information to a node's overlay neighbors. CMP's status is sent to overlay neighbors each T_{round} seconds on change. CMP stores additional information locally on each node.

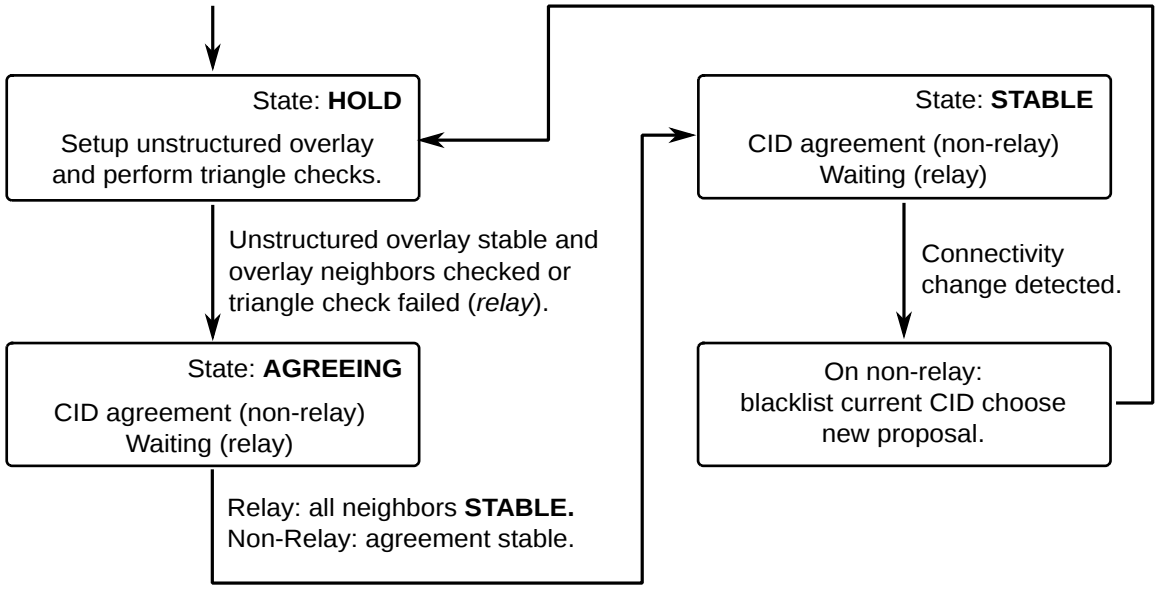


Figure 4.17 – CMP's finite state machine.

1. CMP is initialized.
2. CMP maintains the unstructured random overlay, as described in Section 4.5.2
3. CMP sends the node's status to all overlay neighbors, if changed.
4. CMP initiates triangle checks between overlay neighbors as described in Section 4.5.3.
5. On a non-relay node, CMP agrees on a CID using some selection function, as described in Section 4.5.4.
6. Finally, CMP waits T_{round} seconds and the loop starts over.

Node Status and Finite State Machine

The node's status comprises three fields: the node's state according to CMP's finite state machine, a flag indicating whether the node is a relay, and, the node's

Parameter	Default	Description
– General		
T_{round}	1s	Waiting time before next round
– Unstructured Overlay		
k	20	Maximum number of overlay neighbors per node
$discoveryRate$	2	Number of discovery attempts per round
$T_{link-timeout}$	5s	Link keep-alive timeout
– Triangle Checks		
$T_{\Delta-timeout}$	3s	Waiting time for triangle check messages to return
$\Delta_{retries}$	2	Triangle check retries until a failure is assumed
– CID Agreement		
D	D_{MMH}	CID agreement selection function used: D_{MMH}, D_{RMR}, or D_{HYBRID}
$R_{blacklisted}$	5 rounds	No. of rounds a CID is blacklisted after connectivity change
R_{RMR}	10 rounds	No. of rounds to use D_{RMR} when using $D := D_{\text{HYBRID}}$

Table 4.6 – Overview of CMP's parameters and default values.

current proposal for the CID agreement. For convenience, Table 4.5 summarizes the status. For a binary representation of the status refer to the Appendix B.1.

CMP's finite state machine is an abstract view on the operations CMP performs in the main loop. The state decides which operations are actually performed in CMP's main loop. The finite state machine has three states which have the following meaning:

- HOLD: CMP is still setting up the unstructured overlay or performing triangle checks.
- AGREEING: CMP wants to agree on a CID or detected it is a relay and waits for its overlay neighbors to pass into a STABLE-state.
- STABLE: CMP has converged and is stable.

Figure 4.17 shows a state transition diagram of CMP's finite state machine. In HOLD-state, CMP sets up the unstructured overlay and performs triangle checks. When the unstructured overlay is stable, and all overlay neighbors have been checked using triangle checks, or a triangle check failed, CMP passes to the AGREEING-state. In this state, CMP uses the ϕ status field to agree on a CID with its overlay neighbors, if the node has not detected it is a relay. If CMP detected that the node is a relay, CMP waits until all overlay neighbors are in STABLE-state. When all overlay neighbors are in STABLE-state or the agreement is stable the node passes to the STABLE-state. This state is only left, when a connectivity change has been detected.

Parameter Overview and Discussion

CMP uses several parameters summarized in Table 4.6. There are 4 groups of parameters: parameters concerning CMP's general operation, unstructured overlay, triangle checks, and, CID agreement. The following list describes each parameter and its effect on CMP's performance in detail.

CMP's general protocol parameter:

T_{round} denotes the waiting time before the next round in seconds. T_{round} has impact on CMP's convergence time and bandwidth consumption. The lower T_{round}, the shorter is the convergence time and the higher is the bandwidth consumption. This is because all of CMP's operations are performed in its main loop, i. e., unstructured overlay maintenance, triangle checks, and, CID agreement. However, each operation produces additional traffic. To limit the traffic and to take into account that a message needs some time for delivery, CMP's default value is set to $T_{round} = 1$ second.

Protocol parameters concerning the unstructured overlay:

k denotes the (expected) maximum number of overlay neighbors per node. The unstructured overlay connects to at least $\frac{1}{2} \cdot k$ overlay neighbors, if enough nodes are available. The impact on CMP's performance by this parameter is twofold:

- First, it influences the bandwidth consumption on each node, because the node sends status updates each T_{round} second(s) to its overlay neighbors. Therefore, the upper bound of the bandwidth consumption grows in proportion to k and the lower bound to $\frac{1}{2} \cdot k$.
- Second, it influences the CID agreement algorithm convergence time and stability. The selection functions D_{RMR} and D_{MMH} used within the CID agreement both use majority selections among the proposals received from the overlay neighbors. Thus, the lower the number of neighbors per node is, the less proposals are available in the majority vote. As Section 4.4 points out the CID agreement convergence time benefits from a larger number of overlay neighbors. Furthermore, the less overlay neighbors, the easier for a small number of nodes to influence the majority selection of a node.

The theoretical findings in Sections 4.4 and 4.3 allow the claim that $k = 20$ is a adequate default value for this parameter.

discoveryRate denotes the number of trials to discover new overlay neighbors for the unstructured overlay per round. The higher this value, the faster the unstructured overlay will converge because the overlay neighbors get discovered more quickly. Commonly, the traffic grows in proportion to *discoveryRate*. To limit the convergence time of CMP's unstructured overlay to less than 10 seconds with $k = 20$, *discoveryRate* default value set to 2.

$T_{link-timeout}$ denotes the timeout of a link. The main impact of this parameter is bandwidth consumption and time to detect a connectivity change. The shorter the timeout, the more frequently keep-alive messages need to be sent—increasing bandwidth consumption in the same proportion. Naturally, the smaller the timeout of the link, the faster CMP detects change in connectivity, because CMP detects connectivity changes by observing link failures. The default value

is set to $T_{link-timeout} = 5$ seconds to save bandwidth and to achieve a reasonable connectivity detection delay (max. $T_{link-timeout}$ seconds).

Parameters concerning the triangle checks:

$T_{\Delta-timeout}$ denotes the timeout when waiting for triangle check messages to return to the sender. The default value is set to $T_{\Delta-timeout} = 3$, that means, that CMP assumes that the message delay does not exceed 1 second which is common in today's networks.

$\Delta_{retries}$ denotes the number of retries of the triangle check when the triangle check messages timed out until the node assumes that the triangle check has failed. A high number of triangle check retries compensates the loss of triangle messages. The default value of $\Delta_{retries} = 2$ compensates the loss of maximal one triangle check message out of 4. This is equivalent to a packet-loss smaller than 25 per cent.

Parameters concerning the CID agreement:

D denotes the selection function used for the CID agreement algorithm. It represents one of the selection functions introduced in Section 4.4, i. e., D being D_{MMH}, D_{RMR}, or D_{HYBRID}.

$R_{blacklisted}$ denotes the number of rounds a CID is blacklisted after a CMP detected a connectivity change. The main purpose of this value is to prevent CMP from re-converging to the former CID. $R_{blacklisted}$ is associated with a trade-off: on the one hand, the node needs to wait for the other nodes to detect the connectivity change and replace the old CID at all nodes by a new CID. On the other hand, the node might be in the minority of nodes that detected a connectivity change, i. e., it is probable that the detected connectivity change is a false-positive. In this case, the node should quickly fall back the the old CID. CMP commonly uses a conservative timeout of $R_{blacklisted} = 5$ rounds to give other nodes enough time to detect link failures and, therefore, the connectivity change as well.

R_{RMR} denotes the number of protocol rounds CMP uses the D_{RMR} selection function for agreeing on a CID. This parameter provides a trade-off between robustness against undetected relays (leaks) and convergence time. A higher number of D_{RMR} provides more robustness while a smaller number provides short convergence time. The default value of 10 rounds ensures that the steep drop of distinct proposals seen in Section 4.4.5 is included in the period.

4.5.2 Unstructured Overlay and Status Updates

CMP builds an unstructured overlay by establishing a limitied number of links to randomly selected nodes. Algorithm 4.2 describes the overlay construction. Before CMP starts, CMP initializes the unstructured overlay in *initOverlay*() and chooses a 128-bit long node identifier (nid) randomly so it is unique with high probability.

Then, in each round, CMP executes *maintainOverlay*(). This method tries to discover new overlay neighbors when the current number of overlay neighbors k' is below $\frac{1}{2} \cdot k$. To allow stabilization of the unstructured overlay if less than $\frac{1}{2} \cdot k$ nodes exist, the number of discovery trials is limited by k. The discovery itself is done by the discovery module explained in Section 4.5.5. When enough overlay neighbors

are discovered, the overlay is considered to be stable and CMP remembers the number of overlay neighbors k'' in this state. When half of the k'' links fail, CMP re-stabilizes the unstructured overlay by resetting the discoveries and overlayStable variables. When a new link to an overlay neighbor has been established, CMP exchanges the node identifiers (NIDs) using a keep-alive message and initializes its status.

CMP keeps track of the overlay neighbors using a overlay neighbor table. Table 4.8 illustrates an example of this table. One entry in the table comprises

- the node identifier, i. e., NID,
- the address used to contact the overlay neighbor, i. e., $addr(x)$,
- a flag indicating if the overlay neighbor has been checked, i. e., $\Delta_{checked}(x)$,

```
method initOverlay()
    k' ← 0;
    overlayStable ← false; discoveries ← 0;
    nid ← rand();

method maintainOverlay()
    /* Try to discover new overlay neighbors. */
    if (k' < ½ · k ∧ discoveries < k):
        foreach i ∈ {1, ..., discoveryRate}:
            discoverRandomOverlayNeighbor();
            discoveries ← discoveries+1;
    /* Overlay stable. */
    else if (¬overlayStable):
        overlayStable ← true;
        k'' ← k';
    /* On link drops, try to discover new overlay neighbors. */
    if (overlayStable ∧ k' ≤ ½ · k'') :
        discoveries ← 0;
        overlayStable ← false;

method newOverlayNeighbor(x)
    sendKeepAlive(); addToOverlayNeighborList();
    Δ_checked (x) ← false;
    state (x) ← nil;
    k' ← k' + 1;

method droppedLinkTo(x)
    recordDropForConnectivityChangeDetection();
    k' ← k' − 1;
```

Algorithm 4.2 – Construction of CMP's unstructured overlay.

	Link information				Status		
i	NID	$addr$	$\Delta_{checked}$	$last - recv/sent$	$state$	$isRelay$	ϕ
0	0x6dcd	$addr(A)$	$false$	-20ms/-100ms	HOLD	$false$	0
1	0xae4f	$addr(B)$	$false$	-98ms/-1000ms	STABLE	$true$	0
2	0x3209	$addr(C)$	$true$	-990ms/-786ms	AGREEING	$false$	0x9187
3	0x50c9	$addr(D)$	$true$	-180ms/-972ms	AGREEING	$false$	0x9187

Table 4.8 – Exemplary CMP's unstructured overlay neighbor table. For better readability, node identifiers (NIDs) and proposals (ϕ) in the table are only 16-bit instead of 128-bit.

- a timestamp that denotes the reception time of the last message from the overlay neighbor. i. e., $last - reception$, $last - sent$, and,
- the status of the overlay neighbor, i. e., $state(x)$, $isRelay(x)$, and, ϕ, cf., Table 4.5.

To keep the entries in this table up-to-date, CMP uses keep-alive messages. These keep-alive messages contain the node's identifier (NID) only. The first keep-alive message that is received from an overlay neighbor is used to initialize the NID-field in the according entry of the overlay neighbor table.

If CMP receives/sends a message from/to an overlay neighbor it refreshes respective the timestamps in the $last - recv/sent$-field of the overlay neighbor table. If the last message to an overlay neighbor was sent $\frac{1}{2} \cdot T_{link-timeout}$ seconds ago, CMP sends a keep-alive message to keep the time stamp up-to-date. If the last time a message was received from an overlay neighbor is $T_{link-timeout}$ seconds ago, CMP drops the link. Dropping a link involves removing the entry from the overlay neighbor table and sending a link drop notification message to the overlay neighbor.

Status Updates

Each round CMP sends its status using status update messages to all overlay neighbors of a node when changed. Naturally, new overlay neighbors get informed about the current status right away. When CMP receives a status update, it modifies the respective entry in the overlay neighbor table. There is one exception to the status being sent to an overlay neighbor. When the overlay neighbor has not yet been checked, i. e., $\Delta_{checked}(x) = false$, CMP always sends a status containing the HOLD-state and an invalid proposal. This is important, because CMP must first check if the node is a relay for the new overlay neighbor. Otherwise, the node might become a long-term leak in the CID agreement.

Connectivity Change Detection

CMP detects changes in connectivity properties by constantly monitoring link failures on non-relay nodes. If k'' links fail in an interval of $2 \cdot T_{link-timeout}$ seconds, CMP takes this as indication for a connectivity change.

4.5.3 Relay Detection

CMP uses triangle checks to detect if the node is a relay. For this reason, this section first describes the triangle check mechanism, and then, secondly, describes the CMP's triangle check strategy for relay detection.

Triangle Check Mechanism

As already introduced in Section 4.3, CMP uses triangle messages to initiate a triangle check. The following refines the implementation of a triangle check. In CMP, triangle messages contain a hop field, a nonce field, the source node identifier and a via address:

$$\langle \mathsf{hop}, \mathsf{nonce}, \mathsf{src}, \mathsf{via} \rangle$$

For a binary representation of the status refer to the Appendix B.1. The hop-field denotes the number of nodes the message passed. The nonce-field is a random number identifying the triangle check and kept locally unique by the sender. The src-field denotes the node identifier of the sender. The via-field contains the address[9] of an overlay neighbor of node src. When CMP initiates a triangle check on node (A) between two of its overlay neighbors (X) and (Y), CMP sends two triangle messages. First, CMP sends a triangle message to overlay neighbor (X) allocating the *via*-field with address $addr(Y)$ of overlay neighbor (Y):

$$\langle \mathsf{hop} = 0, \mathsf{nonce} = 10680, \mathsf{src} = \text{(A)}, \mathsf{via} = addr(Y) \rangle$$

Second, CMP sends a triangle message to overlay neighbor (Y) allocating the *via*-field with address $addr(X)$ of overlay neighbor (X) and having same hop, nonce, and, src-fields the same. CMP handles the incoming message in regard to the hop-field as follows:

- When CMP receives a triangle message on a node with a hop-field of 0 it extracts the address stored in the via-field and removes the field from the message. Then, it increases the hop field by one and forwards the message using the communication and discovery layer and the address extracted from the via-field.
- When CMP receives a triangle message on a node with an hop field of value 1, it increases the hop field again. Then, the node extracts the node identifier of the src-field and removes the field from the triangle message. Finally, the node forwards the triangle message using the existing link to its overlay neighbor with an node identifier equal to the NID extracted from the src-field.
- When CMP receives a triangle message on a node with an hop field of value 2, it checks if the nonce of triangle message matches the nonce of a message sent. If this is the case, the node knows that the triangle check was successful.

CMP maintains a table, illustrated by Table 4.9, of pending triangle check messages. It contains the nonce used in for the triangle messages, a time-stamp, that denotes the point in time when the triangle messages have been sent, the number of retries, and the number of triangle messages that have returned already.

[9] As provided by the communication and discovery layer

Nonce	Timestamp	Retries	Returned
10680	1332076739	0	1
26876	1332076769	0	0

Table 4.9 – Example of CMP's table of pending triangle checks.

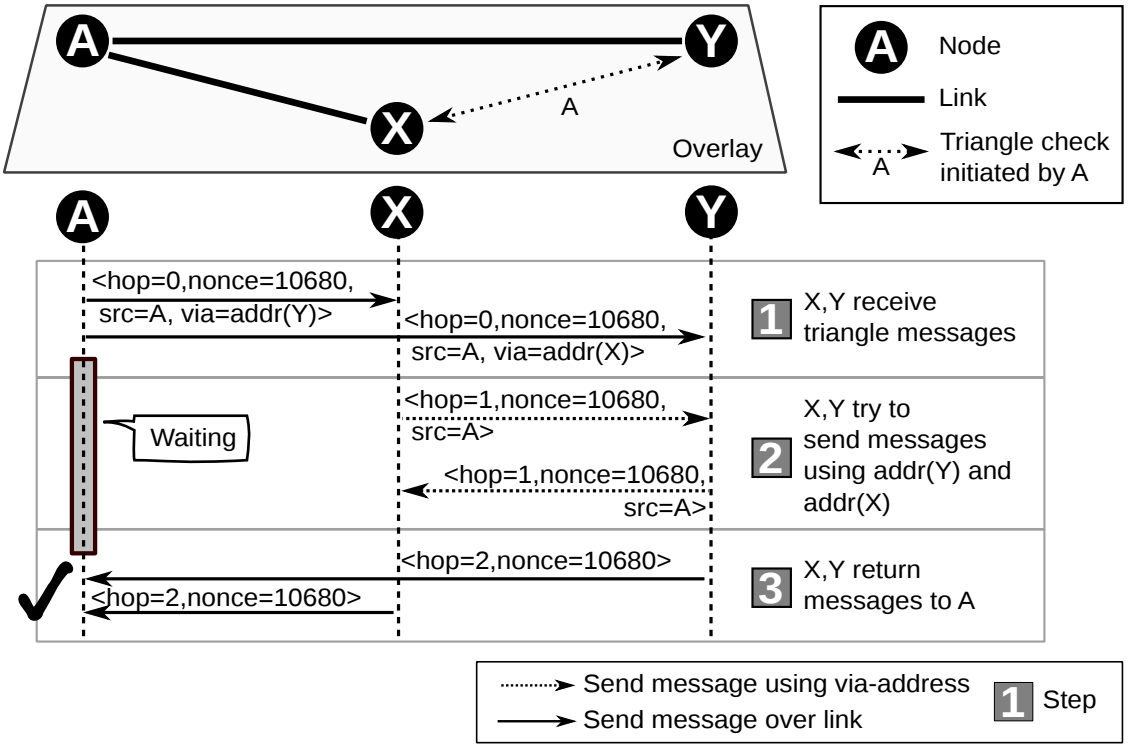


Figure 4.18 – Sequence diagram of a examplary, successful triangle check.

Each round, CMP checks this table if a triangle check timed out. If a triangle check timed out, CMP repeats sending the triangle check messages $\Delta_{retries}$ times. When all attempts fail, it assumes that the triangle check has finally failed, concludes that the node is a relay and sets its *isRelay*-status to *true*. Only if CMP on the node receives a all triangle check messages the triangle check has been successful. When a triangle check failed or succeeded CMP remove the respective entry in the table of pending triangle checks.

Figure 4.18 shows a sequence diagram of a successful triangle check between two overlay neighbors (X) and (Y) issued by (A):

- In the first step 1, node (A) sends triangle messages to nodes (X) and (Y).
- In the second step 2 those nodes use the addresses of the via-field to forward the triangle message to node (Y) and node (X) respectively. The nodes remove the via-field because it is not needed anymore.
- Finally, in step 3, nodes (X) and (Y) return the triangle messages to the sender, i. e., node (A), using the existing link. Before that, the nodes remove

the src-field because it is not needed anymore. When node Ⓐ receives a triangle message it compares the nonce of the message received with the pending checks. After receiving both triangle messages. Node Ⓐ knows that the triangle check was successful.

Triangle Check Strategy

CMP initiates triangle checks each round on unchecked overlay neighbors. CMP uses a slightly modified version of the overlapping strategy introduced in Section 4.3.5 to perform relay detection on a node in the unstructured overlay. More precisely, CMP initiates a triangle check for every unchecked overlay neighbor with its predecessor in the list of overlay neighbors. The first overlay neighbor is checked with the last overlay neighbor in the list. After the triangle check has been successful or failed, the $\Delta_{checked}(x)$ flag is set to *true*. As already pointed out in Section 4.3.5 this strategy leads to k' triangle checks initiated by each node.

Optimization: Triangle Check Skipping

To reduce the number of necessary triangle checks, CMP skips triangle checks between overlay neighbors that are in STABLE-state, propose the same CID, and are not relays. This results in a convergence-time equivalent to the time needed to join the unstructured overlay.

4.5.4 CID Agreement

CMP embeds the CID agreement according to the gossip-based random agreement algorithm described in Section 4.4 by inclusion of the proposal in the node's status updates. When CMP is in AGREEING-state and has not detected that the node is a relay it selects a proposal using the selection functions D_{MMH}, D_{RMR}, or, D_{HYBRID} each round. This proposal is then sent to all overlay neighbors by a status update. The selection function takes all proposals from overlay neighbors that are non-relays, and which are in AGREEING or STABLE state, as input parameter $\mathcal{P}$.

The CID agreement is considered to be stable when $\frac{2}{3}$ of all nodes in AGREEING-state propose the same CID. The value of $\frac{2}{3}$ is chosen so the majority is slightly higher than the absolute majority vote used in D_{MMH}. With $\frac{2}{3}$ of the nodes proposing the same CID it is more likely that the CID represents global absolute majority.

Blacklist of CID on Connectivity Change

On a connectivity change a non-relay node resets its state and re-converges. To prevent CMP from converging to the same CID, CMP blacklists the former CID for $R_{blacklisted}$ rounds. During this time, the CID agreement algorithm ignores the blacklisted proposals when selecting a new proposal using the selection function D.

4.5.5 Discovery

CMP needs to discover overlay neighbors in same connectivity domain(s). Discovering a node means knowing its underlay address in this context. The discovery module provides this functionality by implementing random walks and peer sampling techniques. Two cases are considered:

Random Node Discovery

CMP uses random node discovery to connect to randomly selected overlay neighbors for its unstructured overlay. The function *discoverRandomOverlayNeighbor* is called for this purpose. If such a request is made, CMP executes the following steps:

1. If no link has been established, CMP uses the communication and discovery to find some nodes to bootstrap the overlay.
2. If some links already exist, CMP employs a random walk as specified in [89] to find further random nodes. On a non-relay node the random walk is limited to the node's connectivity domain. To ensure this, the random walk never crosses a relay.
3. Finally, if CMP discovers a node, it sets up a link and reports this to the unstructured overlay.

Random Relay Discovery

For random relay discovery CMP implements a gossip-based peer sampling protocol similar to [50]. CMP keeps two lists: a positives and a negatives list. The lists contain triples, called *relay-triples*, containing the address of a relay, a sequence number, and a flag indicating if the triple is positive or negative. The sequence number denotes the freshness of the relay-triple. The higher the sequence number the fresher it is. The positive/negative flag indicates if the relay-triple denotes if the relay is alive/dead. On a relay, CMP piggybacks a positive relay-triple containing the relay's address and sequence number to each gossip/keep-alive message sent to its overlay neighbors. Furthermore, if the set of the relay's overlay neighbors changes CMP increases the sequence number.

If CMP receives a piggybacked relay-triple from a relay it adds the triple to the respective lists. If a list already contains a relay-triple with the same underlay address, it replaces the relay-triple when the sequence number of the relay-triple in the list is smaller. If the positives list contains a relay-triple with the same address and a smaller sequence than the received negative relay-triple, the relay-triple is removed from the list. This ensures that dead relays are removed from the positive list.

On non-relay nodes, CMP chooses one positive/negative triple from each of the lists randomly and piggybacks them to each gossip/keep-alive message. This way, the relay-triples get disseminated in the connectivity domain. If a CMP on a non-relay detects that a relay in among the node's overlay neighbors fails, it adds a respective triple to its negative list. This way dead relay-triples are removed from the positives lists. The size of both lists is limited to k. If a entry does not fit the list, a least recently used/seen strategy is used to replace an entry. CMP implements a *getRandomRelay* method that returns an address of a randomly selected relay from the positives list. This method is used by CDIP to build its unstructured relay overlay.

Alternatives

Both solutions for node discovery represent possibilities to implement the random node discovery. Many other alternatives exist to implement this typical case for

peer sampling. Peer sampling techniques have been investigated thoroughly in related work, e. g., [91] provides adaptive peer sampling, [4] compensates more churn, [53] provides better safety. Depending on the application scenario, the aforementioned mechanism could be replaced by one of these techniques.

4.5.6 Discussion

In some scenarios, CMP has a specific behaviour, e. g., the common- and best-case scenarios of CMP, in scenarios with churn, and when relays have relay overlay neighbors only.

CMP's Common- and Best-Case Scenario

The theoretic discussions in Section 4.3, and Section 4.4 focus on the worst-case of relay detection and the convergence time, stability and robustness of the CID agreement. The following sheds light on the common- and best-case of CMP.

In the best-case of CMP a non-relay node joins the unstructured random overlay and all neighbors have already agreed on a CID. In this case, all overlay neighbors propose the same CID and no triangle checks must be performed by the added node. The node just adopts the CID of the overlay neighbors after it has been checked by triangle checks from its overlay neighbors. As the CID agreement is stable among the overlay neighbors, the join of fewer than 50% of the quantity of nodes already in the unstructured random overlay has no influence on the CID agreement. The common-case of a relay joining the network is similar. When overlay neighbors propose different CIDs, CMP performs a triangle check between them, detects that the node is a relay, and converges.

Influence of Churn on CMP

In contrast to many P2P applications, churn has a low influence on CMP. The reason for this behaviour is that even with high churn, the CID agreement algorithm will converge to the maximum hash proposal as long as the unstructured random overlay graph is connected for a short period of time. Literature, i. e., [49], confirms this behaviour for gossip-based protocols. The majority selection of D_{MMH}, furthermore, causes that a node quickly converges to a CID as long as less than 50% join the unstructured random overlay. This allows CMP to stay stable even in case of a simultaneous join or leave of a huge amount of nodes.

Pure-Relay Connectivity Domains

CMP divides connectivity domain detection into two parts: relay detection, and CID agreement between non-relay nodes. There one case in which this becomes a problem: when a relay has only relays as its overlay neighbors. Then, there is no node that agrees on a CID. Figure 4.19 illustrates such a scenario. Relay (D) and (E) only have overlay neighbors that are relays too. Hence, CD2 and CD3 are not detected. However, each relay knows that all its overlay neighbors are relays. Hence, CMP works around this problem by placing the relay in a separate *pure-relay connectivity domain* and assigning a unique identifier to it, e. g., the node identifier of the relay itself.

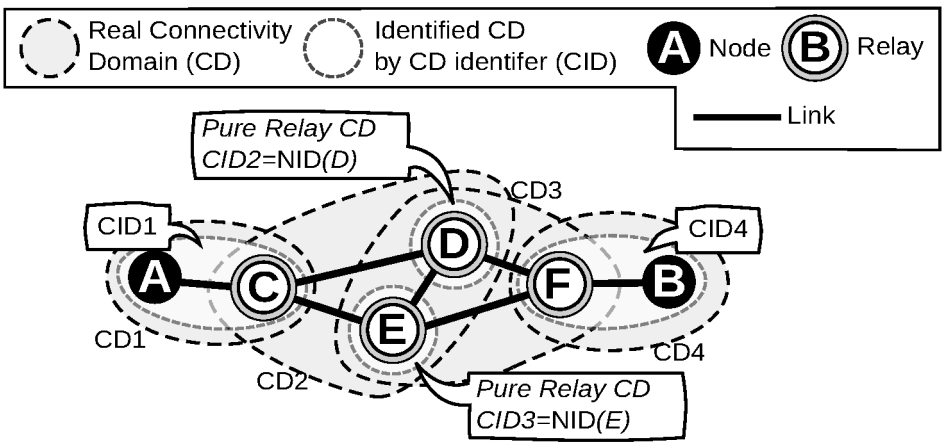


Figure 4.19 – Pure relay connectivity domain identifiers

4.5.7 Summary

This section describes CMP in detail. CMP is gossip-based and uses rounds to operate. The implementation of CMP can be done in a single main loop plus the concurrent handling of received messages, i. e., node status updates and triangle check messages. This makes CMP light-weight.

4.6 Evaluation

Based on the theoretical findings of Sections 4.3 and 4.4, several worst-case bounds according to the scenarios are known. More precisely, Section 4.3 describes that a scenario of two connectivity domains represents the worst-case for CMP. Hence, the experiments can be limited to a small number of connectivity domains with a varying relay ratio r. Furthermore, Section 4.5.6 gives an argument why CMP is robust against churn. The goal of the experiments evaluated in this section is to gain insights on the unknown factors of CMP and to confirm some of the theoretical findings, e. g., the influence of the relay ratio, convergence time, and the reactivity of CMP in case of connectivity changes.

Section 4.6.1 provides a description of the evaluation methodology in a simulation framework. Section 4.6.2 introduces necessary performance metrics. The further sections present the results of the experiments.

4.6.1 Methodology

The evaluation of CMP uses the OMNet++ simulation framework [94]. The main goal of the experiments involving CMP is to confirm the algorithmic behavior of CMP in a more realistic setup.

Since the focus is on application layer connectivity and not on lower layer protocol details, the network models provided by OMNeT++, such as the INET framework, are not used. Instead, the simulation employs a basic underlay model comprising several networks with disjoint address spaces. Each of those networks represents a connectivity domain and provides transitive connectivity. The simulation associates each node with at least one of the networks and assigns a unique

address out of the network's address space to the node. Therefore, nodes can communicate with all other nodes in the same network using their addresses. For modelling network dynamics in the experiments, the simulation migrates nodes between the networks. The simulation migrates a node by removing the address from the current network and instantaneously assigning an address from the new network. To inhibit synchronization effects in CMP, the simulation induces message delay for each network. For modelling message delay, the simulation places each node randomly on a fictive quadratic areal of size 100×100 The euclidean distance between the nodes is used to model a message delay in milliseconds to inhibit synchronization effects. All experiments use a waiting time T_{round} of one second before starting the next round in CMP. Therefore, message delays below 1 second do not have a significant effect on CMP's operation.

CMP is implemented according to the description in Section 4.5. For building CMP's unstructured overlay, the simulation uses a simplified discovery mechanism. This mechanism discovers an overlay neighbor randomly among the networks in the node's underlay. One instance of CMP runs on each of the N simulated nodes. All experiments use a *cold-start* to simulate CMP's worst-case. That means, all nodes in the network start almost at once. To avoid synchronous behavior among the nodes, each node starts with a delay chosen randomly between $[0, T_{round}]$ seconds. Each experiment uses 10 to 100 repetitions with different random seeds. The simulator calculates means of CMP's performance metrics and 95% confidence intervals. If not defined otherwise, all experiments use the default parameters according to Section 4.5.1:

Unstructured overlay

Parameter	Default
T_{round}	1s
k	20
$discoveryRate$	2
$T_{link-timeout}$	5s

Triangle checks

Parameter	Default
$T_{\Delta-timeout}$	3s
$\Delta_{retries}$	2

CID agreement

Parameter	Default
D	D_{MMH}
$R_{blacklisted}$	5 rounds
R_{RMR}	10 rounds

4.6.2 Performance Metrics

The simulation records two primary performance metrics: accuracy of the CID agreement and message overhead in terms of quantity and bandwidth consumption.

The determination of the accuracy of the CID agreement is difficult. For example, if some nodes in a connectivity domain agree on a CID—the simulation does not know if this is the correct CID. Thus, the first problem is to define which CIDs are correct. Another issue is the possibility that the nodes in two or more connectivity domains agreed on the same, i. e., ambiguous, correct CID.

The following definitions solve both problems and provides an adequate determination of the CID agreement accuracy:

Definition 4.1 (Correct CID) – The CID chosen by the majority of non-relay nodes in a connectivity domain is correct. Additionally, the highest CID is correct to break-ties when there is no majority. This allows to determine a correct CID for each connectivity domain.

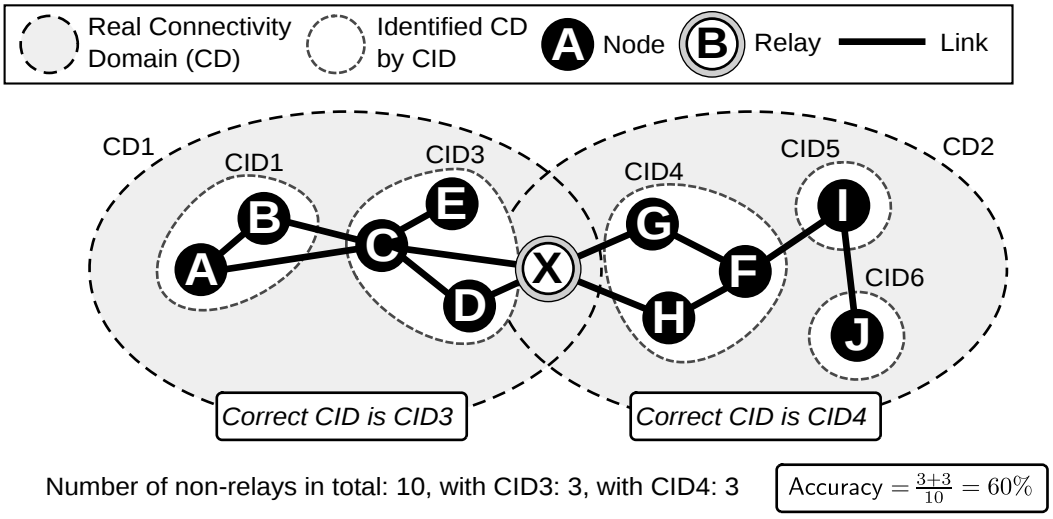


Figure 4.20 – Correct CIDs and accuracy

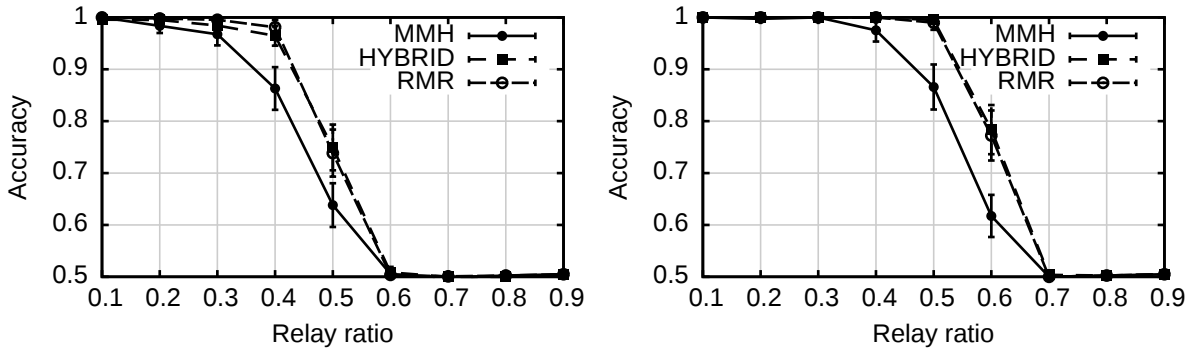
When the correct CIDs are determined, ambiguous correct CIDs among all connectivity domains need to be eliminated.

Definition 4.2 (Accurate CID) – A correct CID of a connectivity domain is accurate if no other connectivity domain has the same correct CID. If several connectivity domains have the same, ambiguous, correct CID, the CID of the connectivity domain with the least number of nodes is accurate. If the number of nodes are balanced among the connectivity domains, one connectivity domain chosen randomly has an accurate CID.

The accuracy of the CID agreement can now be defined as the quotient of the number of non-relays that agreed on an accurate CID, if available, and the total number of non-relays.

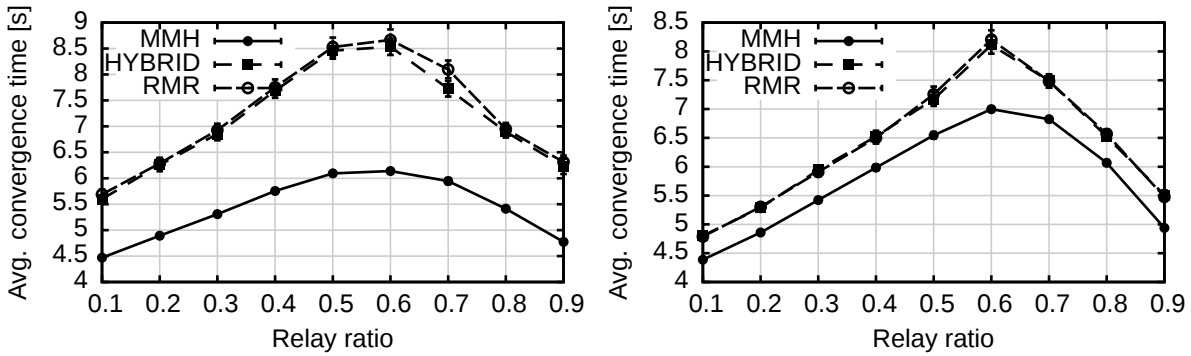
In a scenario with two connectivity domains, both comprising the same number of nodes, and all having agreed on a single CID, the accuracy is 0.5. Figure 4.20 illustrates an additional example. It comprises two connectivity domains (CD1 and CD2), 10 non-relay nodes (A,...,J), and one relay (X). Each node agrees on an CID for the connectivity domain. However, many CIDs exist until a single CID is determined. Since, three non-relay nodes (C, D, E) propose CID3, this is the correct CID for connectivity domain CD1. Respectively, CID4 is the correct CID for connectivity domain CD2. CID3 is chosen by 3 non-relay nodes, CID4 is chosen by 3 non-relay nodes, and there are 10 non-relay nodes in total. The chosen CIDs by the non-relay nodes are $\frac{3+3}{10} = 60\%$ accurate.

Additionally to the CID accuracy, CMP's bandwidth consumption may be crucial to some applications. For this reason, the simulation records the average and maximum overhead in terms of traffic, bandwidth, number of triangle checks, and number of gossip messages per node.



(a) k=20 (b) k=40

Figure 4.21 – CMP's accuracy in the worst-case scenario with varying relay ratio r with D being D_{MMH}, D_{RMR}, or D_{HYBRID} for the CID agreement algorithm and different maximum numbers of overlay neighbors $k \in \{20, 40\}$ and $N = 1024$.



(a) k=20 (b) k=40

Figure 4.22 – CMP's convergence time in the worst-case scenario with varying relay ratio r with D being D_{MMH} or D_{RMR} for the CID agreement algorithm.

4.6.3 Accuracy in case of a High Relay Ratio

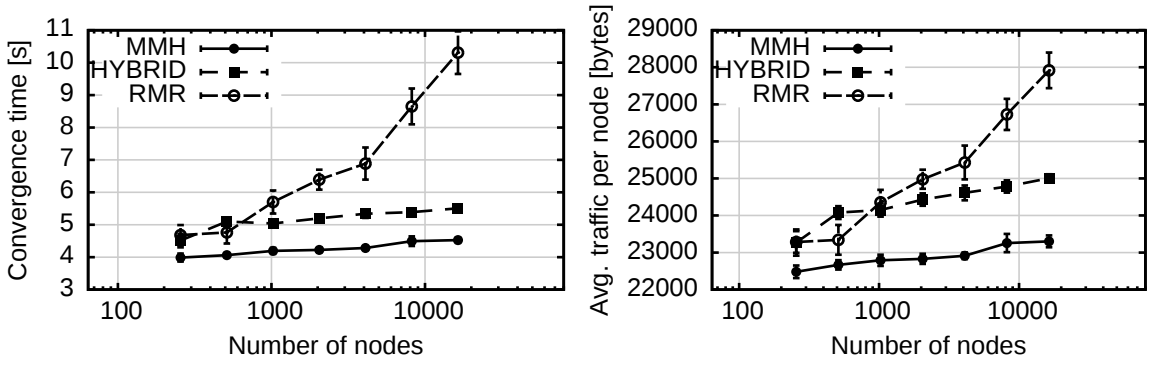
The relay ratio r is one of the most important parameters of CMP's worst-case scenario presented in Section 4.3. The worst-case scenario comprises two connectivity domains comprising the same number of nodes with $r \cdot N$ relays being in both connectivity domains, when N denotes the total number of nodes. The higher the relay ratio r is, the lower the probability that CMP can detect if a node is a relay and the less relays are detected, the more leaks remain. Consequently, the CID agreement loses accuracy. In the worst-case nodes in both connectivity domains agree on the same CID. This yields an accuracy of 50%; the absolute worst-case in this scenario. The experiments in this section explore the influence of the relay ratio on the worst-case scenario with different selection functions D being D_{MMH}, D_{RMR}, or D_{HYBRID} used for the CID agreement and different maximum numbers of overlay neighbors. Each simulation run uses 100 different seeds to ensure a sufficient confidence in the values.

Figure 4.21 shows the accuracy during the experiment with a varying number of maximum overlay neighbors $k \in \{20, 40\}$ and $N = 1024$ nodes. Figure 4.21(a) shows the accuracy as function of the relay ratio r. The curves reflect the expected behavior of CMP in this scenario. The higher the relay ratio is the less probable is a accurate CID agreement. The use of D_{RMR} in the CID agreement algorithm compensates more leaks in contrast to D_{MMH} and improves the accuracy. D_{RMR} nearly compensates a relay ratio $r \leq 0.4$ while D_{MMH} only compensates a relay ratio of $r \leq 0.3$. Furthermore, the slope of accuracy when using D_{MMH} is much steeper. The hybrid selection function D_{HYBRID} provides the same performance to the CID agreement as D_{RMR}. This suggests that D_{HYBRID} converges using D_{RMR} during the first $R_{\text{RMR}} = 10$ rounds. Figure 4.21(b) shows the same experiment with the difference that it uses a maximum number of overlay neighbors of $k = 40$. According to the insight of Section 4.4.6, a larger number of overlay neighbors makes a CID agreement more likely. The figure reflects this insight as the accuracy improves independently of the selection function.

An interesting behavior is shown by Figure 4.22. It shows the convergence time of the experiments as function of the relay ratio r for different maximum numbers of overlay neighbors $k \in \{20, 40\}$. Consider the left Figure 4.22(a) of the experiment that uses a maximum of $k = 20$ overlay neighbors: the figure confirms the longer convergence time when using the D_{RMR} and D_{HYBRID} selection functions. Section 4.4 points out the same. What is interesting on the first glance, is the peak on convergence time at a relay ratio of between $0.5 < r < 0.6$. The reason for this is CMP's relay detection algorithm. When a triangle check fails, CMP retries the triangle check two times by default. The more triangle checks fail, the longer it takes to triangle check new overlay neighbors, and, hence, the longer the convergence time. The probability of detecting a relay is high the convergence time grows with the relay ratio r. If the probability of detecting a relay decreases the convergence time decreases as well. Theoretical insight in Section 4.3 states that a high relay ratio r makes relay detection less likely. At some point, i. e., at a relay ratio $r > 0.6$, the amount of undetected relays supersedes the amount of detected relays, which reduces the convergence time. Figure 4.22(b) illustrates this behavior even more. Since a higher number of maximum overlay neighbors k improve the relay detection, the convergence time grows with the relay ratio r.

4.6.4 Convergence

The convergence time of CMP depends on two primary factors. First, the time the unstructured overlay needs for stabilization including the time the concurrent triangle checks need. Without any churn, this period is within $O(k)$. Second, the period the CID agreement needs. This time is limited by $O(\log N)$ when using the D_{MMH} or D_{HYBRID} selection functions and follows the convergence time determined using the recurrence relation in Section 4.4.4. Since CMP is a round-based protocol, the amount of traffic is proportional to the convergence time. For the evaluation of CMP's convergence, the simulation performs an experiment comprising one connectivity domain with different numbers of nodes N. Furthermore, the experiments use varying numbers of maximum overlay neighbors k. The simulation runs each experiment for 50 seconds and records the convergence time of



(a) Accuracy after convergence

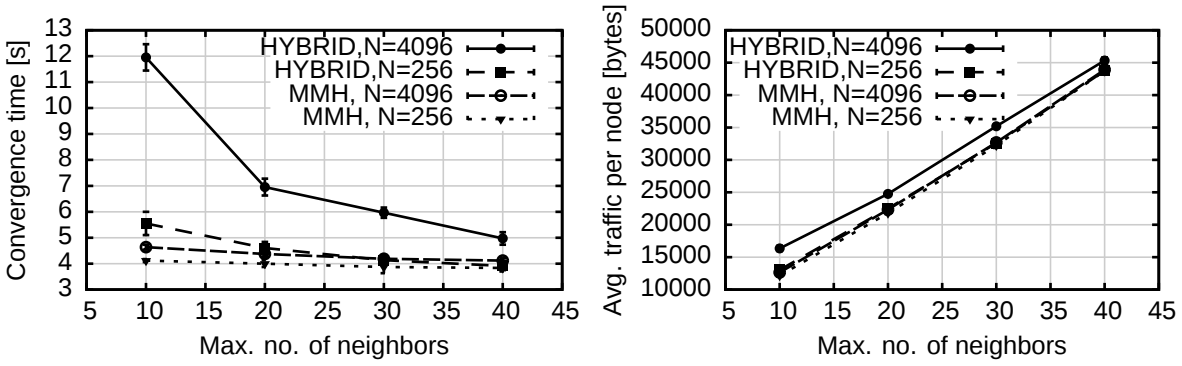
(b) Traffic generated in 50 seconds

Figure 4.23 – CMP's convergence and traffic consumption per node in a single connectivity domain with D being D_{MMH}, D_{RMR}, or D_{HYBRID} for the CID agreement algorithm. Experiments use varying numbers of nodes $N \in \{2^i : i = 8, \ldots, 14\}$ and $k = 20$.

each node. Then, convergence of all nodes within 50 seconds has been checked manually.

Figure 4.23 shows the convergence time and traffic of an exponentially increasing number of nodes N with different selection functions D being D_{MMH}, D_{RMR}, or D_{HYBRID} and $k = 20$. Figure 4.23(a) shows the convergence time as function of the number of nodes N. It proves that the convergence time develops as expected. With the use of the D_{MMH} and D_{HYBRID} as selections function, the convergence time grows linearly when the number of nodes grow exponential—confirming the logarithmic complexity. The convergence time when using D_{RMR} suggests at least a magnitude longer convergence time than with D_{MMH}. As known from Section 4.4.4, D_{RMR} can have a very high convergence time. Figure 4.23(b) confirms the proportional growth of the overhead in terms of traffic subject to the convergence time.

Figure 4.24 shows the convergence time and traffic of an increasing maximum number of overlay neighbors $k \in \{10, 20, 30, 40\}$ with different selection functions D being D_{MMH} or D_{HYBRID} and $N \in \{256, 4096\}$. Figure 4.24(a) shows the convergence time as function of the maximum number of overlay neighbors k. Two factors influence the convergence time when the maximum number of overlay neighbors grows. On the one hand, the more overlay neighbors exist the smaller is the diameter of the unstructured overlay. Hence, the CID agreement needs less time when using D_{MMH}. On the other hand, when the maximum number of overlay neighbors grows, the unstructured overlay needs more time for stabilization. The factors conflict eachother and lead to an nearly linear convergence time subject to the maximum number of overlay neighbors when using D_{MMH}. When using the D_{HYBRID} selection function this is different. D_{HYBRID} benefits from a higher maximum number of overlay neighbors much more than D_{MMH} because of the relative majority vote in D_{RMR}. This leads to a decrease of the convergence time when the maximum number of overlay neighbors grows. Figure 4.24(b) confirms the proportional growth of overhead subject to the maximum number of overlay neighbors in



(a) Accuracy after convergence

(b) Traffic generated in 50 seconds

Figure 4.24 – CMP's convergence and traffic consumption per node in a single connectivity domain using $D \in \{D_{\text{MMH}}, D_{\text{HYBRID}}\}$ selection functions for CID agreement. Experiments with varying number of maximum overlay neighbors $k \in \{10, 20, 30, 40\}$ and $N = \{256, 4096\}$.

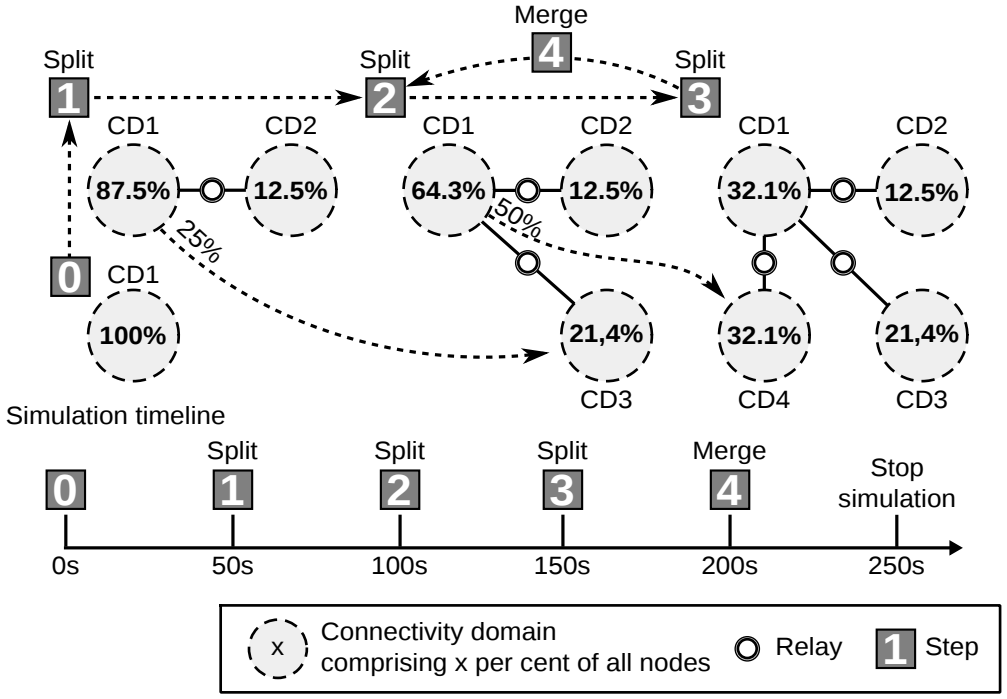
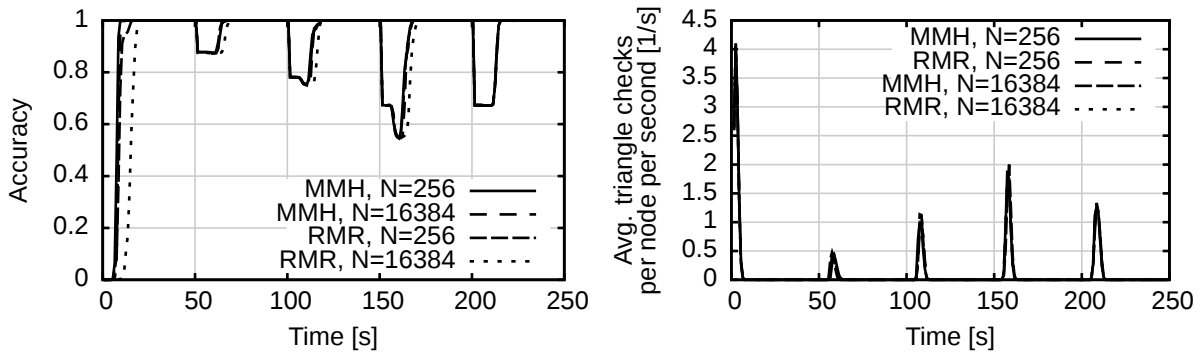


Figure 4.25 – Simulation setup for evaluating CMP's reactivity.

terms of traffic. The influence of the convergence time on the traffic is too small to be seen in the figure.

4.6.5 Reactivity to Connectivity Changes

The experiments in this section explore CMP's reactivity on massive connectivity changes. For simulating these massive connectivity changes, the simulation employs a scenario comprising splits and merges of connectivity domains. Figure 4.25 illustrates this scenario. The simulation starts at step 0 with a single connectivity



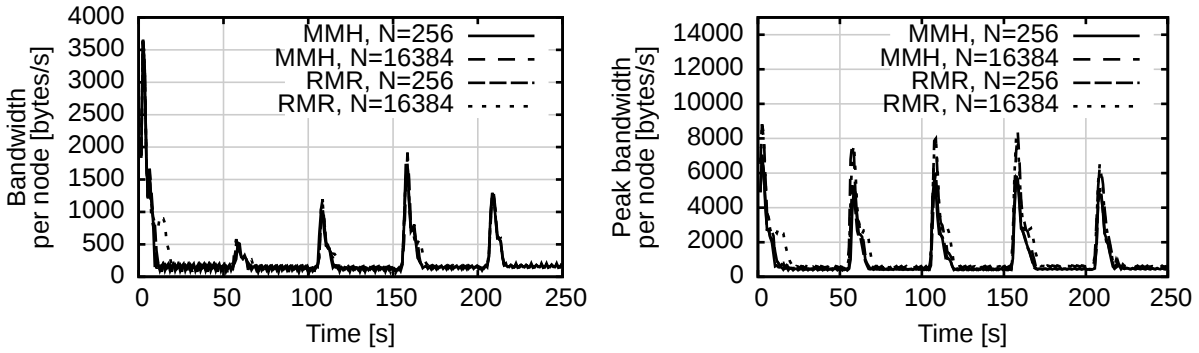
(a) Accuracy (b) Triangle checks

Figure 4.26 – CMP's convergence time and average number of triangle checks initiated by a node on connectivity changes with D being D_{MMH}, D_{RMR}, or D_{HYBRID} for the CID agreement algorithm, $N \in \{256, 16384\}$, and $k = 20$.

domain CD1 of N nodes. Then, after 50 seconds, in step 1, 12.5% of the nodes are migrated to a new connectivity domain CD2. One of the migrated nodes stays in the connectivity domains CD1 and CD2—hence, this node is a relay in CD1 and CD2. After additional 50 seconds, in step 2, 25% of the nodes in CD1 migrate to a new connectivity domain CD3, like before, one of the migrated nodes stays in CD1 and CD3. The last split, 50 seconds later, in step 3, migrates 50% of the nodes in CD1 to a new connectivity domain CD4 with one node staying in CD1 and CD4. The last step 4, after another 50 seconds, reverses the last split by migrating the nodes in CD4 back to CD1. Finally the simulations stops after 250 seconds. In all steps, the nodes migrate instantaneously to the other connectivity domain.

This scenario challenges CMP in many ways motivated by CMP's potential weaknesses. First, the scenario comprises connectivity changes involving a growing percentage of nodes in the same connectivity domain up to 50%. In case of 50%, this is the majority of all nodes and may have an influence on CMP's majority vote in the CID agreement. Second, the instantaneous migration is equivalent to a massive failure and join of nodes in CMP's unstructured overlay. This challenges the stability of the unstructured overlay at maximum. Furthermore, the CID agreement must quickly agree on a new CID for convergence to an accurate state.

Figure 4.26 shows CMP's convergence time and average number of triangle checks initiated by a node on connectivity changes with D being D_{MMH} or D_{RMR} as selection function for the CID agreement, $N \in \{256, 16384\}$, and $k = 20$. Figure 4.26(a) shows the accuracy as function of simulation time. The different curves in the figure are nearly indistinguishable—even with the major step from $N = 256$ to $N = 16384$ nodes. This indicates the fast convergence of CMP and very good scalability. Only the curve of the experiment using the D_{RMR} selection function is distinguishable. This is due to the bad convergence properties—cf. Section 4.4.4. The figure shows the agreement on a single CID for the only connectivity domain CD1 during the first 18 seconds.



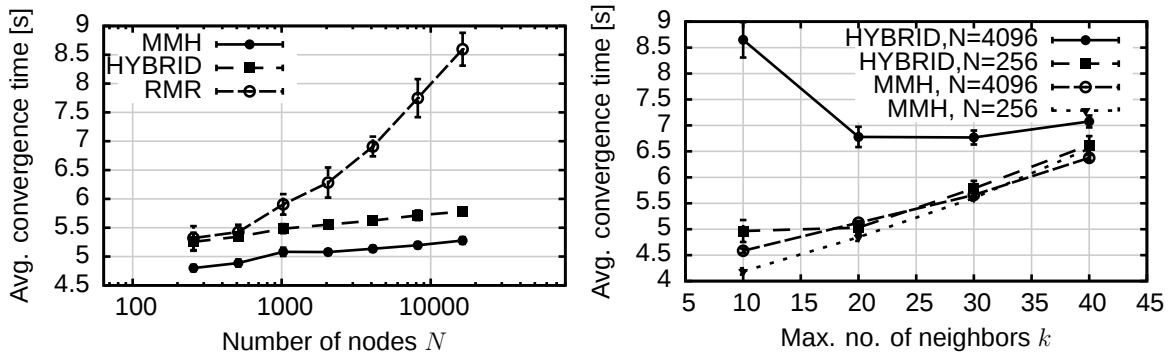
(a) Average (b) Peak

Figure 4.27 – CMP's bandwidth consumption on connectivity changes with D being D_{MMH}, D_{RMR}, or D_{HYBRID} for the CID agreement algorithm.

After 50 seconds the simulation performs the first split. 12.5% of the nodes from the single connectivity domain get attached to a new connectivity domain CD2. One node is attached to both connectivity domains CD1 and CD2. Consequently, the accuracy must drop by 12.5% because the migrated nodes still know the old CID. CMP detects the connectivity change shortly after the migration, puts the old CID on the blacklist, stabilizes the unstructured overlay, and agrees on an new CID for the new connectivity domain. CMP does this in less than 20 seconds as indicated in the figure. The same happens at 100 and 150 seconds of simulation time with 21.5% and 32.1% of all nodes migrating from connectivity domain CD1 to new connectivity domains CD3 and CD4. In both cases, the curves in the figure show a small difference to the first split. About 3 seconds after the split there is an additional drop of accuracy by up to 0.1. The cause of this drop are nodes in CD1 that accidentally detect a connectivity change—a false-positive. In the last step, the simulation migrates the nodes from connectivity domain CD4 back to CD1 after 200 seconds of simulation time. In this case the additional drop cannot be observed since links of nodes in CD1 do not fail. Consequently, the nodes in CD1 will not detect a connectivity change.

Figure 4.26(b) shows the average number of initiated triangle checks oer all nodes per second and node during the experiment as function of simulation time. CMP initiates a triangle check on a node on every new overlay neighbor. In the first seconds of the experiment all nodes initiate checks on all their neighbors—4 per second and node on average. The subsequent splits and merges require the migrated nodes to initiate triangle checks on their new overlay neighbors. Since the simulation quantifies the number of triangle checks as average over all nodes, the number of triangles checks CMP initiates is proportional to the number of nodes the simulation migrates. The gap between the actual split or merger and the first triangle check initiated after the split or merger is the connectivity change detection time.

Figure 4.27 shows the bandwidth consumption on the nodes during the experiments. Figure 4.27(a) shows the average bandwidth consumption on all nodes as function of simulation time. Like in Figure 4.26(a), the curves in the figures are



(a) Varying number of nodes N with $k = 20$ (b) Varying number of max. no. of neighbors

Figure 4.28 – CMP's convergence time on connectivity changes with D being D_{MMH}, D_{RMR}, or D_{HYBRID} for the CID agreement algorithm.

nearly indistinguishable, even with the major increase of the number of nodes from $N = 256$ to $N = 16384$. This suggests that CMP uses the bandwidth mainly for triangle checks. The number of triangle checks CMP initiates depends on the number of new overlay neighbors CMP discovers each round—independently from the total number of nodes. Hence, it is not surprising that, like the number of triangle checks in Figure 4.26(b), the bandwidth consumption correlates with the number of nodes the simulation migrates. After convergence of CMP only the keep-alive messages require some bandwidth, but less than 250 bytes/s. The maximum average bandwidth consumption stays below 4 kbytes/s. To explore the worst-case, Figure 4.27(b) shows the peak bandwidth consumption on all nodes as a function of simulation time. It shows that the peak bandwidth consumption is below 9 kbytes/s. These results suggest that CMP is lightweight in terms of bandwidth consumption.

As last observation, Figure 4.28 shows the average convergence time of CMP in this experiment. Not surprisingly, those observations are almost the same as those in Section 4.6.4. Only Figure 4.28(b), with an growing number of maximum overlay neighbors, differs compared to the figure in Section 4.6.4. The curves of Figure 4.28(b) show a linear growth of convergence time with the growth of the maximum number of overlay neighbors. The reason for that is that the stabilization time of the unstructured overlay has a significant impact on convergence time. Each time a node migrates to another network, it needs to discover new overlay neighbors. In the static case this needs to be done exactly once. This explains the impact of the growing maximum number of overlay neighbors. In summary, these results confirm CMP's good scalability—also in case of connectivity changes.

4.6.6 Summary

The experiments in this section confirm the theoretical findings in Section 4.3 and Section 4.4 in terms of convergence time and overhead. CMP detects connectivity and adapts to connectivity changes in a reasonable time, i. e., below 20 seconds. When the connectivity changes this has the most impact on the migrated nodes. The migration of more than 20% of the nodes of a connectivity domain only affects

a few non-migrating nodes. The accuracy of the detection achieves 100% within the limits Section 4.4 and Section 4.3 discuss, i. e., in case of a low relay ratio.

CMP is highly scalable and lightweight in terms of bandwidth consumption and in the considered scenarios. It never supersedes 9 kbyte/s. It is below 4 kbytes on average during convergence. After convergence CMP consumes about 250 bytes/s for keep-alive messages. CMP uses these keep-alive messages to detect connectivity changes. The main bandwidth overhead is caused by triangle checks. Note that overlay neighbors in some networks might not need triangle checks, e. g., networks with completely incompatible address formats. This will reduce the CMP's overhead significantly.

4.7 Additional Discussion

CMP is the first protocol of its kind that provides information about connectivity for P2P applications. This section outlines several cases for further work with CMP. Section 4.7.1 outlines enhancements and optimizations of CMP. Section 4.7.2 outlines new P2P applications that could emerge with CMP. Finally, Section 4.7.3 discusses security issues of CMP.

4.7.1 Enhancements and Optimizations

This section describes ideas of possible extensions and generalizations of CMP. It addresses the integration of networks with scarce resources, how connectivity domains can be identfied by a natural name, for example, "home network x" or any other name, and how new network paradigms can be supported.

Integration of Networks with Scarce Resources

Not all nodes may have the same resources. For example, a P2P application may run on a mobile phone, or even sensors. To address these devices CMP must scale with the communication and computational resources. CMP can be adapted to fit each purpose. First, it is possible to reduce the communication overhead considerably by configuring CMP that a node is not relay. Then, the node is not performing a CID agreement and triangle checks. Second, CMP can selectively reduce the maximum number of overlay neighbors k to reduce the use of bandwidth. Finally, third, it is possible to omit the CMP meachanisms on the node. In this case, the node simply adapts the identifier chosen by one of the overlay neighbors. For connecting networks with extremely scarce resources, such as sensor networks, relays can be pre-configured for the network in a way that the CD middleware must not be run on non-relay nodes.

Connectivity Domain Names

CMP identifies connectivity domains with CIDs. CIDs are not associated with any semantic information. However, in practice, networks usually have names, e. g., "home network" or "wireless LAN 1" (cf. SSID). For supporting connectivity names (CNs), non-relay nodes need to agree on a common name for each connectivity domain identifier. Depending on the application scenario, this name needs to be globally unique and there might be more than one name for the same connectivity domain. Such functionality could be provided by a name service on the relay nodes, e. g., P2PNS [6], after CMP detected connectivity. Each node inside

a connectivity domain could propose a name to the relays for the connectivity domain. Subsequently, the relays try to register the name with the name service. Furthermore, nodes could ask relays for a CID to a given name.

Content or Information Domains

New clean slate future Internet research initiatives recently introduced content or information centric networking. In these networks, addresses do not locate end-systems but content. Therefore, end-systems do not exchange messages anymore, but provide and retrieve content. To extend the design of CMP for this kind of networks, the term of connectivity needs to be transformed to content. Hence, content domains need to be considered, where all nodes in a content domain can transitively retrieve a certain content. For example, when a node X can receive content A, and X distributes the address of content A to a node Y, then Y can also retrieve content A. Therefore, a "content accessibility check" replaces the triangle check. In this case relays distribute data from one content domain to another.

This is a synthetic scenario. Relays would most likely have a conventional (end-to-end) way of communication to delegate content from one domain to another. Furthermore, CMP requires to build an unstructured random overlay. This enhancement is just stats an example how CMP could be adapted to different, future network protocols and paradigms.

4.7.2 Additional Application Scenarios

CMP offers information about sub-networks in the underlay of an overlay network and determines relays in the overlay that are able to connect them. Using this information new P2P application scenarios emerge. The following outlines additional application scenarios:

Dependable P2P applications

Assume the CD middleware interconnects connectivity domains. This enables P2P application deployment using a hierarchical design. Each layer implements the P2P service provided by an arbitrary P2P application. The local connectivity domain confines the first, or local layer. The second, or global layer, extends the local layer by interconnecting them. Figure 4.29 illustrates this application scenario by considering a chord ring hierarchy. Chord rings can be used for many applications, for example, for implementing a distributed hash table (DHT) to look up names for Internet telephony. The figure shows 4 connectivity domains (CD1-4) connected by 3 relays (4,3, and, 7) and 5 non-relay nodes. Upon these 4 domains, the application that runs on the nodes, constructs chord rings: a local one in each domain and a global one that comprises all nodes across all domains. For better understanding, the figure shows the overlay of the global ring. The text at the links of the global ring show the relays involved for communication across connectivity domains. Each chord ring provides its functionality within its domain. In an DHT P2P application, nodes participating in the local ring put data on local the local and global ring. This leads to redundancy because, e. g., in case of a DHT, it stores data locally and globally. The advantage is that even when one of the relays (4,3, or 7) fails all local rings are intact and self-healing. This highly increases dependability of the P2P applications. One disadvantage is that this approach needs more resources.

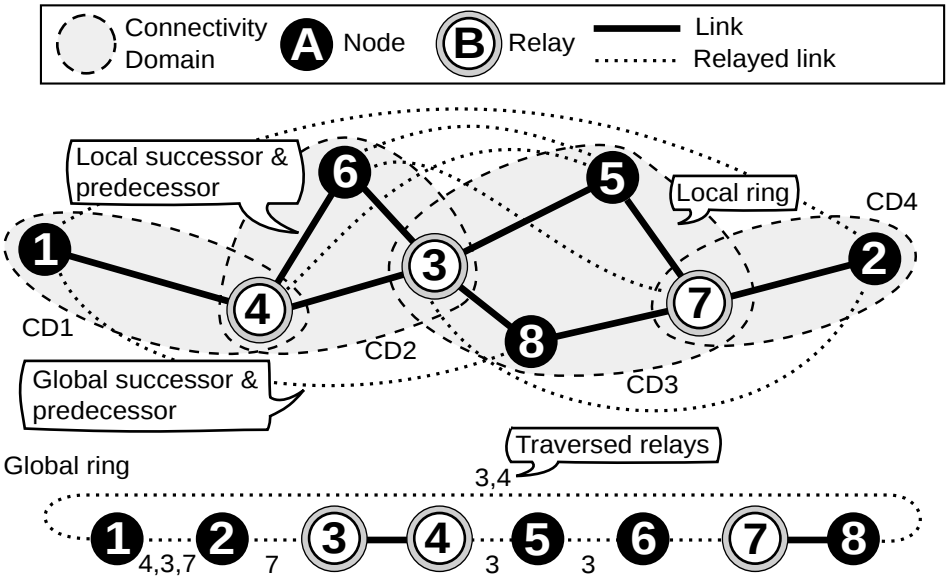


Figure 4.29 – Local and global rings to enhance dependability of P2P applications

Horizontal Composition of P2P-Overlays

Despite the hierarchical design of P2P applications presented in Section 4.7.2, the connectivity domain information can be used to compose P2P overlays horizontally. The information about connectivity domains can be combined with additional cross-layer information, e. g., the network type (ad-hoc, wireless, mobile, ...). Using this information, P2P applications can use employ different P2P overlays in each connectivity domain. For example, if one connectivity domain has native multicast support and another one is an ad-hoc network, this enables the application to use the native multicast directly, while using an P2P protocol for overlay multicast inside the ad-hoc network, e. g., TrAM [7]. Thus, the concept of connectivity domains is an enabler for horizontal compositions in the scope of P2P applications.

4.7.3 Security Considerations

Although this thesis does not address security and safety in the presence of adversarial nodes, this section provides a security analysis of potential attacks of an adversary on CMP. There are several regions that can be attacked. First, the P2P application that uses CMP by become a victim of an attack. Common attacks include the sybil- [24] or eclipse attack [79]. In case of the sybil-attack, an adversary can impersonate a large fraction of nodes inside the P2P application and can, therefore, control parts of, or the entire, network. The eclipse attack describes a stronger attack by impersonating a smaller fraction of the nodes and causing more damage to a specific target. Both have in common, that a fraction of adversaries, participating in a P2P application, influences the "normal" P2P protocol operation. This Section provides an analysis of an scenario where adversaries influence CMP, such as attacks on the random agreement, man-in-the-middle (MITM) attacks, and peer sampling poisoning.

Attacks on the CID Agreement

Attacks on the CID agreement are particularly possible during the first convergence phase of the protocol. Although the worst-case was considered, where all nodes join exactly the same time and try to agree on an identifier, if majority detection fails on all nodes, each attacker has the same influence on the CID agreement as a leak caused by an undetected relay. Depending on the used selection function in the CID agreement algorithm, in the worst-case, a single adversary can affect the outcome of the CID agreement. However, once CMP has converged, an adversary needs to impersonate about 50% percent of nodes to change the identifier of a connectivity domain. This is because CMP only re-negotiates a new identifier when 50 percent of the nodes fail or the majority dictates a different identifier. Hence, CMP is remarkably robust against adversaries, against a large fraction of adversarial nodes after convergence, but highly vulnerable during the first phase of convergence.

Man-in-the-Middle Attacks using Relays

Every node using the CD middleware can announce it is a relay. This opens the door for man-in-the-middle attacks. If an adversary announces it is a relay for a connectivity domain, at least some traffic will be forwarded to the adversary, so-called man-in-the-middle. This can be compensated by using different disjoint paths via other relays, in case the adversary just drops packets. Furthermore, a node can use secure end-to-end encryption and authentication to prevent an attacker from eavesdropping.

Peer Sampling-Poisoning

There are several known attacks on peer sampling, e. g., the mosquito attack [52], which attacks the randomness of the peer selection. Hence, the adversary can partition the unstructured random overlay or poison the list of peers. The latter means that all harmless nodes will only know nodes impersonated by the adversary in the end. This gives the adversary full control over the network. As the CMP's discovery module uses peer sampling excessively during bootstrapping phase the adversary has a fairly small window to attack the unstructured random overlay. Nevertheless, as mentioned before, there exists no solution to prevent the adversary to claim that its impersonated nodes are relays and re-route, drop, modify or eavesdrop on messages.

4.8 Summary

This chapter presents CMP, a protocol that identifies transitive connectivity for P2P networks. To this end, CMP solves two main problems: relay detection and identification of connectivity domains. By theoretical analysis, it has been shown that a node detects whether it is a relay with high probability with a triangle check per overlay neighbor and a maximum of overlay neighbors $k \geq 20$.

Furthermore, three selection functions for the CID agreement algorithm are discussed. First, D_{RMR} which is resilient against leaks, but has slow convergence time. Second, D_{MMH}, which is vulnerable against leaks, but has an exceptionally short convergence time within $O(\log N)$, when N is the number of nodes in a connectivity domain. Third, D_{HYBRID} uses both selection functions.

CMP combines the relay detection and the CID agreement algorithms to detect connectivity autonomously. CMP is scalable, converges to accurate CIDs in less than 20 seconds in the considered worst-case scenarios while using bandwidth overhead of 4 kbyte/second per node on average during convergence time. CMP's convergence time and bandwidth consumption grows logarithmically subject to the number of nodes. After convergence, CMP induces no additional overhead besides keep-alive messages.

Chapter 5

Interconnection of Heterogeneous Networks

This chapter introduces the Connectivity Domain Interconnection Protocol (CDIP) as part of the CD middleware. CDIP interconnects heterogeneous networks using the detected connectivity and provides seamless connectivity for P2P applications. This allows the deployment of existing P2P applications in heterogeneous networks with small modifications only. The main difference when deploying a P2P application that uses the CD middleware is the address format. In contrast to the deployment in the Internet only[1], the CD middleware requires the P2P application to use a tuple of a CID and an underlay address, called *CDIP-address*,

$$\langle \mathsf{CID}(x), \mathsf{addr}(x) \rangle$$

to send a message across heterogeneous networks to node x. Therefore, the CDIP-address may be a substitute for the IP address and UDP-port tuple used in current P2P applications. For further abstraction of the addressing, e. g., identifier-based, or name-based addressing, P2P applications may use a key-based routing protocol to map the node's name or identifier to an CDIP-address. One example of of such a mapping is provided by the Base Overlay in the *ariba* middleware described in Chapter 3.

Figure 5.1 illustrates the placement of CDIP in the CD middleware. CDIP is situated beneath the P2P application and provides a bootstrapping and message-based communication interface to the P2P application. The bootstrapping interface allows P2P applications to request several CDIP-addresses of nodes that already run the CD middleware. The message-based communication interface enables the P2P application to send and receive messages with CDIP-addressing.

[1] On the Internet P2P application commonly use a tuple of IP address and UDP

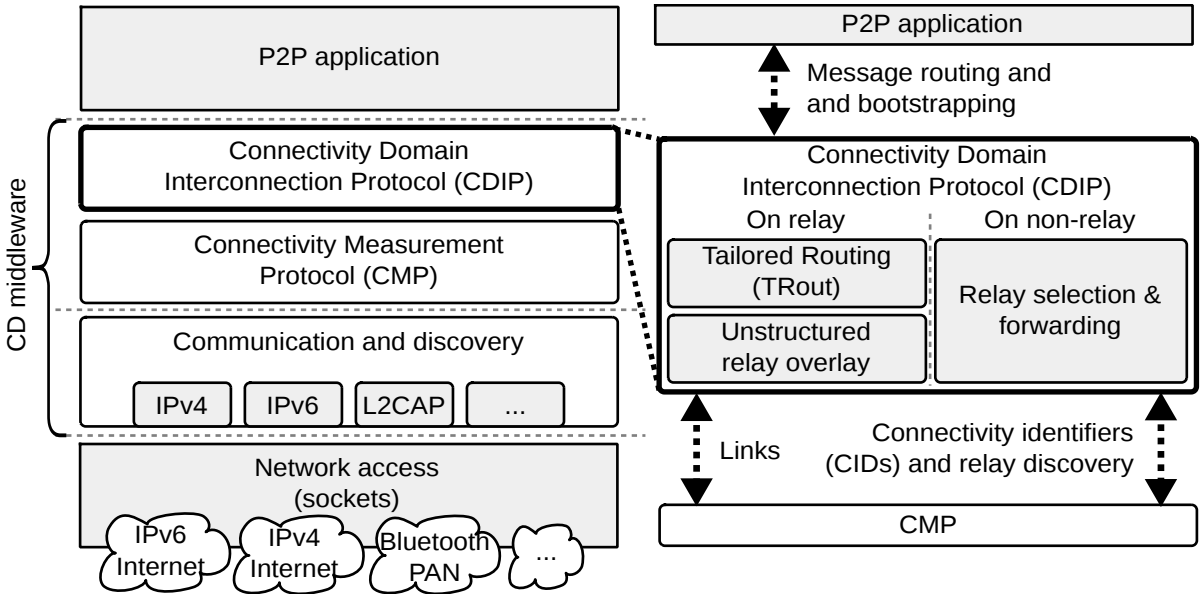


Figure 5.1 – CDIPs placement in the connectivity domain middleware.

CDIP uses the information about the connectivity domains detected by the Connectivity Measurement Protocol (CMP) which is situated beneath CDIP. Using its relay discovery module, CMP provides the connectivity identifiers (CIDs) and underlay addresses to relays in the node's connectivity domain(s) to CDIP.

When a P2P application sends a message on a node using the CD middleware, CDIP compares the CID in the CDIP-address with the CID(s) of the node. If the CID matches one of the node's CIDs, CDIP delivers the message directly using the underlay address in the CDIP-address. If the CID does not match any of the node's CIDs, CDIP adds a header to the message containing the destination and source CDIP-address. Then, CDIP forwards the message to a relay. The relays take care of routing messages across connectivity domains. For this purpose CDIP employs two main mechanisms illustrated by two layers in Figure 5.1:

Unstructured Relay Overlay

CDIP builds an unstructured relay overlay by establishing links to relays selected randomly. The construction of the unstructured relay overlay uses the same mechanisms as CMP for the unstructured overlay. In contrast to the unstructured overlay of CMP, the unstructured relay overlay comprises relays only.

Tailored Routing

CDIP employs the Tailored Routing protocol (TRout) to route messages across connectivity domains in the unstructured relay overlay. If a non-relay node wants to send a message to a node in a different connectivity domain, it sends the message to a known relay in its connectivity domain. Then, the relays route the message towards the destination connectivity domain based on the CID in the CDIP-address included in the message header. When the message reaches the destination connectivity domain, the relay in the destination connectivity domain delivers the message using the underlay address in the CDIP-address.

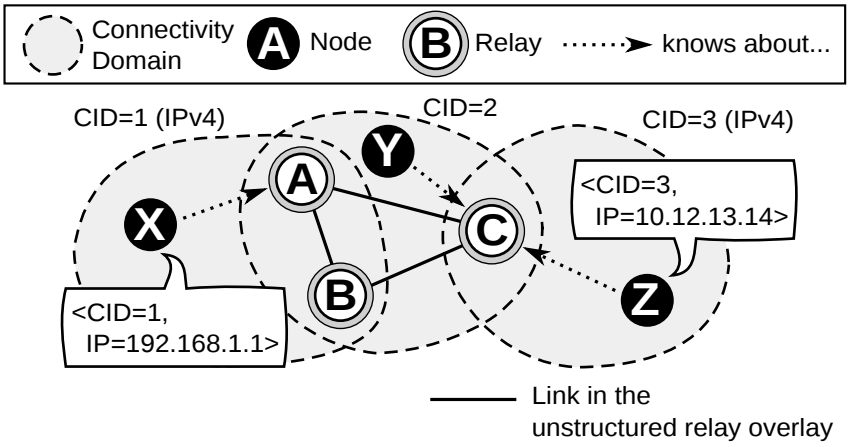


Figure 5.2 – Example of CDIP scenario comprising 3 connectivity domains, 3 relays forming the unstructured relay overlay, and 3 non-relay nodes. All non-relay nodes know at least one relay inside their connectivity domain.

The scenario depicted in Figure 5.2 illustrates three connectivity domains (CID1, ..., CID3), three relays, (A), ..., (C), and three non-relay nodes, (X), ..., (Z). The relays form an unstructured relay overlay and use TRout to route messages across connectivity domains. Each of the three non-relay nodes knows its CID and underlay address. Additionally, each non-relay node knows at least one underlay address of a relay in its connectivity domain[2]. As mentioned before, a P2P application uses CDIP-addresses to send messages to other nodes. When node (X) wants to send a message to node (Z) then node (X) recognizes that (Z) is in a different connectivity domain by comparing the CIDs in the CDIP-address tuple. Then, CDIP adds the source and destination CDIP-address to the message and forwards the message to the known relay (A). Relay (A) routes the message over the unstructured relay overlay to relay (C) in node (Z)'s connectivity domain. Finally, relay (C) delivers the message to node (Z) using node (Z)'s underlay address included in the CDIP-address.

The CD middleware needs to handle a broad range of heterogeneous networks. For evaluation of CDIP in such networks, Section 5.1 describes reasonable connectivity domain scenarios. These scenarios are relevant for today's and most likely for future networks. Furthermore, the section discusses possible routing modes in those scenarios, and defines the requirements of a appropriate routing protocol. To find an appropriate routing protocol Section 5.2 discusses related work and subsumes that the Virtual Ring Routing (VRR) protocol can be considered as a reasonable starting-point. Section 5.3 describes the Tailored Routing protocol (TRout), that fulfills the requirements defined in Section 5.1. Section 5.4 provides an evaluation of TRout using the unstructured relay overlays extracted from the connectivity domain scenarios. Finally, Section 5.5 summarizes the results.

[2]CMP provides this information.

5.1 Scenarios and Requirements

In contrast to the scenarios considered with CMP, the evaluation of a routing protocol requires more realistic and larger scenarios that describe how relays are deployed in heterogeneous networks. Section 5.1.1 provides and discusses those scenarios. They describe the access of relays to connectivity domains. Furthermore, Section 5.1.2 describes how the unstructured relay overlay built between relays. The scenarios and the unstructured relay overlay construction allows the derivation of unstructured relay overlay topologies. These serve as foundation to evaluate and design a routing protocol that is able to route messages across connectivity domains on the relays. Two routing modes may be used in these network topologies: unicast and anycast. Section 5.1.3 discusses the those modes in the context of the network toplogies. Finally, Section 5.1.4 defines the requirements of a routing protocol for use in CDIP.

5.1.1 Connectivity Domain Scenarios

In general, it is difficult to obtain scenarios which describe relays connected to connectivity domains. The reasons for that are twofold. First, multiple heterogeneous networks exist that have not yet been interconnected autonomously, e. g., Bluetooth capable mobile phones, virtual private networks (VPNs), or mobile ad-hoc networks. Second, it is unclear how the Internet and other networks will evolve, e. g., if all nodes use IPv6 and have transitive connectivity in the future, then only one connectivity domain will exist. Even related work, e. g., the Unmananged Internet Architecture (UIA) [34] that supports similar networks does not state concise scenarios.

The goal of this section is to extrapolate possible scenarios describing how relays are connected to connectivity domains. These scenarios are inspired by today's and future heterogeneous networks. The first scenario is inspired by today's scenario of deployed NATs and firewalls. The second scenario is inspired by the evolution of the Internet. Those scenarios are conceptional. Relays do not actually connect to connectivity domains but to relays in other connectivity domains. To fill this gap Section 5.1.2 provides insight how relays connect to other relays in the same connectivity domain(s) using an unstructured relay overlay.

Star Scenario

The main reasons of today's violation of transitive communication properties are personal firewalls and network address translation routers (NATs) [83] that separate a private (home-)network from the public Internet. Almost 90% of all P2P applications run behind a firewall or NAT [21]. To reach devices in the private network behind a NAT router from the Internet, the NAT router needs a virtual server rule so a device in the private network can be reached from the Internet. Technologies, like Universal Plug and Play (UPnP) [1] enable the automatic configuration of such forwarding rules.

A P2P application has to deal with these obstacles. Most of the P2P applications, e. g., μTorrent [2], use UPnP to communicate transitively with other nodes. UPnP has significant drawbacks. First, the number of possible virtual server rules

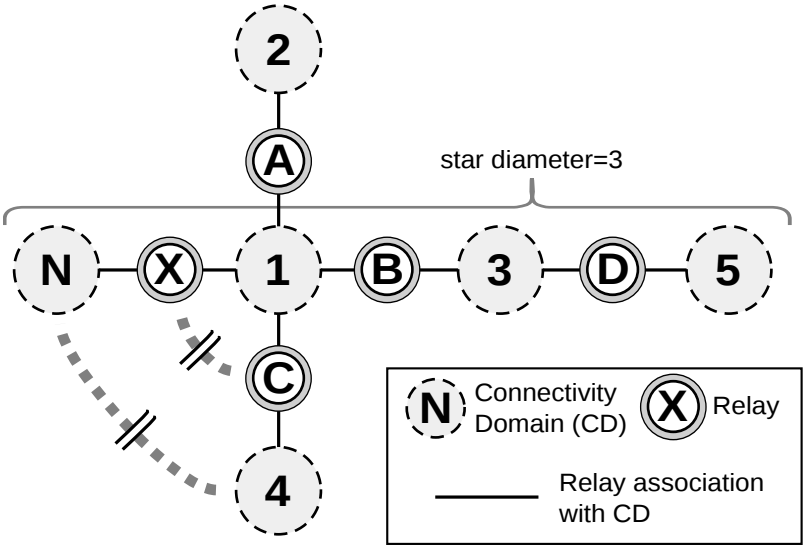


Figure 5.3 – Example of the Star connectivity domain scenario.

on home routers is limited[3]. Second, many people or administrators disable UPnP in the routers because of security issues [93].

The use of CMP allows a more general and controllable way to achieve forwarding between private networks and the Internet. Using the CD middleware only one node must have a virtual server rule in the private network. CMP detects that this node is a relay which connects the private network with the public Internet. Nodes inside the private network will automatically agree on a CID for the connectivity domain spanned by the private network. All relays and nodes directly connected to the Internet will agree on a CID for the connectivity domain spanned by the Internet. Consequently, nodes in the private network will use the relay to exchange messages in other private networks. This setup allows the derivation of the star scenario. It comprises one main connectivity domain spanned by the global Internet and thousands to millions of small connectivity domains spanned by the private networks attached to the Internet. Figure 5.3 shows an more general version of this scenario. It comprises a center, i. e., the Internet, and branches, i. e., the private networks connected to the Internet by relays. The branches in the figure show more than one private network resulting in a the star diameter of $d_s > 2$. This generalizes the scenario mentioned before with respect to additional private networks inside them. One example of such an additional private network would be a Bluetooth network created by a node.

Internet-inspired Scenario

The previous scenario is based on the consideration of current facts in the Internet. As known, the Internet is a network of networks. Each network forms an *autonomous system* (AS) connected by border gateway (BGP) routers. This is quite similar to connectivity domains. One could imagine connectivity domains connected by relays as an analog to ASes connected by BGP routers. When P2P

[3]The number of available TCP or UDP ports limits the number of virtual server rules

applications are deployed on more heterogeneous networks in the future, it is likely that connectivity domains will evolve and look like the ASes in the Internet.

The AS-graph is a model of the ASes connected on the Internet. Each vertex in the graph represents an AS and each edge two BGP routers connecting the ASes. Related work states a long list of studies, e. g., [47, 78, 29], which resemble and discuss the structure of the AS-graph. A early observation in the AS-graph indicates that its degree distribution follows the power-law. This means, there exists a majority of ASes that connect to a single other AS, while only a few ASes connect to a large number of other ASes.

The Barabási-Albert random graph model (BA-model) [5] with preferential attachment can be used to generate graphs with such an power-law distribution. The generation starts with a small, randomly connected graph. Then, it iteratively adds new vertexes and connects them to a vertex already present in the graph with an edge. To achieve a power-law degree distribution in the graph, the BA-model uses a preferential attachment strategy. This strategy connects new vertexes more likely to vertexes with a high degree than to vertexes with a low degree. More formally, the probability that the BA-model connects a new vertex a to a vertex b is

$$P(\text{“connect } a \text{ with } b\text{”}) := \frac{\deg(b)}{\sum_{x \in V} \deg(x)}$$

for a graph $G := (V, E)$, set of vertexes V, and set of edges $E \subseteq V \times V$.

The challenge is the derivation of a scenario for connectivity domains and a relay graph with similar properties. A straight-forward solution of dealing with this challenge is to use an AS-graph and model the vertexes inside the graph as connectivity domains and each edge as a relay connected to two connectivity domains. The drawback of this is obvious: each relay connects to two connectivity domains only. It is doubtful if this is always the case in reality.

As the power-law accompanies many graphs observed in real life, it is probable that this law can be applied for relays connected to connectivity domains as well. This means that most of the relays are in fact connected to exactly two connectivity domains but a small fraction of the relays connects to more than two connectivity domains. Related work does not provide observations on this matter. Hence, this thesis claims that such a scenario represents a reasonable case of future heterogeneous networks.

The following tasks describes a strategy to generate a scenario of relays connected to several connectivity domains:

1. Generate a graph $G := (V, E)$ with $|V|$ vertexes based on the preferential-attachment scheme of the BA-model. Let the vertexes V represent connectivity domains $\mathbf{C}$ and each edge $\{a, b\} \in E \subseteq V \times V$ represents a relay connected to exactly two connectivity domains. In the following $N_C := |V|$ denotes the total number of connectivity domains and $N := |E|$ the total number of relays.
2. Replace each edge with a relay vertex connected to the connectivity domains.

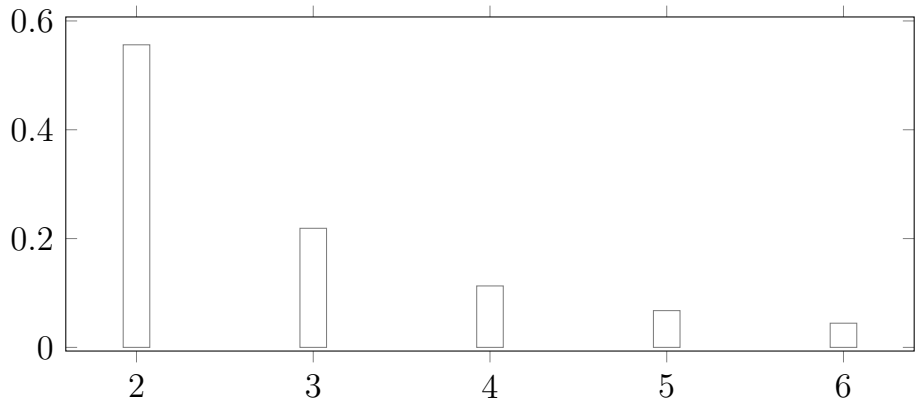


Figure 5.4 – Power-law distribution $f'(x)$ for $m = 6, \lambda = 2.3$

3. Compute a power-law distribution $f(x) := c \cdot x^{-\lambda}$ that describes the fraction of relays $f(x)$ connected to $m \geq x \geq 2$ connectivity domains. The parameter m denotes the maximum number of connectivity domains a relay may connect to.
4. Connect the relays to additional connectivity domains to fit the power law distribution given by $f(x)$ using preferential attachment.

One problem of this strategy is the determination of an appropriate power-law distribution $f(x)$, more precisely, the parameters c and λ. Several constraints allow the determination of those parameters. First, the sum of $f(x)$ nodes between 2 and m must be equal to 1, as the distribution should cover all available relays. This defines parameter c as a function of the maximum number of connectivity domains connected to a relay m and the distribution parameter λ:

$$\sum_{x=2}^{m} f(x) = 1 \Leftrightarrow c := \frac{1}{\sum_{x=2}^{m} x^{-\lambda}} \tag{5.1}$$

Applying c to $f(x)$ yields the distribution function $f'(x)$:

$$f'(x) := \frac{x^{-\lambda}}{\sum_{u=2}^{m} u^{-\lambda}} \tag{5.2}$$

The distribution parameter $\lambda > 0$ defines the steepness of the slope of the power-law's distribution. The steeper the slope of the power-law distribution, the less relays are connected to several connectivity domains.

From the current point of view, an educated guess of the maximum number of connectivity domains, a relay is connected to about $m \approx 6$ connectivity domains. Estimating that a "power-user's" node might connect to multiple private and business VPN or local network, a Bluetooth network, and the public Internet with IPv4 and/or IPv6 connectivity. However, this highly depends on the evolution of future heterogeneous networks.

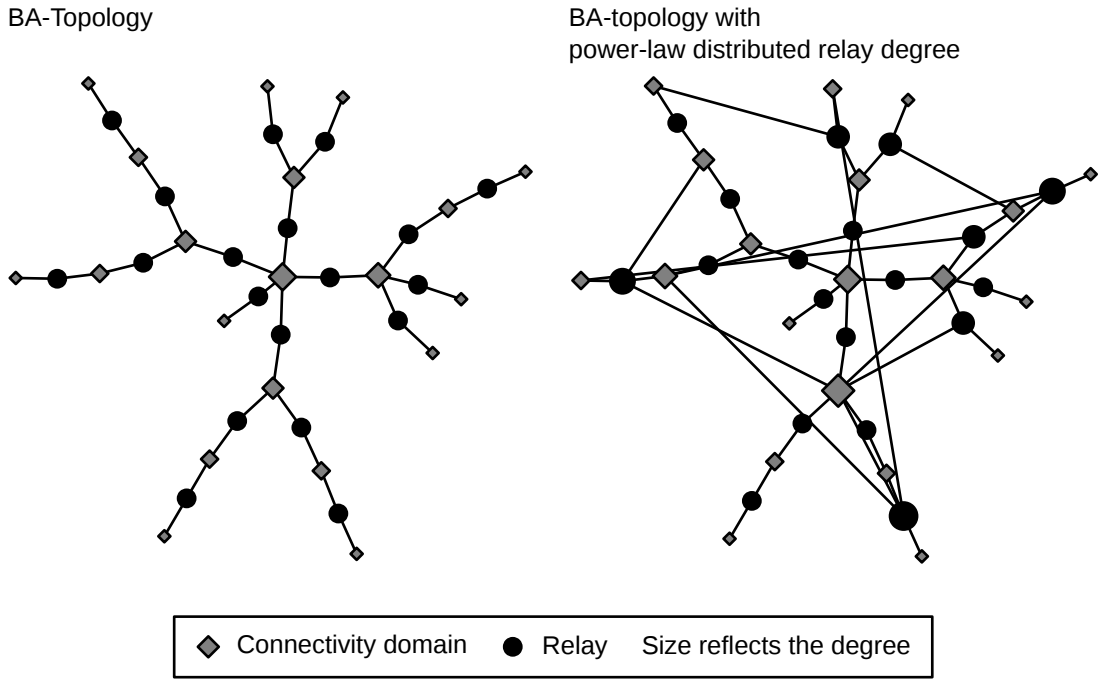


Figure 5.5 – Example of the Internet-inspired connectivity domain scenario. The left graph shows the relays connected to only two connectivity domains each using the BA-model (left). The right graph shows relay connected to additional connectivity domains following the power-law.

A reference point for the distribution's steepness is the observation of the exponent of the power-law distribution on the Internet [29]. The observations claim a value between 2.2 and 2.5. Therefore, it is claimed that a value of $\lambda = 2.3 \approx \frac{1}{2} \cdot (2.2 + 2.5)$ will most probably work for $f'(x)$ as well. Figure 5.4 shows the power-law distribution of $f'(x)$ with $m = 6, \lambda = 2.3$. The bar chart shows that 55.56% of the relays connect to 2 connectivity domains, while only about 4.44% connect to the maximum of 6 connectivity domains. Figure 5.5 illustrates a generated connectivity domain graph with relays. On the left one can see a Barabási-Albert random graph of connectivity domains where each edge has an inserted relay. On the right the relays have been connected to more than two connectivity domains following the power-law with $m = 6, \lambda = 2.3$ and preferential attachment. The next section discusses how relays are connected to relays across connectivity domains by an unstructured relay overlay.

5.1.2 Unstructured Relay Overlay

CMP exchanges underlay addresses of relays in each connectivity domain. Each node using CMP knows a set of underlay addresses of randomly sampled relays which are in a common connectivity domain. CDIP uses these underlay addresses to build an unstructured relay overlay covering all detected connectivity domains. To achieve shortest paths in each connectivity domain, ideally, this unstructured relay overlay should have full meshes in each connectivity domain. That means,

each relay maintains links to all other relay in the same connectivity domain. This is infeasible because of scalability.

To achieve a trade-off between path length and number of overlay neighbors, CDIP builds an unstructured relay overlay in the same way as CMP builds it for detecting connectivity. Each relay tries to establish links to at least $\frac{k}{2}$ other relays in the same connectivity domain and deliberately accepts link requests from other nodes. This way, it is expected that the number of neighbors does not exceed k. Detailed information about the process can be obtained from Section 4.5.2.

As mentioned before, not forming a full mesh between relays in the same connectivity domain has consequences on the path lengths in each connectivity domain. More precisely, it is known from Bollobás and Fernandes [10] and newer results from Fernholz and Ramachandran [30], that random graphs have a diameter within $O(\log N)$, when N denotes the number of nodes. The relays in the same connectivity domain build such an random graph in unstructured relay overlay.

This means, on the one hand, a shortest path traversing n_c connectivity domains would traverse exactly $n_c - 1$ relays when using a complete mesh. On the other hand, when using the unstructured relay overlay, the shortest path takes $O(n_c \cdot \log N_C)$ additional hops in each connectivity domain. The *only* way to lower the number of additional hops is to add additional overlay neighbors. This can be done in two ways:

1. by increasing the maximum number of overlay neighbors k of the unstructured relay overlay, and
2. by setting additional links between relays in the same connectivity domain on-demand, e. g., for special traffic.

When combining the unstructured relay overlay with the connectivity domain scenarios from Section 5.1.1, large-scale unstructured relay overlays topologies can be generated. Each relay in those unstructured relay overlays knows CIDs of the connectivity domains and maintains links to other relays in the same connectivity domain. The next section discusses two routing modes for routing messages across connectivity domains.

5.1.3 Routing Modes

Once the relays built an unstructured relay overlay, they are able to route messages across connectivity domains. For that, relays employ a routing protocol. This routing protocol can route messages across connectivity domains using two different routing modes:

- Anycast routing: When using anycast routing, each relay announces the CIDs of the connectivity domains it is connected to. The routing protocol then routes messages addressed to a CID to *one* of the relays that registered the CID. Ideally, the path to this relay is shorter than alternative routes to other relays that announce the same CID.
- Unicast routing: When using unicast routing, relay cannot announce the CIDs using the routing protocol. Therefore, each relay x chooses an identifier $\mathsf{NID}(x)$ to route messages from one relay to an other. Hence, CIDs are only

used to determine if a node is in the same connectivity domain. To forward messages between connectivity domains, a non-relay node must know the NID of a relay in the destination's node connectivity domain to send a message. This requires an additional mapping mechanism.

If both modes are considered separately, each has its advantages and disadvantages. First, anycast routing takes the decision which relay is used to deliver the message in the destination domain to the routing protocol. The sender cannot decide which relay to use in the destination connectivity domain. Unicast routing would allow this by explicitly addressing a relay in the destination connectivity domain. Second, using anycast routing, the sender can directly use the CIDs to send messages to nodes in a different connectivity domain. Unicast routing would need an additional step of mapping a destination CID to a NID. Consequently, it is desirable to use a routing protocol that supports both modes, namely, anycast *and* unicast routing for best flexibility.

5.1.4 Requirements

The scenario of peer-to-peer applications allows to summarize the following requirements for a suitable routing protocol:

Requirement 5.1 (Scalable) – One of the key properties of many P2P applications is scalability. That means, each node contributes a poly-logarithmic amount of resources with an increasing number of nodes. It is important that the routing protocol on the relays is scalable as well. Literature defines scalability of routing protocols by the growth of the routing-table depending on the network size. A routing protocol is *scalable*, when the size of the routing-table grows sub-linearly with the network size.

Requirement 5.2 (Adaptation-scalable) – One of the key features of P2P applications is that each node can join, leave, or fail at any time resulting in network changes. This also applies to relays. Hence, the routing protocol used on the relays must adapt quickly and with low overhead to these changes. A routing protocol is called adaptation-scalable if the communication overhead for re-stabilization grows poly-logarithmically with the number of changes.

Requirement 5.3 (Anycast and unicast) – As mentioned in Section 5.1.3 anycast and unicast support provides best flexibility for routing messages between connectivity domains.

Requirement 5.4 (Identifier-based) – CMP uses flat identifiers called CIDs to identify connectivity domains. Therefore, the routing protocol needs to handle location-independent identifiers for addressing—at least for the anycast routing (cf. Section 5.1.3)

The next section discusses appropriate routing protocols for use on the relays in the unstructured relay overlay.

5.2 Related Work and Discussion

The goal in this section is to find a suitable routing protocol for routing across connectivity domains via relays. To this end, Section 5.2.1 defines complexity

and performance metrics of routing protocols. Furthremore, it and maps them to the requirements stated in Section 5.1.4. Then, the properties of existing routing protocols are summarized. Finally, based on the complexity of the routing protocols and the requirements Section 5.2.3 subsumes that Virtual Ring Routing (VRR) is a starting-point for a appropriate routing protocol.

5.2.1 Performance Metrics and Features of Routing Protocols

This section provides an overview of complexity/performance metrics and features needed to compare the routing protocols in terms of the requirements stated in Section 5.1.4.

Complexity and Performance Metrics

Almost all routing protocols have a common denominator. They use a routing-table to route messages to other nodes. Furthermore, they use signalling messages to build these routing-tables which result in a certain message overhead. Every time a node joins, leaves, or fails inside the network, the routing protocol has to adapt some of the routing-tables. This leads to three main complexity metrics:

Definition 5.1 (Routing-table size) – The *routing-table size (RT-Size)* denotes the number of routing-table entries needed on each node to route messages. If the routing protocol is scalable the RT-size grows sub-linearly (cf. Requirement 5.1).

Definition 5.2 (Convergence costs) – The *convergence costs* denote the traffic per node the routing protocol for convergence. This includes the traffic from a simultaneous start until convergence of the routing protocol on N nodes.

Definition 5.3 (Adaptation costs) – The *adaptation costs* denote the traffic per node that the routing protocol needs to stabilize its routing-tables when $|\Delta N|$ nodes fail, leave, recover, or join. If the adaptation costs grow sub-linearly with $|\Delta N|$, the protocol is adaptation-scalable, cf. Requirement 5.2.

When the routing-tables are filled with entries and the routing protocol forwards messages, the *stretch* is one of the main performance measures that decides on the quality of the routing protocol:

Definition 5.4 (Stretch) – The (multiplicative) *stretch* denotes the factor α the routing path differs from the possible shortest path. E. g., a stretch-2 means that the routing path is twice as long as the shortest path. Another variant is *additive* stretch β. It denotes the absolute value β the routing path differs from the shortest path. If not explicitly noted differently, the multiplicative stretch is considered in this thesis.

Features

Routing protocols come with different features. To comply with the requirements (cf. Section 5.1.4), the following two features are considered:

- **Anycast support**: When a routing protocol supports *anycast*, several nodes are reachable using the same *anycast address*. When a message is sent to an *anycast address* it sends it to exactly *one* node, e. g., the closest one.

– Direct-ID support: When a routing protocol supports identifiers (IDs) directly, it does not need an additional indirection step of mapping an identifier to a location-dependent locator.

The next section uses these features, complexity- and performance metrics to provide a comparison of suitable routing protocols.

5.2.2 Comparison of Routing Protocols

In this section several protocol routing protocols are compared to find a suitable one for use on the relays. The pre-selection of the routing protocols has been done in two primary classes:

1. *Infrastructure routing:* Infrastructure networks usually have less dynamics than P2P applications. However, the autonomous systems (ASes) and the concept of connectivity domains have some similarities. Since the Internet uses BGP to interconnect ASes, classical routing algorithms, like the path-vector, BGP uses, and distance vector algorithms, are discussed. Current research in this field recently discusses practicable compact routing schemes [37, 90] to provide scalability for infrastructure routing. Since scalability is required in P2P applications the most recent approach called Disco [80] is considered.
2. *Ad-hoc routing:* Ad-hoc networks assume network dynamics close to those seen in P2P applications. This makes ad-hoc routing protocols interesting. Two protocols are highly scalable—also in case of dynamics: Virtual Ring Routing (VRR) [13] published 2006 by Caesar et al. and Dynamic Address RouTing (DART) [28] first published 2004 by Eriksson et al. The latter was selected because its core idea has been picked up again by Virtual Id ROuting (VIRO) [81] in the year 2011 by Jain et al. and adapted for use in infrastructure networks. Virtual Ring Routing (VRR) in contrast to other, similar protocols, i. e., UIP [33], PeerNet [27], SSR [39], is better understood, both theoretically and practically [14, 63].

The following briefly describes the basic functionality and properties extracted from the protocol description and evaluation results in each paper. These properties and results are not comparable one-by-one; however, they allow to see a trend which helps choosing a suitable routing protocol.

Path-/Distance-Vector (PV/DV) Routing

Routing protocols based on the path- and distance-vector algorithms disseminate route information of each node in the network and provide a shortest-path routing from a node to any other node. This allows these routing protocols to route messages with a perfect stretch of 1.0 when using a routing-table with N entries and N being the network size. On topology changes, routing updates of each joining or failing node need to be sent to all other nodes. This induces adaptation costs per node that grow proportional with the network size. Thus, these protocols are not adaptation-scalable. However, both categories of protocols can support identifiers directly. Furthermore, it is easy to support anycast; several nodes just announce the same identity while the protocol chooses the closest-path to one of them.

The main drawback of shortest-path routing is the lack of scalability. Frederickson et al. [37] and Thorup et al. [90] have proved bounds describing the trade-offs between stretch, i. e., the deviation from the shortest-path, and routing-table size. This lead to several protocol designs. One of them is distributed compact routing (Disco).

Distributed Compact Routing (Disco)

Distributed Compact Routing [80] provides scalable routing by introducing a bounded stretch of 7.0 before, and 3.0 after the the Disco sent the first message to another node. This allows a maximal growth of the routing-table within fractional power, i. e., $O(\sqrt{N \cdot \log N})$ complexity.

Disco uses a 2-steps hierarchical design to reduce the routing-table size. To create the hierarchy, Disco chooses $\sqrt{N \cdot \log N}$ landmarks in the network randomly. Then Disco provides shortest-paths between the landmarks using a standard (PV/DV) routing protocol. In result, each node knows the shortest-paths to all landmarks. Each landmark chooses a unique identifier. Furthermore, each node learns about the shortest-paths of the $\sqrt{N \cdot \log N}$ its closest neighbors. This set is called vicinity.

For routing, each node chooses an address comprising the identifier of its closest landmark and the shortest source-route from the landmark to the node. When a node sends a message to this address Disco first routes the message to the landmark. Then, Disco uses the source-route to deliver the message to the destination node. It is obvious that this detour to the landmark induces stretch. To reduce this stretch, Disco uses several short-cut mechanisms. Finally, it has been shown that the stretch is between $[1.05, 1.10]$ on practical network topologies. The evaluation results claim convergence costs that grow with $O(\sqrt{N \cdot \log N})$ (fractional power) complexity. Adaptation costs are not considered in the paper.

Disco's addresses are location dependent, i. e., comprising route information. To support location-independent addressing Disco uses a mapping scheme from an arbitrary identifier to this address. The resolution of the identifier results in stretch when Disco delivers the first message. Furthermore, the mapping scheme does not allow anycast.

One problem of Disco and path-/distance-vector routing is that they are mainly evaluated on static networks or on networks with little dynamics. This makes a rating of adaptation-scalability difficult. The main concern in the evaluations of these protocols are stretch and routing-table size to highlight scalability. A class of protocols that supports highly dynamic networks are ad-hoc routing protocols. The following introduces two popular representatives.

Dynamic Address RouTing (DART) and Virtual Id ROuting (VIRO)

The design of Dynamic Address RouTing (DART) [28] and Virtual Id ROuting (VIRO) [81] is quite similar. Both protocols cluster nodes of the network hierarchically. The result is binary tree. In this binary tree, the root represents the cluster covering the whole network and siblings form smaller clusters as subsets from the parent cluster. The leaf clusters of the binary tree contain a single node. Subsequently, the address of a node is formed using the path from its leaf to the

binary tree root. The routing-table of a DART/VIRO node comprises of $\log N$ entries to closest nodes in other clusters and are determined using a distance-vector protocol.

For constructing and maintaining the routing-tables, the evaluation results show that the convergence and adaptation costs per node follow the trend of at least poly-logarithmic complexity. Also, due to the topology-dependent addressing, the evaluation shows a stretch of $1.30 \ldots 1.60$ for small-sized networks from 100 to 1000 nodes in an ad-hoc network.

Like Disco, DART and VIRO use locators. To deal with identifiers both protocols use a mapping service that maps the identifier to the respective locator. The main difference between DART and VIRO is that VIRO has a more sophisticated identifier mapping mechanism which is more resilient against node failing or leaving. VIRO also sheds more light on the theoretical properties of the routing scheme.

One drawback of DART and VIRO is the initial clustering scheme. While DART provides a very simple protocol to cluster the nodes, VIRO only documented a centralized clustering. One problem in the clustering scheme is the imbalance of the resulting binary tree. This can easily lead to very long addresses lengths beyond 128 bit in large networks. Although DART comes up with several solutions to some of these problems, a general technique to deal with this is an open problem. Additionally, obtaining and balancing the binary tree ends up in large-scale address changes, so called, *renumbering*. This goes hand in hand with identifier to locator mapping updates and leads to high overhead. This would make an extension for anycast support very complicated.

The following section describes Virtual Ring Routing (VRR) protocol that handles identifiers directly.

Virtual Ring Routing (VRR)

The Virtual Ring Routing (VRR) was one of the first routing protocols that was inspired by DHTs. DHTs comprise a key-based routing (KBR) layer. KBR forwards data using flat identifiers in a virtual network build upon the existing infrastructure, i. e., on the application layer. One of the popular KBR protocols is Chord [87]. It builds a virtual ring ordered by the nodes' identifiers. Each node on the ring has a successor, i. e., a node with a greater identifier, and predecessor, i. e., a node with a smaller identifier. To close the ring, the successor of the node with the greatest identifier is the node with the smallest identifier and vice versa. Routing in such a ring is simple. The ring forms an convex identifier space. That is the reason why a node can forward messages greedily along the ring without worrying about running into a "dead-end" in the address space. However, the path length of the routes can be very long, more precisely, within $O(N)$. To reduce the path length, Chord adds so called fingers, i. e., additional links to other nodes as "short-cuts" through the ring. This reduces the path length to $O(\log N)$.

VRR pushes the idea of Chord "down-the-stack" from the application-layer to the network-layer. To fill its routing-tables, each node tries to discover routes to closer successors or predecessors until the routing-tables are stable. VRR has the

	PV/DV	Disco	Requirements
RT-size	Linear	$O(\sqrt{N \cdot \log N})$	Poly-log.
Convergence costs	Linear	$O(\sqrt{N \cdot \log N})$	Poly-log.
Adaptation costs	Linear	Not considered	Poly-log.
Anycast	Yes	No	Yes
Direct-IDs	Yes	No	Preferably yes
Mult.-stretch	1.0	1.05 . . . 1.10	Preferably small

	DART/VIRO	VRR	Requirements
RT-size	Logarithmic	$\Omega(\sqrt[2]{N^3})$ (on 2D-grid)	Poly-log.
Convergence costs	Poly-log.	Poly-log.	Poly-log.
Adaptation costs	Poly-log.	Poly-log.	Poly-log.
Anycast	No	No	Yes
Direct-IDs	No	Yes	Preferably yes
Mult.-stretch	1.30 . . . 1.60	Logarithmic	Preferably small

Table 5.1 – Comparison of trends by complexity and performance metrics and features of distance/path-vector (DV/PV) routing, distributed compact routing (Disco) [80], dynamic address routing (DART) [28], virtual ID routing (VIRO) [81], and virtual ring routing (VRR) [13]. Additional sources: [3, 56, 57]

.

potential to reduce the necessity of flooding routing updates because of this. Furthermore, when nodes join, leave, or fail, only routes to the virtual ring neighbors need to be repaired. This leads to a poly-logarithmic trend in convergence and adaptation costs. Recent theoretical findings by Malkhi et al. [63] show that the routing-table size grows within fractional power, i. e., $\Omega(\sqrt[2]{N^3})$, complexity when using a two-dimensional grid network topology. The small state and the direct use of identifiers to form the ring leads to a logarithmic stretch, i. e., $O(\log N/\sqrt{p})$ (p denotes the path intersection coefficient of the network), according to [63]. VRR itself does not support anycast. However, due to the direct use of identifiers for routing it should be possible to add this functionality.

Table 5.1 summarizes the complexity/performance metrics and features of the protocols extracted from the respective related work. The protocols were evaluated under various conditions, i. e., several network topologies and different network sizes were considered. Hence, the values in the table represent trends only. While complexity of the protocol are quite comparable, the stretch denotes the specific values from the respective experiments.

5.2.3 Discussion and Design Decision

In summary, the main shortcoming of the observed routing protocols is the lack of anycast functionality. The only category providing this functionality is path-vector routing. This category is out of scope due to scalability issues. Disco is an excellent choice for static networks that require scalability but no adaptation-scalability. It is not clear how Disco will perform in cases of dynamics.

This leaves two options: VRR and VIRO/DART. Both protocols are have the potential to be adaptation-scalable. However, the use of locators and the mapping

mechanism of an identifier to the locators has not been evaluated in DART and its performance is an open issue. VIRO adds this component, but still does not support anycast. This leaves Virtual Ring Routing (VRR) as one appropriate option. VRR scales with thousands of nodes and with network changes. Furthermore, VRR uses identifiers for routing directly without an additional mapping mechanism. However, the logarithmic stretch is one problem of VRR. This problem is left for further work.

The next section introduces Tailored Routing (TRout), a routing protocol inspired by Virtual Ring Routing (VRR) and Destination-Sequenced Distance Vector (DSDV) routing. Furthermore, TRout uses linearization and parallel discovery to build its routing-tables more efficiently than VRR, and provides anycast support.

5.3 Tailored Routing (TRout)

This section describes the Tailored Routing (TRout) protocol. TRout is inspired by the well-known virtual ring routing (VRR). However, in contrast to VRR, TRout has been improved as follows:

Modularity: In TRout route maintenance is split from virtual ring maintenance. This makes TRout more controllable and easier to test than VRR. Furthermore, the route maintenance module takes care of concurrency effects and handles node failures properly.

Improved stabilization: VRR only uses a discovery mechanism for building the virtual ring. In contrast, TRout uses the concept of the self-stabilizing linearization algorithm [71] to evaluate the promising features of this algorithm. These features include fast and theoretically proven re-stabilization. Furthermore, TRout uses a parallel discovery mechanism for faster convergence. Furthermore, in contrast to VRR, TRout does not flood any routing information. To achieve this, TRout first builds a virtual path and then connects this virtual path to a virtual ring.

Anycast support: Based on the virtual ring and in conjunction with the route maintenance module, TRout can easily support anycast. For this purpose, it re-uses the routes of the virtual ring and builds an anycast tree.

Before describing TRout in further detail, Section 5.3.1 states notations used to describe TRout. Section 5.3.2 gives an overview to TRout's modules. Section 5.3.3 describes the mechanisms that maintain routes and virtual links between virtual neighbors. Section 5.3.4 describes the mechanisms used to build the virtual path/ring. Section 5.3.5 presents the anycast extension to the virtual ring. Finally, Section 5.3.6 presents TRout's additional optimizations and summary of TRout's protocol parameters.

5.3.1 Preparations

TRout uses a virtual identifier space for routing like the Virtual Ring Routing Protocol (VRR) published by Matthew Caesar et al. [13]. The following describes this virtual identifier space, routes, virtual links, co-relation to the unstructured relay overlay, the virtual ring, and how messages are routed on the virtual ring.

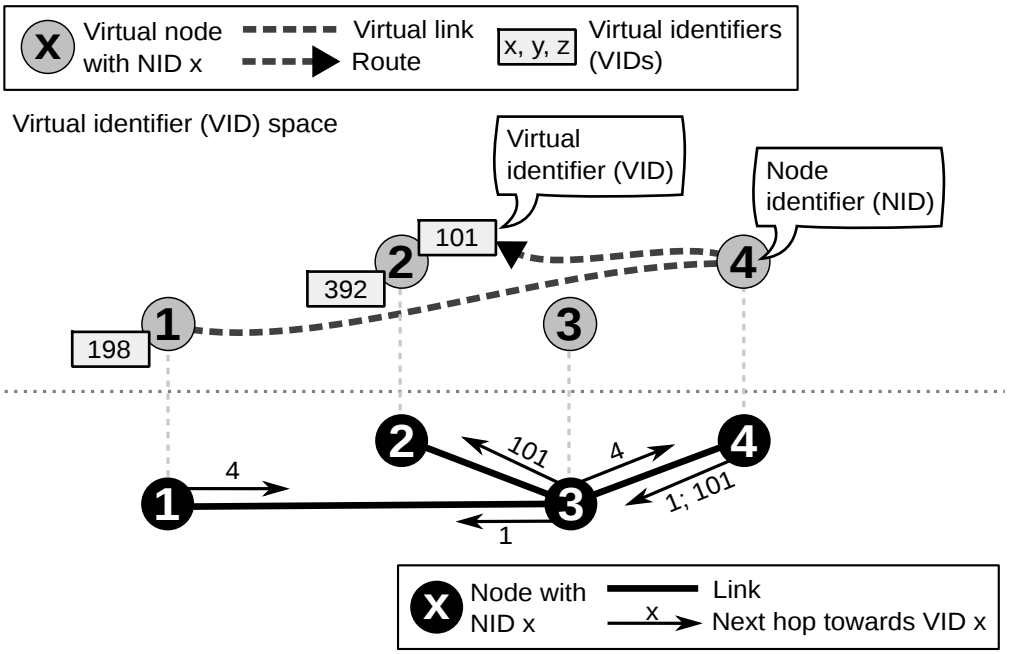


Figure 5.6 – Physical network and virtual identifier space.

The Virtual Identifier Space, Routes, and Virtual Links

The virtual identifier space $\mathcal{I} := \{\ x \mid 0 \leq x \leq 2^{128} - 1\} \subset \mathbb{N}$ is a subset of the natural numbers. Each virtual identifier (VID) $x \in \mathcal{I}$ is encoded as a 128-bit string. TRout uses these virtual identifiers for routing messages between nodes. For this reason each node x *possesses* a permanent virtual identifier chosen randomly[4] called node identifier $\mathsf{NID}(x)$. A node x may also *possess* additional virtual identifiers $\mathsf{VID}(x) \subset \mathcal{I}$. This allows TRout to let a node to receive messages addressed to any of the VIDs possessed by the node if required. This is important as a relay possesses several VIDs each equal to one of the CIDs for anycast routing, for example.

Figure 5.6 shows a network of nodes connected by links and the virtual identifier (VID) space. The virtual identifier space comprises virtual nodes which reflect the NIDs chosen by each node. Furthermore, the figure shows additional VIDs, e. g., VID 101 and 392 *possessed* by node ❷, in the virtual identifier space.

TRout routes messages to nodes which possess a VID using a routing-table (RT). Each entry in this routing-table contains some VID x and the next hop NID towards a node possessing the VID x. Figure 5.6 illustrates this by an arrow with the adjacent VID pointing to towards the next hop neighbor. TRout builds unidirectional *routes* to a node with a given VID by setting up routing-table entries (RT-entries). The figure shows one exemplary route. This route leads from the virtual node ❹ to ❷ which possesses VID 101. Two RT-entries are needed for setting up this route: one RT-entry on node ❹ pointing to its neighbor ❸ and one on node ❸ pointing to its neighbor ❷.

[4]This ensures that the virtual identifier is unique with high probability.

Two routes leading from a node possessing a VID to a node possessing a an other VID and reverse denote a *virtual link* between a pair of VIDs. The figure shows one virtual link between the virtual nodes ④ and ① resulting in RT-entries on nodes ①, ③, and ④. Two virtual nodes, e. g., virtual nodes ① and ④, connected by a virtual link are denoted *virtual neighbors* in the virtual identifier space. The process of setting up virtual links using RT-entries to form a certain topology is called *embedding*. The process of setting up one or more routes towards a node x that possesses a certain VID is called *announcement*.

Co-relation to the Unstructured Relay Overlay

TRout runs on each node in the unstructured relay overlay and, thus, TRout is able to forward messages between connectivity domains. For this reason, TRout's virtual identifier space is compatible to the set of connectivity identifiers (CIDs) defined in Section 4.4.2. Thus, like NIDs, the set of CIDs is a subset of the virtual identifier space. Each relay in the unstructured relay overlay possesses VIDs equal to the CIDs of the connectivity domains comprising the relay. TRout itself does not modify the unstructured relay overlay, e. g., does not establish new links. It uses the unstructured relay overlay as is. Therefore, for TRout the unstructured relay overlay is a network comprising nodes and links. Each node in this network has a NID chosen by TRout and several additional VIDs equal to the respective CIDs known by the relay.

Virtual Ring

The virtual ring enables TRout to route messages between nodes using their NIDs. The virtual ring is formed by connecting the virtual nodes with virtual links. Each virtual node on the virtual ring has a virtual link to a *successor* and a *predecessor*. The successor of a virtual node x is the virtual node s whose $\mathsf{NID}(s)$ is greater than the $\mathsf{NID}(x)$ of x and closest to x in terms of the euclidean distance between both NIDs, i. e., $|\mathsf{NID}(x) - \mathsf{NID}(s)|$. In the same way, the predecessor is the virtual node p whose $\mathsf{NID}(p)$ is *smaller* than the $\mathsf{NID}(x)$ of x. To close the ring, the virtual node with the greatest NID chooses its predecessor to be the virtual node with the smallest NID and vice versa. TRout's challenge is to build the virtual links for the virtual ring, i. e., to fill the routing-tables accordingly.

Figure 5.7 shows an exemplary virtual ring comprising 5 nodes (①, ..., ⑤) and their respective virtual nodes (①, ..., ⑤) connected to a virtual ring using virtual links. Furthermore, it shows the RT-entries in the routing-tables necessary to form the virtual ring. The figure contains *short-cuts* which appear when links are reflected by virtual links in the virtual identifier space. The next section will shed more light on the impact of these short-cuts.

Forwarding in the Virtual Ring

When the virtual ring is built, a node can send messages to any other node by routing messages along the virtual ring. A virtual node x with $\mathsf{NID}(x)$ routes a message towards a virtual node a with $\mathsf{NID}(a)$ by sending the message towards its virtual neighbor y with $\mathsf{NID}(y)$ closest to $\mathsf{NID}(a)$ in terms of the euclidean distance $|\mathsf{NID}(y) - \mathsf{NID}(a)|$. Intuitively, this will lead to a maximal path length within $O(N)$ as this reflects the maximal hop-count on the virtual ring. However,

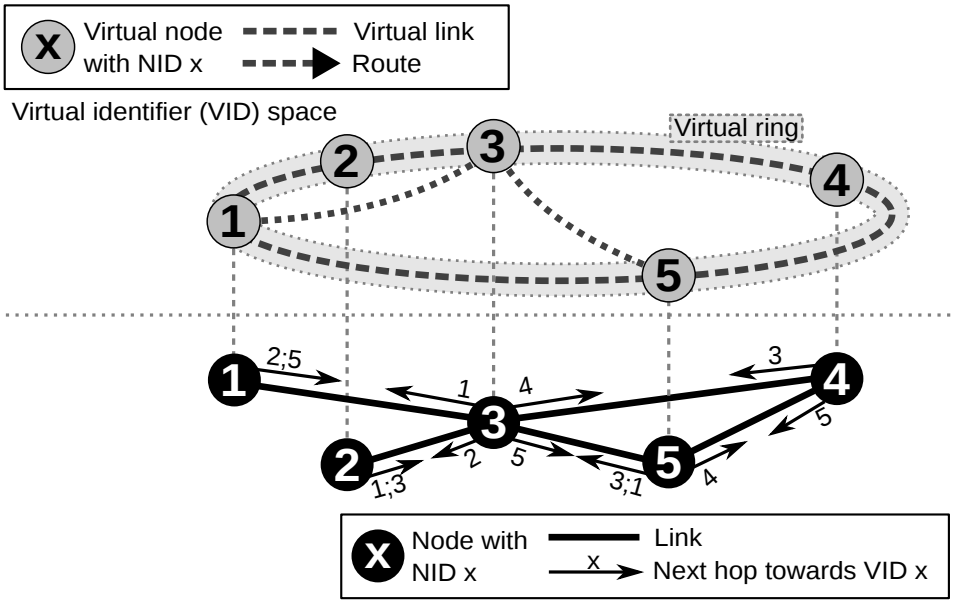


Figure 5.7 – Example of a virtual ring embedded into the network.

like VRR, TRout uses the *short-cuts* in the virtual ring to reduce the path-length significantly.

Using a Virtual Path to Build a Virtual Ring

The problem with the virtual ring is that the virtual node with the greatest NID needs to wrap around zero to find its successor on the virtual ring. The same goes for the virtual node with the smallest NID when discovering its predecessor. The wrap in the virtual ring causes VRR to converge to a so-called loopy-cycle. A loopy-cycle consists of several concatenated rings, e. g., |-1–5–9-|-3–6–10-| (the bar | denotes the wrap of each ring). For each node on a loopy-cycle the predecessor and successor is accurate locally. Globally, however, the virtial ring is not accurate because it should look like this: |-1–3–5–6–9–10-|. The loopy-cycle in the example occurs if the node with NID 9 it not able to discover the node with NID 10 as its ring successor. VRR fixes the problem by flooding routes to nodes that assume that they have the smallest NID in the network (FloodMin). Using FloodMin, the route to node with NID 3 would be flooded in the network. Consequently, the node with NID 1 would discover is real successor, i. e., the node with NID 3. This will repeat until the loopy-cycle converges to an accurate virtual ring.

One observation on routing in the virtual ring is that a convex topology is enough to successfully route messages to a virtual node. It is obvious that a virtual path is also convex; hence, is not necessary to close the ring for routing messages. The result is a virtual path comprising all virtual nodes sorted by their NIDs. The advantage of the virtual path is that it simplifies the protocol and does not need to flood the network.

To obtain a virtual ring afterwards, TRout first builds a virtual path and subsequently connect the ends of the virtual path to a virtual ring. Figure 5.8 shows an exemplary virtual path embedded into the network. The difference to the virtual

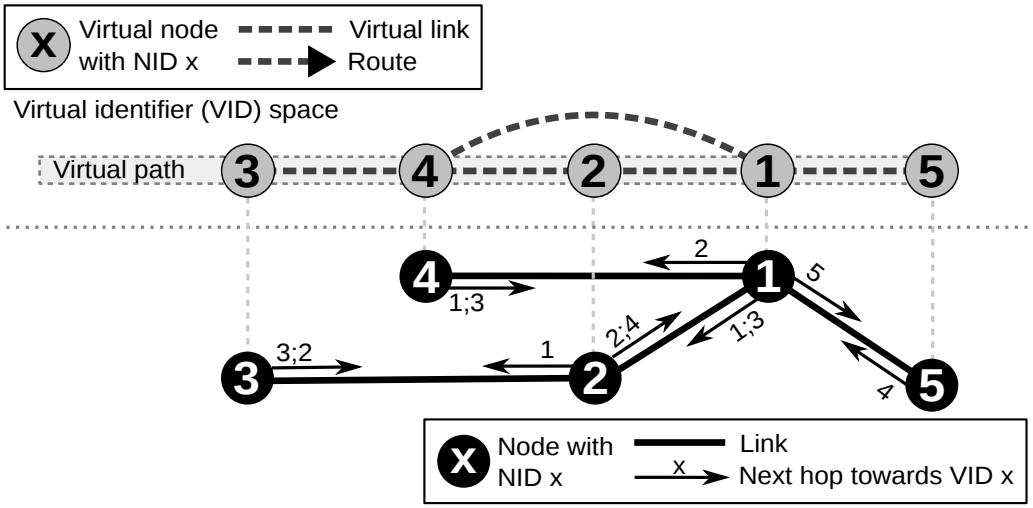


Figure 5.8 – Example of a virtual path embedded into the network.

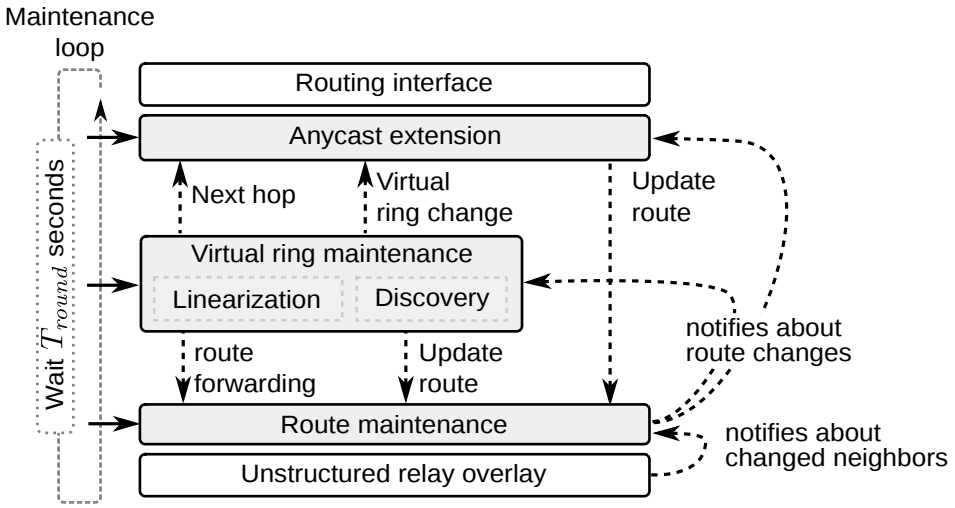


Figure 5.9 – An overview of TRout's architecture.

ring is the missing virtual link between the virtual nodes with smallest and greatest NIDs.

The following sections describe how TRout's effectively builds and maintains its virtual ring. Furthermore, they describe an anycast extension that uses the virtual ring to form an anycast tree.

5.3.2 Overview of TRout

This section gives a general overview of TRout. Figure 5.9 illustrates this overview. As part of CDIP, TRout internally provides a routing interface that allows sending messages addressed to a known VID. Furthermore, it allows to register several anycast-VIDs, i. e., the CIDs of CMP. Below the interface two main modules provide the functionality of TRout.

The virtual ring maintenance module builds the virtual ring using one of the two virtual ring construction mechanisms: linearization and discovery. These two mechanisms use two primitives for establishing and maintaining virtual links in the virtual ring: route updates and route forwarding. The route maintenance module provides this functionality. The route update primitive allows adding new and updating existing RT-entries. The route forwarding mechanism allows to build new routes and virtual links using existing ones.

When virtual links or routes between virtual nodes have been established, the route maintenance module handles virtual link failures and notifies the anycast, and virtual ring maintenance module about newly added or removed routing-table entries. The unstructured relay overlay beneath the route maintenance module provides the network for TRout and the information about failed or new neighbors.

Based on the virtual ring the anycast extension module provides the required anycast functionality. For this purpose, it uses the next-hops in the virtual ring to setup additional RT-entries using the route maintenance module. The anycast extension needs to adapt to the RT-entries of the virtual ring. For this purpose, the virtual ring maintenance informs the anycast extension module on any change of the successor or predecessor in the virtual ring.

TRout itself runs in a single loop and is *round-based*. In this loop each module performs its maintenance mechanisms every T_{round} seconds. Figure 5.9 illustrates this maintenance loop on the left.

The following describes the mechanisms of TRout in detail. The description starts with the modules that are most important for TRout: maintenance of the routing-table in Section 5.3.3 and maintenance of the virtual ring in Section 5.3.4. Section 5.3.5 describes the anycast extension which uses the virtual ring and route maintenance modules. Additionally, Section 5.3.6 presents several optimizations of TRout and summarizes its parameters. Implementation details of TRout are available in Appendix B.2.

5.3.3 Route Maintenance

The route maintenance module keeps a routing-table comprising RING and ANYCAST routes and provides two primitives:

1. *Route update* which allows adding new and updating existing RT-entries, and
2. *Route forwarding* which allows the extension of a route to some other virtual node.

Furthermore, it handles route teardowns efficiently when links/nodes fail or routes are not needed anymore, e. g., in case they are not part of the virtual ring anymore. For this purpose, the route maintenance module maintains a route usage record for each route.

The route maintenance module is based on the destination-sequenced distance vector (DSDV) protocol. DSDV uses sequence numbers to solve the problems of count-to-infinity and routing loops seen in distance-vector routing. The reasons for using this protocol as basis are manifold. First, DSDV allows to flood the network on-demand for a shortest-path between two nodes. Second, DSDV eases

route maintenance and has the potential to provide shorter virtual links because DSDV prefers shortest paths. Finally, DSDV has a lower bandwidth overhead compared to path-vector protocols because they do not have to transfer complete path-vectors. The following sections describe the primitives of updating routes and route forwarding.

Routing-Table Entry and Route Updates

A routing-table entry (RT-entry) of TRout's routing-table comprises 7 primary fields:

1. *Route type:* states the kind of the route, i. e., if the route is part of the virtual ring (RING) or part of the anycast extension (ANYCAST).
2. *Virtual identifier* VID*:* denotes the destination VID.
3. *Owner*-NID *(optional):* denotes the owner of the route. The field is used only in conjunction with a ANYCAST-route to differ an equal VID possessed by several nodes.
4. *Distance:* denotes the number of hops to the destination.
5. *Sequence number:* denotes the current sequence number of the route. The sequence number is a 16-bit integer.
6. *Next-hop:* denotes the NID of the next-hop node towards the node possessing the destination VID.
7. *Route usage record:* denotes a set of NIDs of the neighbors that rely on this RT-entry.

The route update derives directly from the contents of a RT-entry. It only differs from a RT-entry in the next-hop field and route usage record for two reasons: first, the next-hop field is not needed by a recipient because the node knows where the update message came from. Second, the route usage record is valid locally only.

When the route maintenance module receives a route update[5], it first checks whether it has an existing RT-entry matching the VID and, when applicable, the owner-NID. If no RT-entry has been found, the route maintenance module accepts the route update: the route update is turned into a new routing-table entry[6] by copying the contents of the route update to the new RT-entry. Furthermore, the distance is increased by one the next-hop field is set to to the NID of the node the route update came from.

If an entry matching VID and owner-NID already exists, the route maintenance module only accepts route updates with greater sequence number or smaller distance, i. e.,

1. the sequence number of the update is greater than the sequence number in the RT-entry, or
2. the distance in the update plus one is smaller than the distance in the RT-entry.

[5] A route update initiated by the virtual ring maintenance to add a trailing route back to the sender of a discovery message, for example.

[6] Adding a RT-entry is just a special case of a routing-table update.

This solves the count-to-infinity problem identically to the DSDV protocol. If the topology changes or routes become longer, the RT-entry needs to be updated with a new sequence number to get accepted by nodes. The sequence number is initialized with 1 and is increased on each topology change. TRout currently does not handle a sequence number overflow. However, this can be handled, e. g., by accepting a lower sequence number when the current sequence number approaches $2^{16} - 1$.

The routing-table also contains self-RT-entries, i. e., RT-entries having VIDs possessed by a node with a distance of zero and the current sequence number. The linearization and discovery mechanisms use these self-RT-entries to forward a route to one of the node's VIDs to other nodes. Furthermore, the self-RT-entries keep track of the sequence numbers for each VID the node possesses. To increase readability, figures in this Section only show self-RT-entries if they are actually needed.

Route Forwarding

Route forwarding allows a virtual node to forward a route to another virtual node. To achieve this, the route maintenance module extracts a route update from its routing-table. Let this route lead to virtual node u. Then, it forwards the route update using a route forwarding message along a known route to virtual node v. Each node on the path to virtual node v updates its routing-table with the route update included in the message. Hence, when the route forwarding messages reaches virtual node v, the node knows a route to virtual node u. This primitive eases the establishment of new virtual links between virtual nodes, e. g., during linearization and discovery.

Figure 5.10 illustrates an example of the route forwarding mechanism. The figure comprises the 5 nodes (❶, ..., ❺), and the respective virtual nodes (❶, ..., ❺), in the virtual identifier space. Virtual node ❶ initially knows two routes: one to virtual node ❺, and one to virtual node ❷. Route forwarding allows virtual node ❶ to forward its known route to virtual node ❷ to virtual node ❺. After forwarding the route, virtual node ❺ also knows a route to virtual node ❷. For this purpose, virtual node ❶ sends an route forwarding message along the route towards virtual node ❺. This message contains the route update contents of virtual node's ❶ RT-entry leading to virtual node ❷.

In step 1, node ❸ receives the route forward message and adds a RT-entry leading to virtual node ❷. Then, it forwards the route forward message to node ❹. When node ❹ receives this update in step 2, the node realizes it already knows a shorter route to virtual node ❷ and updates the route forward message with its RT-entry and forwards it to node ❺. In step 3, node ❺ adds a new RT-entry leading to virtual node ❷. This results in a route from virtual node ❺ to ❷ with node ❹ being the only intermediate node. The route has been successfully forwarded.

Route Usage Records

The route usage records are important for three tasks. First, they allow to notify nodes that announced a route that it has been partially torn down, e. g., in case of a

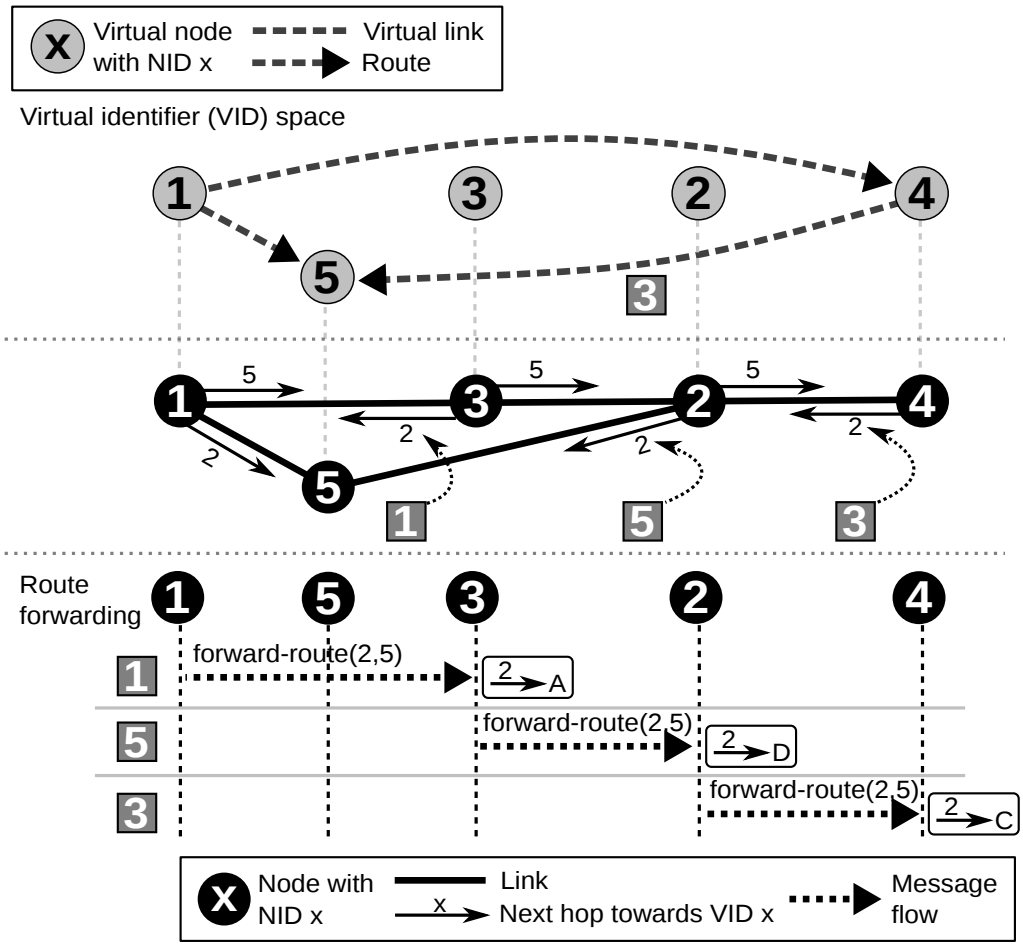


Figure 5.10 – Example of the route forwarding mechanism in TRout's route maintenance module.

node failure. Second, they allow TRout to perform consistency checks as presented in Section 5.3.6. Third, they allow the efficient removal routes that are not part of the virtual ring, e. g., routes that are only needed until the virtual ring has been stabilized.

The route usage record in each RT-entry keeps a set of neighbors whose route uses the RT-entry. For this purpose, TRout informs a node's neighbors about the removed or required RT-entries. More precisely, the following invariant describes the route usage record: when a node u knows a route to VID x with the next hop being node v, then node v must know a route to VID x as well. Furthermore, node v's route usage record of the RT-entry to VID x comprises node u's NID. The route usage records causes TRout to use some additional traffic which is almost proportional to the route updates received. Thus, it does not have an influence on scalability. Furthermore, it is possible to compress the usage information message to the neighbors by using a bloom-filter, if required.

Figure 5.11 shows the route usage records of NID1 on the nodes ❶ and ❸. If one of the nodes' neighbors routes to NID1 traverses a node this is kept in the route

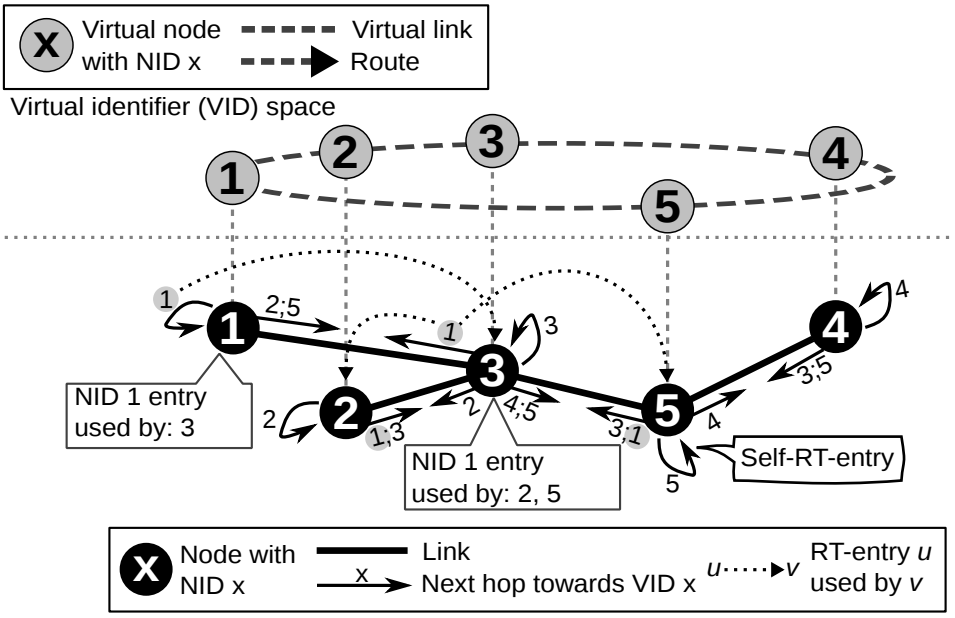


Figure 5.11 – Example of a route usage record of TRout's route maintenance on a virtual node with NID 1.

usage record. Node 1 knows that its neighbor 3 uses the RT-entry leading to node 1's virtual node 1. Node 3 knows that nodes 2 and 5 use the RT-entry leading to node 1's virtual node 1.

Route Teardowns and Notifications

When a a link between nodes fails, invalid routes using this link need to be removed from the routing-tables. The route maintenance module takes care of this problem by forwarding teardown messages to the neighbors that hold routes using the link by using the route usage records. When a node receives a teardown message for a specific route, it forwards this message to other neighbors using this route until all invalid RT-entries are removed from the network.

Furthermore, the destination nodes are notified that one of their routes has been partially or completely torn down. Notified nodes and neighbors affected by the failure increase the sequence number of all self-RT-entries in their routing-table and notify all other modules to react properly, e. g., fixing the virtual ring. All routes removed from the routing-table will get blacklisted for $R_{timeout}$ rounds to prevent old routes from reappearing. The blacklist sensitive to the sequence number. This means, a new route with a greater sequence number than the one blacklisted is accepted. The blacklist is important as the virtual ring stabilization does not care about torn down routes and, hence, may re-establish partially broken routes. The blacklist effectively inhibits this.

Figure 5.12 shows a link failure between the nodes 3 and 5. In consequence, TRout checks the route usage record of all RT-entries on nodes 3 and 5 and sends notifications to all nodes whose route has been affected by the failure. In this case, 5 sends a notification to virtual node 4, and node 3 to virtual node

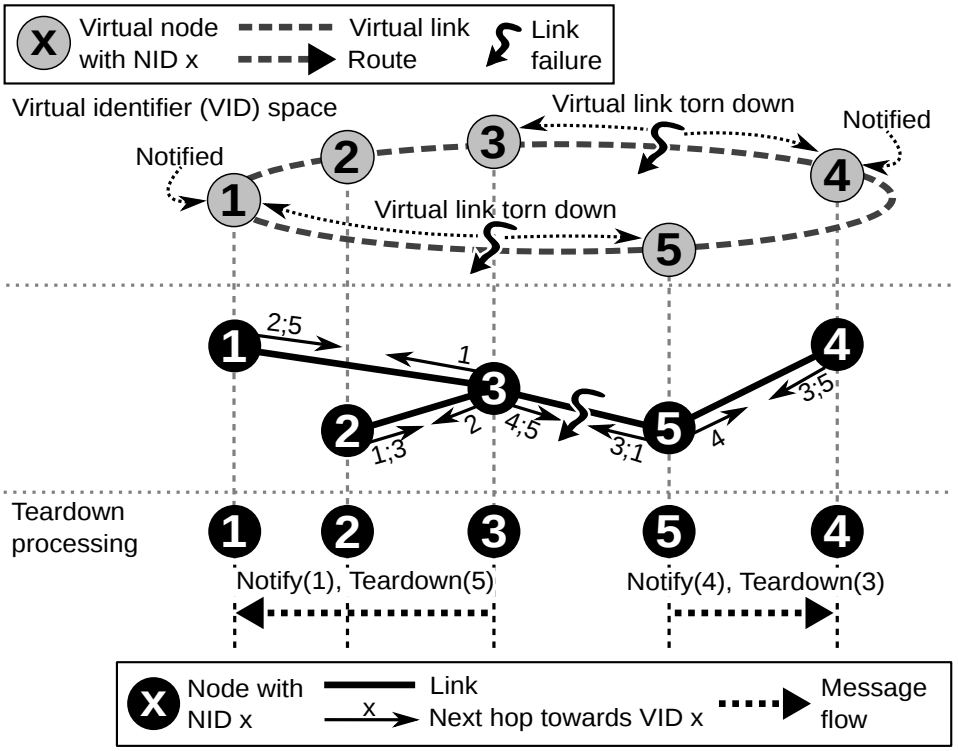


Figure 5.12 – Example of a route teardown by TRout's route maintenance.

1. Additionally, nodes E and C send teardown messages to nodes that use a route traversing the failed link, i. e., to virtual nodes 3 and 5.

5.3.4 Virtual Ring Maintenance

Section 5.3.1 states that the virtual ring can be build using two steps: first, build a virtual path, and second, connect its ends to a virtual ring. TRout does this exactly this way. It uses two mechanisms either discovery or linearization to build a virtual path. Then, TRout connects both ends to build a virtual ring.

Both mechanisms, first, add the node's own NID to its routing-table, i. e., a self-RT-entry. Second, TRout informs the node's neighbors about its virtual node using a route update of the route maintenance module. This way, the network topology is reflected by virtual nodes and virtual links in the virtual identifier space. Both mechanisms use the initial virtual neighbors to subsequently build the virtual path. The following describes both mechanisms in further detail.

Discovery

The discovery mechanism of TRout is almost identical to the one used in VRR. TRout, however, only builds a virtual path instead of a virtual ring. The advantage of the virtual path is, that it does not converge to loopy cycle as mentioned in Section 5.3.1. The VRR protocol used an algorithm that floods routes to the node with the smallest NID to fix this problem. This has a significant drawback. When the node with the smallest NID fails, the network must be flooded again—this reduces the adaptation-scalability.

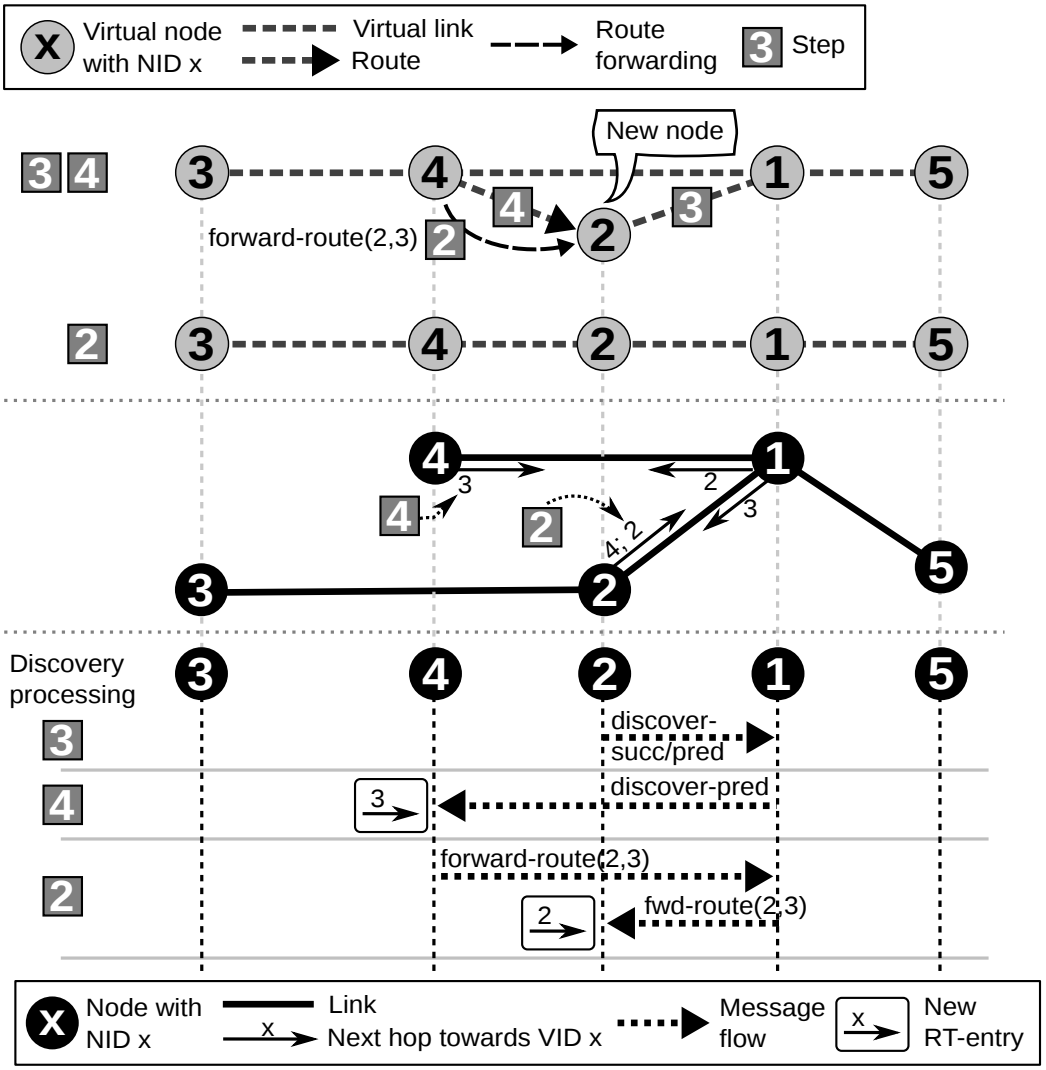


Figure 5.13 – TRout's virtual ring maintenance: example of one discovery message.

Starting with the initial virtual neighbors, the discovery mechanism successively attempts to find a closer predecessor and a closer successor. The discovery mechanism does this using two discovery message types. One message's goal is to discover a closer predecessor, the other ones goal is to discover a closer successor. Both message types include the route update information of the node's self-RT-entry to enable other nodes to add a RT-entry towards the sender of a discovery message.

Each node x sends two messages of each type to the same neighbor chosen randomly each round. The recipient of a discovery message remembers a route back to the sender of the message by adding a RT-entry using the update information inside the discovery message. This way each discovery message leaves a trailing route behind. Then, the recipient routes the message towards the closest predecessor/successor route available in its routing-table. If the recipient of the discovery message is the closest predecessor/successor itself, it establishes a virtual link to virtual node x by forwarding its self-RT-entry route towards the sender. Natu-

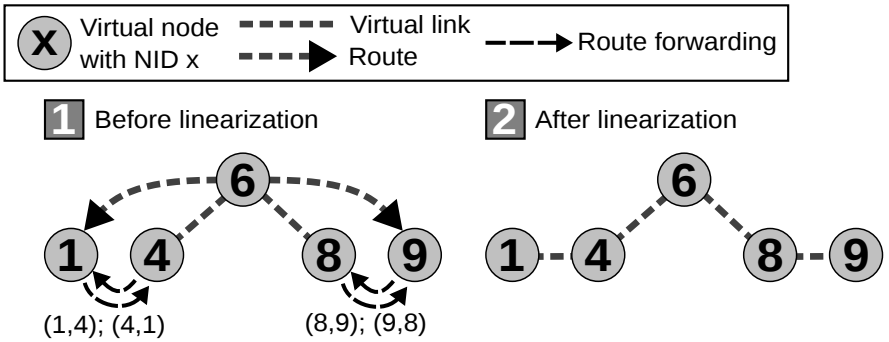


Figure 5.14 – TRout's virtual ring maintenance: example of one linearization step.

rally, virtual nodes that are already a successor or predecessor do not establish a new virtual link. The periodic attempts of each virtual node to discover a closer predecessor/successor leads to the virtual path after a finite amount of time. The proof of correctness and upper bounds of time complexity can be derived from [15] and [19].

Figure 5.13 illustrates an example of one discovery step. It comprises 5 nodes (❶, ..., ❺) with their respective virtual nodes (❶, ..., ❺). Virtual node ❸ has just started integrating itself into the virtual path. For this purpose, it sends two discovery messages to node ❶ in step 1. Node ❹ is a predecessor of ❸. Since virtual node ❹ already has a virtual link to ❸ no further action is required. In step 2, virtual node ❹ routes the predecessor discovery message to virtual node ❷. This is the predecessor of virtual node ❸. Therefore, in step 3, node ❸ forwards the route to itself back to virtual node ❸. This results in an virtual link between nodes ❸ and ❷. So, node ❸ is correctly integrated in the virtual path.

Linearization

Linearization is a self-stabilizing algorithm introduced by M. Onus et al. [71] in 2007. Linearization sorts a graph comprising vertexes addressed by an identifier to a straight line sorted by the identifier in ascending order. Self-stabilizing means that the algorithm can recover from any state. Furthermore, [71] shows that linearization needs $O(\log N)$ iterative steps to converge. Linearization splits the virtual neighbors of node with $\mathsf{NID}(x)$ into two sets:

- the set of left neighbors, i. e., virtual neighbors having a NID smaller than $\mathsf{NID}(x)$, and
- the set of right neighbors, i. e., virtual neighbors having a NID greater than $\mathsf{NID}(x)$.

Without loss of generality, let a be a new left neighbor on virtual node x. Then, TRout establishes a virtual link between $\mathsf{NID}(a)$ and the closest virtual neighbor z of x on the right hand side of a, i. e., $\mathsf{NID}(a) < \mathsf{NID}(z)$. The virtual node z is called virtual node a's linearization neighbor. The link is established by forwarding the routes between the virtual nodes a and z on virtual node x in both directions. Linearization of a new right neighbor works in the mirrored way.

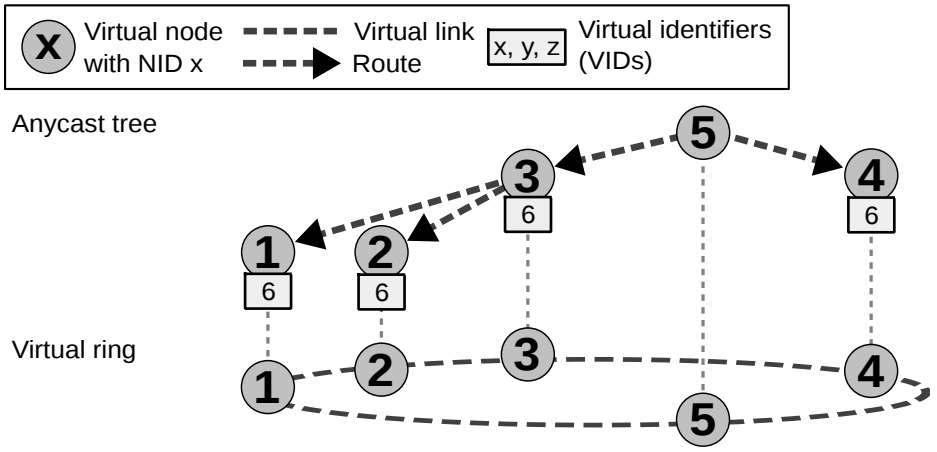


Figure 5.15 – TRout's anycast extension: additional anycast routes.

When using linearization with TRout, each node helps other nodes to find their closest predecessor and successor. Linearization is triggered each round for each new virtual neighbor that has been forwarded to the virtual node once. Figure 5.14 illustrates how linearization builds the virtual path. As mentioned before, in the beginning, each node learns routes to NIDs of its neighbors, i. e., in step 1, virtual node 6 learns about the virtual nodes 1, 4, 8, and 9. Virtual nodes 4, 8 are the best successor and predecessor virtual node 6 and the virtual node has a virtual link to both of them. The routes to virtual nodes 1, 9 are new and need to be linearized. Consequently, routes are forwarded between virtual nodes 1, 4, and, 8, 9, each pair in both directions, so a virtual link is established for each pair.

On node failures, nodes need to linearize the virtual path. The route maintenance keeps track of failed virtual routes. If a route fails the virtual ring maintenance is notified of the failure. In case TRout uses linearization, it marks the linearization neighbors around the failed virtual link as new. These virtual neighbors will be linearized in the next round.

Building the Virtual Ring

For building the virtual ring, virtual nodes which do not have a predecessor seek a the virtual node with the greatest NID using a discovery message each round. Then, the discovered virtual node that does not have an successor establishes a link; the virtual path has transformed into a virtual ring.

5.3.5 Anycast extension

The anycast extension uses the virtual ring routing to add additional routes leading to anycast VIDs. Each node may announce several of those anycast VIDs. The announcements are forwarded along the routing paths of the virtual ring until they reach the node with the closest NID to the announced VID. Furthermore, the announcements add a route to owner, i. e., the node that possesses the same VID, on each node it passes. To distinguish the routes the anycast extension allocates the *owner*-field with the NID of the announcing node. This makes each route unique.

If a subset of nodes announce the same VID the resulting routes intersect at a some node. In this case, the announcement is only forwarded when the distance to the node that possesses the same VID is shorter or has a lower owner-NID to break ties. The result is a anycast tree. When routing to an anycast VID, the virtual ring routing is used until an anycast route is found. Then, the message follows this path to the announcing node.

Figure 5.15 illustrates the routes provided by the anycast extension. It comprises five nodes with their respective virtual nodes. Virtual nodes (1) to (3), and virtual node (4) announce an additional anycast VID 6. These nodes announce routes towards the virtual node that is closest to VID 6, which is (5) in this case[7]. As mentioned before, the announcement of a route stops if a node knows a shorter route to this VID. This results in an anycast tree as shown by the figure.

5.3.6 Optimizations, and Protocol Parameters

TRout implements several optimizations. First, TRout enhances the Discovery mechanism by using a parallel discovery mechanism. Furthermore, it automatically reduces signalling traffic when the virtual ring is stable. For optimization of the concurrency effects of the Linearization mechanism TRout keeps track of the best successors and predecessors linearized so far. Automatic teardowns and a route consistency check ensure that TRout does not maintain unnecessary routes. Finally, TRout can aggregate route update messages into larger messages to reduce the number of messages sent by TRout, if required.

Parallel Discovery Mechanism

TRout a parallel discovery mechanism to speed up convergence. In this case, discovery messages are sent to D_α randomly chosen and pairwise different neighbors.

Discovery/Anycast with Decreasing Overhead

Unlike when using linearization, TRout sends discovery and anycast announcements each round even when the virtual ring is stable and all routes are intact. This induces unnecessary overhead. To reduce this overhead, the anycast and discovery mechanism counts the number of rounds the node did not receive any notification message and no virtual neighbor has been added or removed. If this counter exceeds *delayThreshold* rounds, the next announcements are delayed with each round by a linearly increasing number of rounds, per round. If any change occurs, the counter is reset and the announcements are sent each round.

Linearization with Predecessor/Successor Awareness

The concurrent linearization mechanism has a bad performance [40] as the algorithm was designed for iterative processing of graphs. In a distributed setting, linearization is performed on many nodes concurrently. This leads to a high amount of unnecessary overhead. To lower the unnecessary overhead, TRout uses a simple method: it only linearizes new left or new right neighbors if the linearization neighbor is closer to the NID than the NIDs linearized before. For this purpose, two additional fields are added to the routing-table: the NIDs of the forwarded route to the closest successor and predecessor.

[7] Note that using the euclidean metric, two NIDs may be close to a VID—however, this is highly improbable when the address space is large, i. e., TRout is using 128-bit.

Route Consistency Check and Automatic Teardowns

The route usage record allows TRout to check whether the entries in the routing-table of a node are valid. For this purpose TRout notifies its neighbors of the usage of routes every $R_{consistency}$ rounds. If a node receives a notification of a route that it does not know about, it sends a teardown message to drop the route. This way, invalid routes are removed from the network.

Furthermore, TRout removes unused RT-entries. If a RT-entry does not represent a route used in the virtual ring or is not used by one of the neighbors, it removes this RT-entry after $R_{timeout}$ rounds.

Route Update Queue

To reduce the amount of messages sent by a node, TRout implements a route update queue. Normally, when TRout processes an incoming message and subsequently sends messages, these messages are sent immediately. These messages may comprise only a small number of route updates, leading to a large number of small messages. To aggregate them into larger messages, TRout keeps a message queue for every neighbor. This queue is either flushed, i. e., messages in the queue are sent to the respective neighbor, immediately, *flushQueue* =Immediately, or queued until the next round, *flushQueue* =In-round.

Protocol Parameters

TRout uses several protocol parameters that have a significant influence on its performance. Table 5.2 summarizes these parameters. The T_{round} parameter denotes the period in seconds until the TRout's next round starts. The route maintenance $R_{consistency}$ parameter denotes the number of rounds until TRout checks its routing-table of consistency, so all routes in the routing-table are valid. Furthermore, TRout cleans up orphaned, unused RT-entries that are not needed anymore and the route blacklist each $R_{timeout}$ rounds. The virtual ring module has two modes of constructing the ring: Discovery (default), or Linearization. *mode* denotes the mode used. In Discovery mode, D_α denotes the number of parallel discovery messages sent each round. The greater D_α, the faster TRout converges at the expense of more traffic being generated. Finally, the *flushQueue* parameter denotes whether the route update queue should be flushed immediately after TRout processed route updates from a neighbor, or whether the route updates should be queued until the next round. TRout has been designed to comply with the requirements in Section 5.1.4 for scalability and anycast support. The next section evaluates the properties of TRout using the scenarios presented in Section 5.1.1.

5.4 Evaluation

The TRout protocol has been implemented hand-in-hand with a routing simulation framework, called RoutingSim [97], based on the OMNet++ discrete event simulator [94]. This framework provides build-in statistics, abstract graph readers, and a modular design to ease large-scale benchmarks of routing protocols. The following describes the simulation methodology. Then, the subsequent sections present the evaluation results of TRout's convergence and behavior under network dynamics in Section 5.4.4.

Parameter	Default	Description
T_{round}	1s	Waiting time for the next round
$R_{consistency}$	60	Rounds before the next consistency check
$R_{timeout}$	30	Rounds before the next cleanup of the routing-table
D_{α}	4	No. of parallel discovery attempts
delayThreshold	5	No. of rounds until the discovery/anycast mechanism slows down
mode	Dis.	Mode of virtual ring construction (Discovery (default), or Linearization)
flushQueue	Imm.	Mode of queue flushing (Immediately (default), or In-round)

Table 5.2 – TRout protocol parameters and default values.

5.4.1 Simulation Methodology

The goal of the simulation is to verify TRout's algorithmic scalability on network topologies that match the unstructured relay overlay. Stretch and message delay are not the primary focus of these simulations; however, some results are shown when applicable. Evaluation of scalability usually requires to simulate a large-scale network, i. e., of size up to several thousands. To run the evaluation in a reasonable time, e. g., up to several days, a trade-off between complexity and a realistic simulation must be found. The following describes this trade-off.

Simulation vs. Reality

To reduce the complexity of the simulations the following simplifications have been made:

CMP simulation: Connectivity detection is only feasible if the connectivity domains contain one magnitude more non-relay nodes than relays, cf. Chapter 4. Hence, simulating CMP and TRout in a large-scale network requires even more non-relay nodes. This makes the evaluation unnecessarily complex, since the detection itself has no influence on TRout. Hence, the simulation uses the unstructured relay overlay topologies, cf.. Section 5.1.1, to evaluate TRout.

Unstructured relay overlay: In the unstructured relay overlay topologies, each relay has a unique node identifier NID and several anycast VIDs equal to the CIDs of the detected connectivity domains. For an evaluation of routing stretch, the simulator pre-calculates shortest-paths between all relays. Dynamics in the unstructured relay overlay topology, e. g., failure or recovery of relays, does not change the topology. This relieves the simulation from re-calculating the shortest-paths.

Message loss and delay: The simulation does not consider message loss in the unstructured relay overlay topology since message loss can be compensated using an ARQ mechanism on TRout's deployment. Furthermore, message delay has a small effect on the algorithmic behavior of TRout. It would increase convergence time but does not prevent TRout from converging in the end. Hence,

Parameter	Scenario	Value
Network size N	Both	255, 511, 1023, **2047**, 4095, 8191
Number of CDs N_C	Both	256, 512, 1024, **2048**, 4096, 8192
Max. number of neighbors k	Both	10, **20**, 30
Star diameter d_s	Star	2
Power-law coefficient λ	Int.-insp.	2.3
Max. CDs per relay c	Int.-insp.	6

Table 5.3 – TRout's parameters for the unstructured relay overlay scenario. Values printed in **bold** denote default simulation parameters.

the simulation uses a uniformly distributed message delay between 50-350 milliseconds per link to simulate queueing, and to avoid synchronization effects. The variance of the delay reflects typical delays seen with a DSL-line to national servers ($\approx$ 50 milliseconds) and transatlantic delays ($\approx$ 350 milliseconds).

Neighbor detection and link failure: The simulation abstracts from the detection of new neighbors and neighbor failure. More precisely, the simulation informs TRout about a failing or new neighbor instantaneously. In reality, the detection would take some time and result in a small increase of TRout's convergence time.

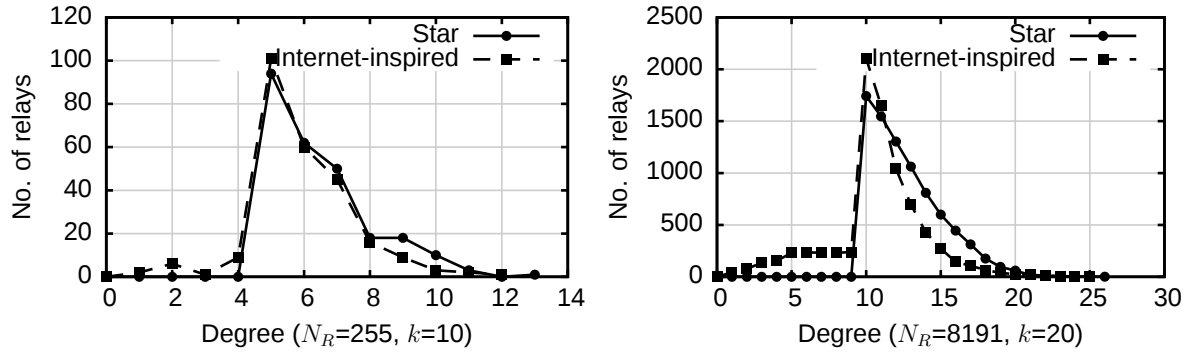
As aforementioned, the simulation uses unstructured relay overlay topologies. The following sections denote these topologies as network topologies.

Network Topologies

The network graphs for TRout's evaluations are generated using the star- and Internet-inspired scenarios presented in Section 5.1.1 and the unstructured relay overlay construction mechanisms presented in Section 5.1.2. Depending on the scenario, these network graphs have either a star- or a Internet-inspired topology. Table 5.3 summarizes the parametrization of the network graph generation. Both star and Internet-inspired topologies connect N_C connectivity domains (CDs) with $N = N_C - 1$ relays. This explains the correlation of the values of N_C and N in the table. For TRout, N is the network size. It denotes the number of relays in the unstructured relay overlay. Furthermore, N_C is the number of different anycast addresses used in the network. The diameter of the star scenario is set to $d_s = 2$, thus, the branches of the star topology comprise a single connectivity domain. This is likely in today's networks. The parameters of the Internet-inspired scenario are the values presented in Section 5.1.1.

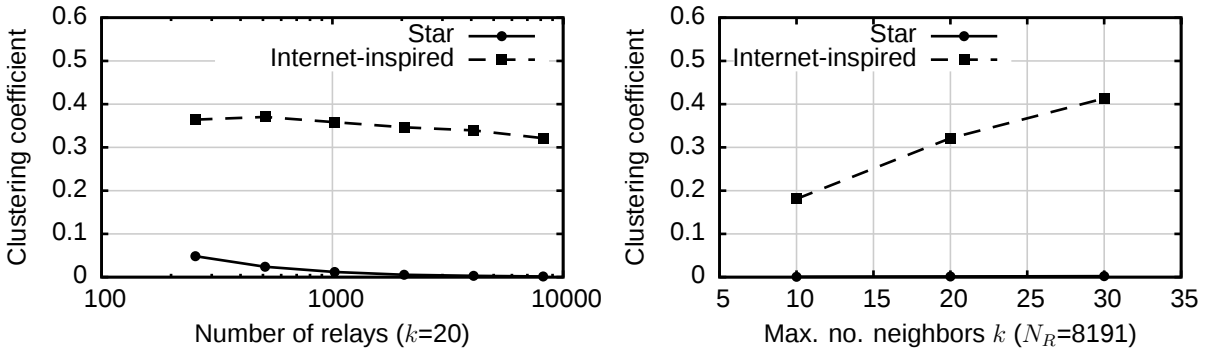
Each relay in the generated network graph has a node identifier NID and several CIDs of the connectivity domains the relays connect to. The simulation uses the NID and CIDs to parametrize TRout on each relay for a realistic simulation. The following paragraphs describe the properties of the resulting network graphs in further detail.

Figure 5.16 shows degree distributions, i. e., the distributions of the number of neighbors, in the generated network topologies of the star and Internet-inspired scenarios. Figure 5.16(a) with $N = 255$ and $k = 10$. Figure 5.16(b) with $N = 8191$



(a) $N = 255, k = 10$ (b) $N = 8191, k = 20$

Figure 5.16 – Degree distributions of the generated star and Internet-inspired unstructured relay overlay.

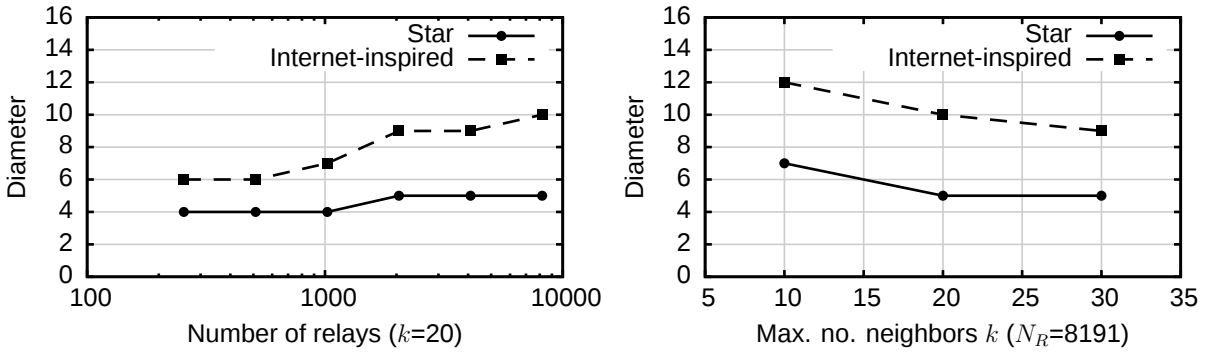


(a) $N = 255, 511, 1023, 2047, 4095, 8191$ (b) $k = 10, 20, 30$

Figure 5.17 – Clustering coefficient of the generated star and Internet-inspired unstructured relay overlay.

and $k = 20$. As expected both show a peak at $\frac{k}{2}$, as each relays tries to connect to at least $\frac{k}{2}$ relays chosen randomly. Also, the degree is bound by k with high probability. In the star topology all relays can connect to each other; hence, the degree is always greater or equal than $\frac{k}{2}$. In the Internet-inspired scenario, however, connectivity domains comprising less than $\frac{k}{2}$ relays may exist. This explains that some relays have a degree lower than $\frac{k}{2}$.

Figure 5.17 shows the distribution of the clustering coefficient as a function of the network size N and the maximum number of neighbors k. Informally, the clustering coefficient describes the connectedness of a graph. A complete graph has a clustering coefficient of 1, while a sparsely connected graph has a clustering coefficient close to 0. For a formal definition refer to Section 2.1. Figure 5.17(a) shows the clustering coefficient of the star and Internet-inspired network topology with an exponentially increasing network size $N = 255, \ldots, 8191$. The figure shows that the clustering coefficient decreases for both, star and Internet-inspired, with the increase of N. This is natural, as the number of neighbors is constant while N increases. In contrast, Figure 5.17(b) shows how the increase of the number of neighbors k increases the clustering coefficient, because the network becomes



(a) Increasing network size (b) Increasing maximum number of neighbors

Figure 5.18 – Diameter of the star and Internet-inspired unstructured relay overlay.

more dense. Both figures show a notable difference between the star and Internet-inspired scenario. The reason for this is that a relay is connected to exactly two connectivity domains in case of the star scenario. In the Internet-inspired scenario, a relay connects to a power-law distributed number of different connectivity domains. This increases the clustering coefficient. In fact, the clustering coefficient is close to the one observed in the Internet [95], i. e., of $\approx 0.2 - 0.3$.

Figure 5.18 shows the diameter of the network graph as function of an increasing network size and an increasing maximum number of neighbors. The diameter is important for TRout's convergence. The larger the diameter is, the more hops TRout's signalling messages need. Figure 5.18(a) suggests that the diameter increases with the network size and follows a logarithmic trend. This logarithmic trend reflects—as expected—the theoretical bounds of the diameter of random graphs. This has already been pointed out in Section 5.1.2. The figure also shows that the star topology has a smaller diameter than the Internet-inspired topology. This is obvious, as the star graph has a smaller diameter than the BA graph. Figure 5.18(b) shows that with an increasing maximum number of neighbors the diameter decreases. This is obvious because each additional edge reduces the diameter.

Network Dynamics

Churn and catastrophic failures are typical models for dynamics in P2P networks. Churn describes continuous change of the network. A catastrophic failure (CF) describes a sudden failure and recovery of relays in the network without prior notice. While a catastrophic failure is a simple model, realistic churn models are hard to obtain. The most prominent churn model is the *LifetimeChurn*-model (LC model). In this model, a new relay joins the network and stays for $t_{lifetime_1}$ seconds. After this time the relay leaves the network and stays out of it for $t_{deadtime_1}$. Then, the relay joins again and stays for $t_{lifetime_2}$ seconds, and so forth. The LC model is periodic. Relays join and leave with different life- and dead-times. Studies [75] show that life- and dead-times vary from several seconds to one hour. Those studies also discuss several possibilities to model churn by stochastic lifetime distributions.

The work by Stutzbach et al. [88], and more recently, by Steiner et al. [84] claim that a Weibull distribution

$$f(x) := \frac{k}{\lambda} \cdot \left(\frac{x}{\lambda}\right)^{(-x/\lambda)^k}, x \geq 0$$

with the Weibull parameter $k = 0.5$ and mean lifetime $meanLifetime = \lambda \approx 10000$ seconds fits observations from the *Kad*-filesharing network. The simulation uses this model as reference and varies the *meanLifetime* between 1000, 5000, 10000, and 20000 seconds. The network dynamics are implemented as relay failure and relay recovery in the network graph. The catastrophic failure uses the following parametrization: after $t_{failure} = 100$ seconds, a fraction $F \in \{0.10, 0.20, 0.40\}$ of all relays fail. Then, after $t_{recover} = 100$ seconds, all failed relays recover.

The following table summarizes the parameters of both models:

Parameter	Model	Value
Fraction of relays that fail F	CF	5%, **10%**, 20%, 40%
Time of failure $t_{failure}$	CF	100s
Duration of failure $t_{recover}$	CF	100s
meanLifetime	LC	1000s, 5000s, **10000s**, 20000s

Table 5.4 – Model parameters of the network dynamics. LC denotes the lifetime churn model, CF the catastrophic failure model. Default values are printed in **bold**.

This completes the simulation's network topology and dynamics. The next section introduces the general simulation setup and environment.

Simulation Setup

The simulation consists of the following steps:

1. The network graph and pre-calculated shortest-paths are loaded. Subsequently, relays are connected by links.
2. The simulator instantiates TRout on each relay. TRout is notified about its physical neighbors, i. e., the neighbors in the unstructured relay overlay topology. Furthermore, the simulation parametrizes TRout with the node identifier NID and the anycast-VIDs on each relay as defined by the network graph.
3. If dynamics are enabled: apply relay failures and recoveries depending on the chosen dynamics model, i. e., either *LifetimeChurn* or *CatastrophicFailure*. On failure or recovery of a relay, the simulation notifies the respective TRout instance.
4. When $T_{sendDelay}$ simulated seconds have been passed, TRout sends a message to a randomly selected relay that has not failed, each T_{round} seconds. If *anycast*-functionality is enabled in TRout, it sends an additional message to an anycast address each T_{round} seconds.

5. After T_{sim} simulated seconds, the simulation ends.

During the simulation, the simulator samples several performance metrics from all relays each second. Thus, these values represent a snapshot of TRout's current network-wide state.

Performance Metrics

Each second, the simulation samples the performance metrics from all relays in the network. Furthermore, the simulation determines the complementary cumulative density function (CCDF), mean, and difference for each metric. The performance metrics are defined as follows:

Convergence time: is the time TRout needs to build an accurate virtual ring. The simulation obtains this metric by checking if the virtual ring is accurate each second. Hence, with a sampling rate of 1/second, the convergence time is an integer and subject to quantization errors.

Traffic: is the accumulated incoming and outgoing signalling traffic per relay as absolute value in bytes.

Bandwidth consumption: is the bandwidth consumption per relay of the incoming and outgoing signalling traffic in bytes per second.

Routing-table size , or, *RT-size:* is the routing-table size as number of entries.

Delivery ratio: is the fraction of successfully and correctly delivered messages.

Stretch: is the stretch of the routes the delivered messages took.

Refer to Section 5.2.1 for additional information on the performance metrics, e. g., a definition of stretch.

Summary of Simulation Parameters

Table 5.5 lists the simulation parameters. The parameters are categorized into three groups. The simulation specific parameters comprise the simulated time limit in seconds, the used network topology using the given network size and maximum number of neighbors. Furthermore, the $T_{sendDelay}$ parameter denotes the delay until the simulation starts sending probe messages using TRout to determine the delivery ratio and stretch. These parameters are followed by TRout's general parameters. These include the duration of a TRout protocol round T_{round} and a flag if the message queue should be flushed immediately or in each protocol round *flushQueue*. The routing mode states wheather the anycast traffic and routing is simulated *additionally* to the unicast routing. Furthermore, $R_{consistency}$ denotes how often TRout checks its routing-table for consistency. $R_{timeout}$ denotes the number of rounds until unused routing-tables are removed. Finally, the TRout virtual ring construction parameters describe how TRout maintains the virtual ring. *mode* denotes the maintenance mechanism, i. e., **Discovery**, or **Linearization**. The **Discovery** mechanism needs further parametrization by D_α and *delayThreshold*. D_α denotes the number of discovery attempts done in parallel. *delayThreshold* denotes the number of rounds the successor and predecessor need to be stable until the discovery attempts slow down round by round to reduce overhead. If not stated otherwise, the simulations use default values. The next sections present the evaluation results.

Parameter	Simulation values
– Simulation specific	
T_{sim}	Simulated time limit in seconds
Topology	Star (default), Internet-inspired
Network size N/N_C	255/256, ..., **2047/2048**, ..., 8191/8192
Max. number of neighbors k	10, **20**, 30
$T_{sendDelay}$	1 second
– TRout general	
T_{round}	1 second
flushQueue	Immediately (default), In-round
Routing mode	Unicast (default), Anycast
$R_{consistency}$	60 rounds
$R_{timeout}$	60 rounds
– TRout virtual ring construction	
mode	Discovery (default), or Linearization
D_α	$1, \ldots, \mathbf{4}, \ldots, 8$
delayThreshold	5 rounds

Table 5.5 – Overview of TRout's simulation parameters. Default values are printed in **bold**.

5.4.2 Convergence

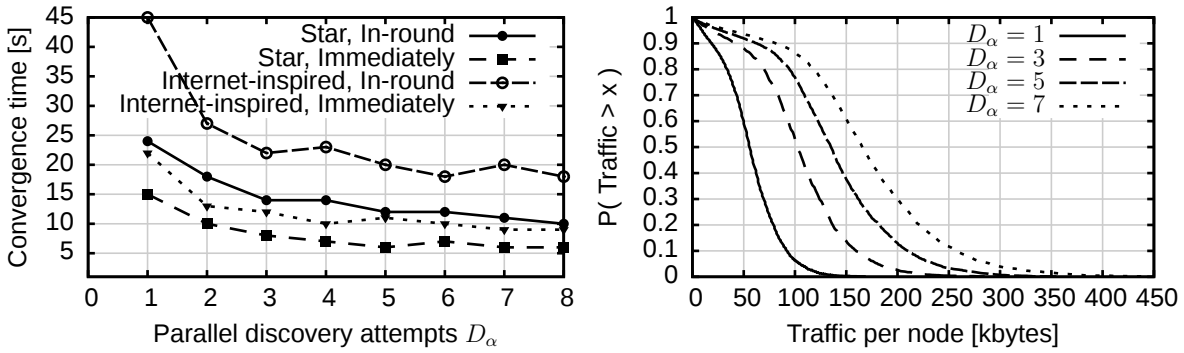
By definition, TRout converges when the virtual ring has been built and no routing-table changes are pending. To evaluate convergence, the simulator performs a "cold-start", i. e., TRout is started on all relays virtually simultaneously. The simulator adds a variance of $1000ms$ to avoid synchronous behaviour. The simulator monitors the state of the virtual ring and ends the simulation after $T_{sim} = 60$ seconds of simulated time. The following paragraphs describe simulation results concerning the impact of

- D_α and message queue
- network size, and
- maximum number of neighbors

on convergence time and traffic using the Star topology. Furthermore, light is shed on TRout's stretch.

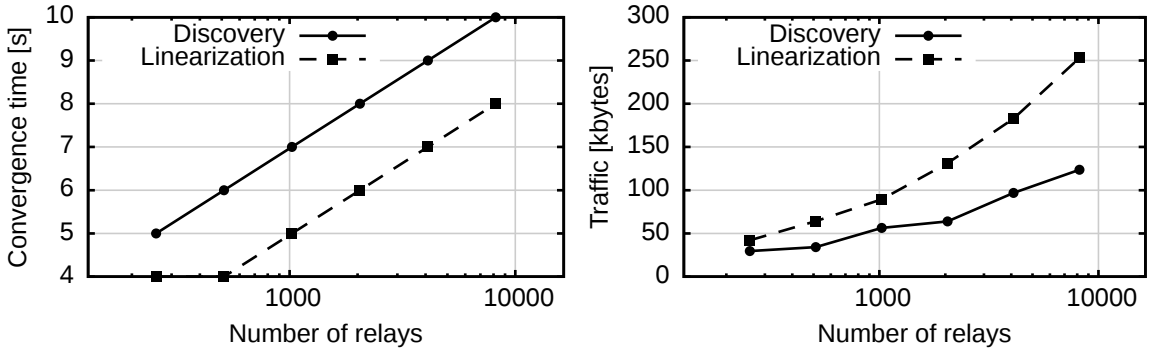
Impact of D_α and Message Queue

When TRout uses the Discovery mechanism to maintain its virtual ring, it sends discovery messages in parallel to several relays. TRout does this to reduce convergence time. Figure 5.19(a) shows the convergence time as function of the number of parallel discovery attempts D_α. The results come from experiments using Star and Internet-inspired topologies and both *flushQueue* strategies. In all cases, convergence time decreases when D_α increases. However, with a D_α greater than 4, the convergence time shows only weak improvement. Thus, the default value of



(a) Convergence time (b) Traffic (Star topology)

Figure 5.19 – Impact of parallel discovery attempts (using $k = 20$, Unicast, Discovery).



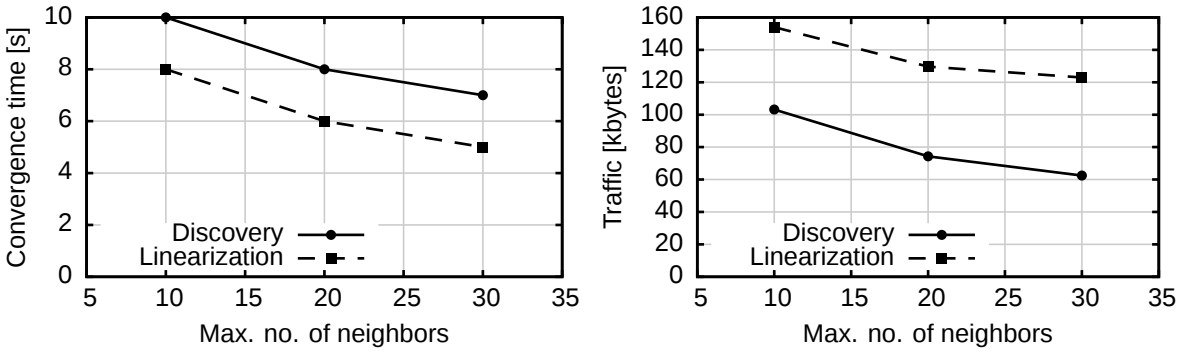
(a) Convergence time (b) Traffic

Figure 5.20 – Convergence with different network sizes: Linearization vs. Discovery (using Star topology, $k = 20$, Unicast).

$D_\alpha = 4$ is adequate. The figure also shows that the message queue flush strategy In-round increases convergence time. This is natural, because each signalling message is delayed by $\frac{T_{round}}{2}$ seconds on average. The fluctuations in convergence time are caused by quantization errors in the sampling of the ring accuracy each second. Parallel discovery attempts are associated with costs in terms of generated traffic. Figure 5.19(b) shows the CCDF of the traffic with linearly increasing number of parallel discovery attempts D_α. As expected, the traffic increases proportional to D_α.

Impact of the Network Size

Convergence time with a growing network size N is an important metric of scalability. Figure 5.20 shows simulations of TRout with an exponentially increasing network size N from 255 to 8191 using the Discovery and Linearization virtual ring maintenance mechanisms. For the Discovery mechanism, the number of parallel discovery attempts D_α is 4. Figure 5.20(a) shows the convergence time as function of the network size. The convergence time increases logarithmically with the network size with both mechanisms, i. e., Discovery or Linearization. This is sur-



(a) Convergence time (b) Traffic

Figure 5.21 – Convergence with a different maximum number of neighbors k using Linearization and Discovery (using Star topology, $N = 2048$, Unicast).

prising, as the Linearization mechanism is often considered as the better alternative to the Discovery mechanism [55, 58]. However, these results show that parallel discovery attempts can nearly achieve the convergence time of Linearization.

Figure 5.20(b) compares the traffic generated until TRout converges. The results come from experiments using the Discovery and Linearization mechanisms. The figure shows that, with the Discovery mechanism, TRout converges with sub-linear (almost logarithmic) signalling overhead subject to the network size. The Linearization mechanism, however, lets TRout need more traffic which is caused by its concurrent and partly overprovisioned stabilization. The plot suggests a polynomial development.

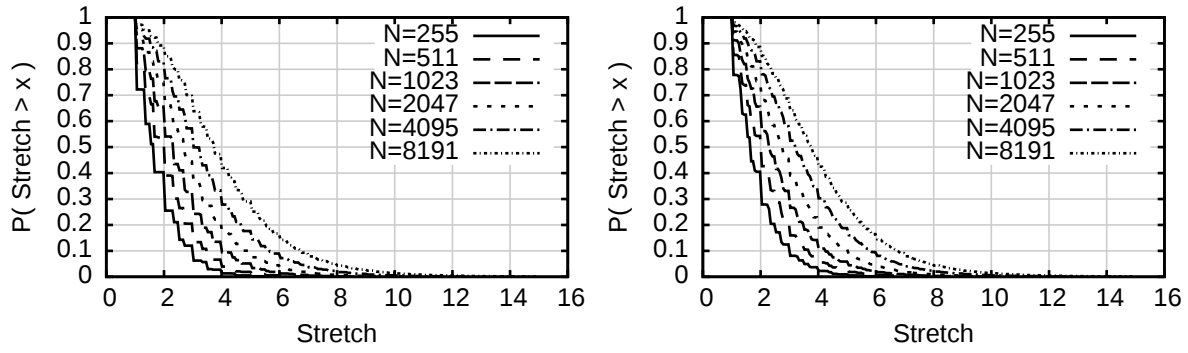
When comparing the costs in terms of traffic needed for TRout to converge, the Discovery mechanism has a convergence time almost as good as the Linearization mechanism but with less traffic.

Impact of the Maximum Number of Neighbors

As mentioned in Section 5.4.1 the maximum number of neighbors k has an impact on the diameter of the network topology. Thus, an increase of k is expected to reduce the convergence time of TRout since path lengths decrease. Figure 5.21 shows the convergence time and traffic as function of the maximum number of neighbors k. The figures confirm that convergence time and traffic reduce with an increase of k—independent from the virtual ring maintenance mechanism.

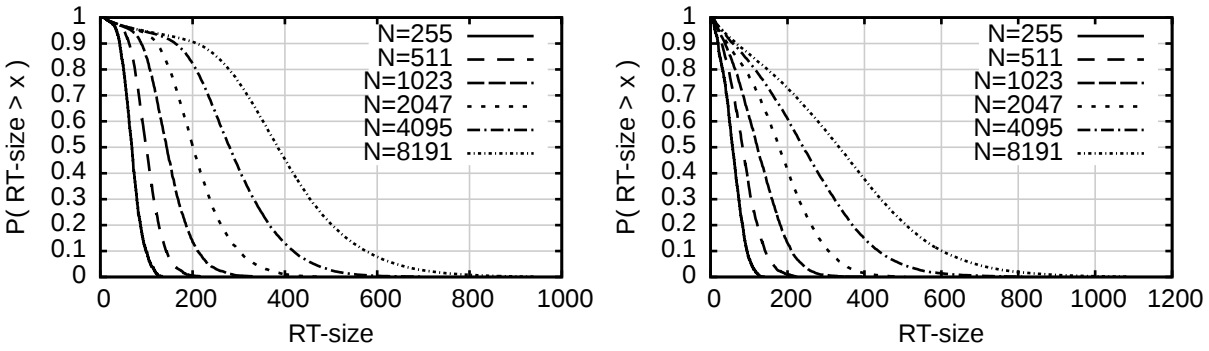
Stretch

The simulation sends probe messages each second to a random relay in the network and records the stretch. Figure 5.22 shows the CCDF of the stretch for an exponentially increasing network size. The results in Figure 5.22(a) come from experiments within the Star topology. The results in Figure 5.22(b) come from experiments with in the Internet-inspired topology. When comparing the figures, the topology has virtually no influence on the stretch, even though the topologies highly differ in the clustering coefficient, cf. Section 5.4.1. Furthermore, the stretch



(a) Star topology (b) Internet-inspired topology

Figure 5.22 – Impact of the network topology on (Multiplicative-)Stretch (Star topology, $k = 20$, Unicast, Discovery).



(a) Star topology (b) Internet-inspired topology

Figure 5.23 – RT-size distribution with different network sizes using Star and Internet-inspired topologies (and $k = 20$, Discovery, Unicast)

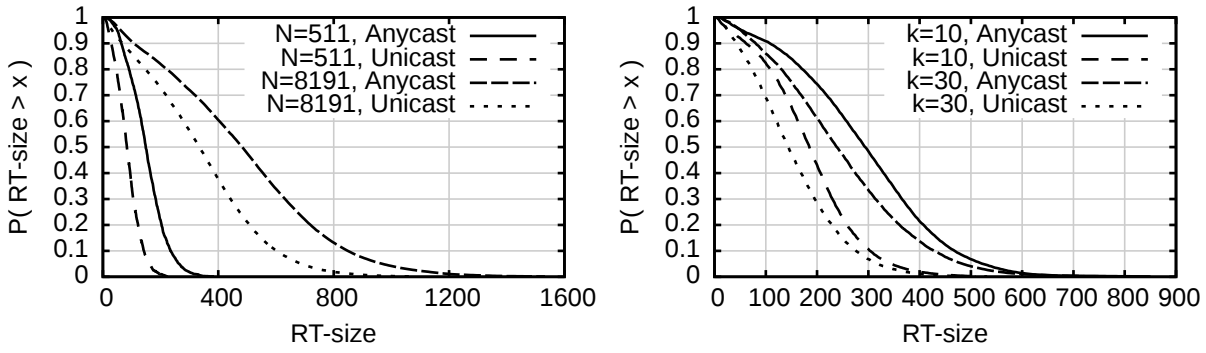
increases logarithmically subject to the network size. This confirms the findings in [63].

5.4.3 Routing-Table Size

As mentioned before, scalability of routing protocols is defined by the number of routing-table entries (RT-size) required to route messages within a network of size N. If the increase in the number of RT-entries is sub-linear subject to the network size, a routing protocol is considered *scalable*. The following discusses the RT-size with and without anycast routing-table entries subject to the network size.

Impact of the Network Size

Figure 5.23 shows the CCDF of the RT-size of all relays with an exponentially growing network size for both Star and Internet-inspired topology. In both cases the RT-size grows poly-logarithmically subject to the network size—indicated by the horizontal distance of the CCDF curves. This means, TRout is scalable on both topologies. However, when using the Internet-inspired topology, a bigger amount of relays have a large routing table—indicated by the steeper CCDF curves. It is likely, that this is caused by the higher clustering coefficient of the Internet-inspired



(a) Varying network size

(b) Varying maximum number of neighbors

Figure 5.24 – RT-size distribution using Unicast and Anycast, and Internet-inspired topology

topology. In summary, the results suggest that TRout is scalable in the considered scenarios.

Unicast vs. Anycast

TRout's anycast extension adds additional routing-table entries. Figure 5.24 shows a comparison of TRout's RT-size CCDF with and without the Anycast routing-table entries for $N = 511$, $N = 8191$ with $k = 20$ and $k \in \{10, 20, 30\}$, $N = 2047$. Figure 5.24(a) shows that with a network size increased by a factor of 16, $N = 511$ to $N = 8192$, the number of additional routing-table entries grows only by 50%. This suggests that the number of routing-table entries added by the anycast extension is logarithmic subject to the network size. The same applies when the maximum number of neighbors grows (see Figure 5.24(b)). In summary, the results suggest that the anycast extension does not violate TRout's scalability.

5.4.4 Churn and Catastrophic Failure

To show how TRout handles network dynamics, the delivery ratio and the bandwidth consumption and traffic required on each relay to repair the routing-tables is of importance. The following describes the evaluation results using *LifetimeChurn* with the mean lifetimes of Lt=1000, 5000, 10000, and 20000 seconds.

Bandwidth Consumption with Lifetime Churn

Figure 5.25 shows the bandwidth consumption per relay as function of the network size for Linearization, Discovery, and different mean lifetimes under the *LifetimeChurn* model. On the one hand, Figure 5.25(a) shows Discovery's unperturbed behaviour under churn. Even with a low mean lifetime, and thus high churn, the bandwidth consumption per relay grows only sub-linearly with the network size. This is natural, because discovery messages are sent once each T_{round} seconds at maximum, independently of the changes in the network. Hence, TRout is adaptation-scalable when using Discovery. Figure 5.25(a) shows the bad behaviour of Linearization under churn. Since re-linearization is triggered on any change in the network this leads to a bandwidth consumption that grows almost linearly with the network size. Even small changes in the network cause a large

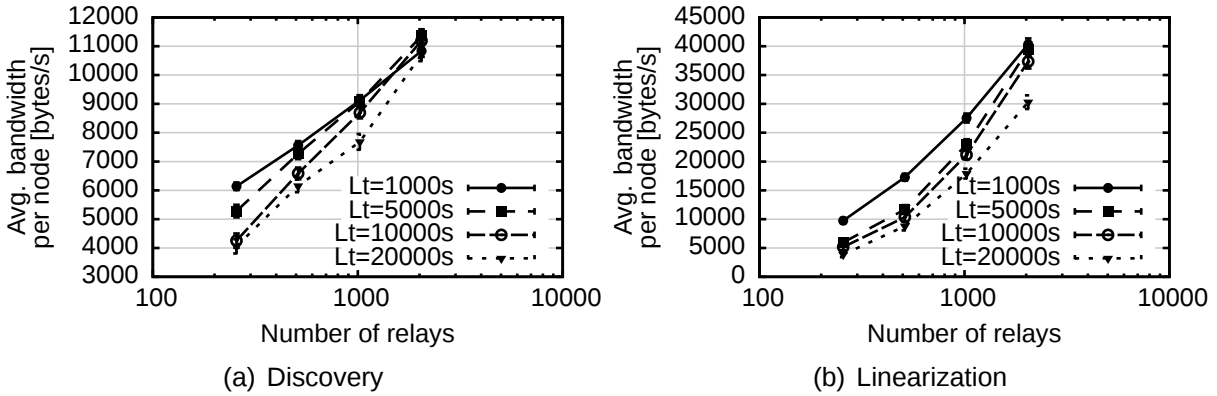


Figure 5.25 – Bandwidth consumption with lifetime churn in networks of increasing size and Internet-inspired topology (using $k = 20$, Unicast).

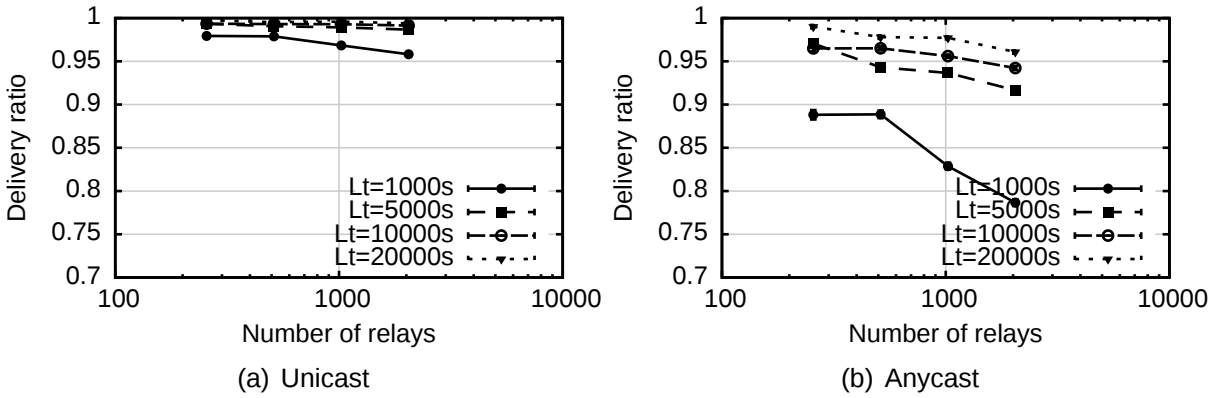


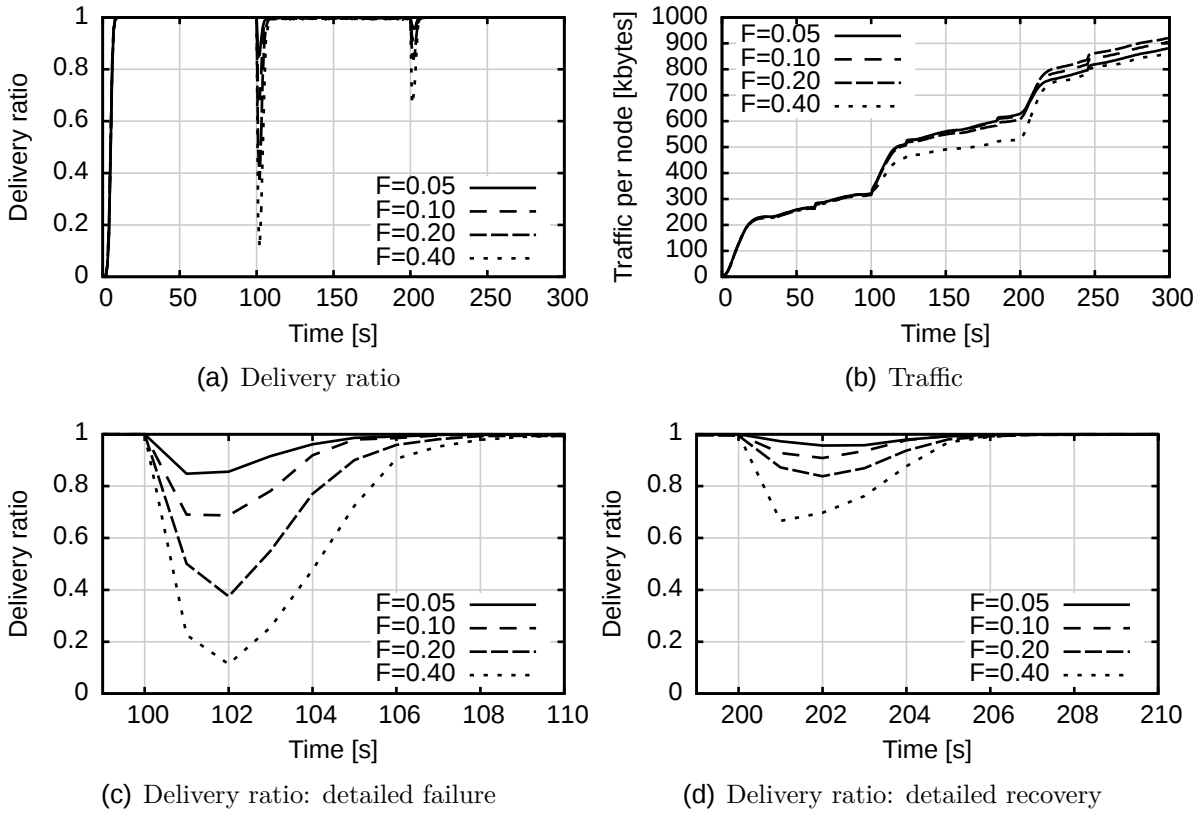
Figure 5.26 – Delivery ratio with lifetime churn in networks of increasing size (Star topology, $k = 20$).

number of concurrent Linearization steps. This makes Linearization a bad candidate for virtual ring maintenance in cases of churn.

In summary, the results suggest that the Discovery mechanism handles lifetime churn better than the Linearization mechanism. Worse, the results suggest that TRout is only adaptation-scalable when using the Discovery mechanism. The further discussion only covers the Discovery mechanism because of this insight.

Delivery Ratio of Unicast and Anycast with Lifetime Churn

Figure 5.26 shows the delivery ratio as function of the network size for Unicast and Anycast messages under different mean lifetimes. Figure 5.26(a) shows that the Unicast delivery ratio stays above 95% even in cases of high churn and up to a network size of 2047. The figure also shows that the amount of churn TRout can handle depends on the time TRout needs to repair the routing-tables. The longer this takes the less messages are delivered. The curves that represent experiments with a mean lifetime of 1000 seconds confirm this suggestion; the delivery ratio is logarithmically reduced with the network size growth. This correlates with



(a) Delivery ratio (b) Traffic

(c) Delivery ratio: detailed failure (d) Delivery ratio: detailed recovery

Figure 5.27 – Delivery ratio and traffic during a catastrophic failure of a fraction of $F \in \{0.05, 0.10, 0.20, 0.40\}$ relays using the Internet-inspired topology (and $k = 20$, $N = 2047$, Anycast).

the diameter of the network graph. The greater the diameter is, the more hops signalling messages need, and the slower the routing-table stabilization.

Furthermore, Figure 5.26(b) shows the delivery ratio of messages routed using the Anycast routing-table entries. Anycast strongly relies on an intact virtual ring, e. g., Unicast messages need to be delivered reliably. This explains why the Anycast delivery ratio is worse than the Unicast delivery ratio. Moreover, the figure shows that cases of high churn, i. e., a mean lifetime of 1000 seconds, lead to a high loss of messages.

In summary, the results suggest that Unicast and Anycast delivery ratios are over 92% with an mean lifetime of 10000 seconds which is expected in P2P applications and a network size less than 2048. The results suggest a logarithmic drop of the delivery ratio subject to the network size. It is worth noting that the fraction of undelivered messages include those that were already on its way to destination before the relay failed.

Delivery Ratio and Traffic during a Catastrophic Failure

Another case of dynamics is the catastrophic failure. In this case, TRout has to repair its routing-tables to compensate for failed relays and re-integrate recovered relays.

Figure 5.27 shows the delivery ratio and traffic as function of simulation time during these events using the Discovery mechanism, the Internet-inspired topology of size $N = 2047$ and a maximum number of neighbors $k = 20$. Figure 5.27(a) shows that TRout recovers quickly, i. e., in less than 8 seconds, from a failure of 40% of all relays. Furthermore, the recovered relays are integrated at the same speed. The figure shows a drop of the delivery ratio in case a fraction of the relays fail or recover. In case of a relay failure TRout must repair all routes traversing the failed relay. In case of a relay recovery TRout needs to re-integate the recovered relay into the virtual ring. This explains the sharp drop of the delivery ratio on relay failures compared to the relay recovery. Figure 5.27(c) shows the detailed delivery ratio of the failure. The curves show a sharp drop of the delivery ratio almost proportionally to the fraction of relays that fail. Figure 5.27(d) shows the detailed recovery of the relays. Since all relays send a message each round even when the relay has not yet been re-integrated in the virtual ring, the delivery ratio drops in proportion to the fraction of relays that recovered.

Figure 5.27(b) shows the traffic consumption per relay during the catastrophic failure. The Discovery mechanism slows down when the virtual ring is stable; consequently the traffic grows less after 30 seconds. When relays fail, the Discovery mechanism returns to normal speed to repair the damage. The same occurs when relays recover to quickly re-integrate the relays.

In summary, the results suggest that TRout can handle a catastrophic failure and recovery of up to 40% of all relays and re-stabilizes in less than 10 seconds in an experiment with $N = 2047$ relays.

5.5 Summary

The CDIP protocol comprising TRout interconnects heterogeneous networks. For this reason, this chapter provides an in-depth analysis of connectivity domain scenarios which are likely in today's and future networks. For interconnecting the connectivity domains, several routing protocols were discussed and, subsequently, the Tailored Routing (TRout) has been presented to provide anycast and identifier-based routing for CDIP.

TRout is inspired by the Virtual Ring Routing (VRR) protocol. In contrast to VRR, TRout is modular, implements improved stabilization mechanisms to build the virtual ring, and supports anycast. Furthermore, TRout does also not suffer from loopy-cycles that required VRR to flood the network. TRout has been evaluated in Section 5.4 using simulations. One of the most surprising result and insight is the behavior of the linearization mechanism for building the virtual ring. Related work stated this mechanism as new and better way of maintaining the virtual ring with proven behavior. While this seems to be true for bootstrapping a virtual ring, in the case of lifetime churn, linearization performs less than the discovery mechanisms. More precisely, the parallel discovery mechanism allows the same performance as linearization with less costs in terms of traffic and bandwidth consumption.

The simulations suggest that TRout can recover from a catastrophic failure of up to 40% of all relays. They also suggest that the bandwidth needed to handle

lifetime churn grows logarithmically to the network size when using parallel discoveries. The experiments observing the routing-table size suggest that it has a poly-logarithmic growth subject to the network size. This confirms the theoretical findings in [63] and the good scalability of TRout. In summary, the evaluations suggest that TRout does comply with all requirements stated in Section 5.1.4 for CDIP's scenarios.

Chapter 6

Conclusion and Further Research

The popularity of P2P applications in the Internet is growing. Telephony using Skype, Filesharing via BitTorrent, and virtual private networks state some examples. Furthermore, research projects, like the Spontaneous Virtual Network project (SpoVNet), use P2P technologies for provisioning new services. However, the growing amount of heterogeneous networks limit usability of P2P applications due to the limited connectivity these networks provide.

6.1 Results

This thesis presents a middleware that enables the deployment of P2P applications on heterogeneous networks. For this purpose, it introduces the CD middleware that provides seamless connectivity for P2P applications on heterogeneous networks. The CD middleware uses an approach that consists of two main achievements:

- autonomous detection of connectivity, and
- interconnection of heterogeneous networks.

Autonomous Detection of Connectivity

For efficient provisioning of seamless connectivity, it is important, to detect groups of nodes with existing seamless connectivity, so-called Connectivity Domains (CDs), first. Nodes in more than one connectivity domains are denoted relays. Due to the properties of P2P applications, no explicit knowledge about heterogeneous networks and the existing connectivity is available. This makes detection of connectivity challenging. Chapter 4 discussed this problem in further detail. It provides an in-depth theoretical analysis of the feasibility to detect the existing connectivity in heterogeneous networks. This analysis results in a solution, that includes building an unstructured overlay, detecting all relays in this overlay, and non-relay nodes agreeing on a connectivity identifier (CID). This solution does not weaken

any of the strengths of P2P applications. The main findings of the analysis of connectivity detection using the solution are as follows:

- Detecting relays is an essential element of connectivity detection as undetected relays may lead to a failure of the CID agreement.
- Connectivity detection for P2P applications is feasible if the number of nodes in several connectivity domains together is much smaller than the number of nodes in a single one. This is the case in most of today's networks.
- The worst-case for relay detection is a node with access to other nodes in exactly two connectivity domains.

Based on the theoretical insight, the Connectivity Measurement Protocol (CMP) implements a protocol for connectivity detection. CMP's evaluations confirm the theoretical findings and show additionally, that CMP is reactive to connectivity changes. In the considered worst-case scenarios, CMP detects connectivity in less than 20 seconds and consumes only about 4 kbytes/s on average per node during this time. The peak bandwidth per node stays below 9 kbytes/s. In the best-case, CMP converges at the same time the unstructured overlay is set up, i. e., when all overlay neighbors k' checked a new node using triangle checks. This induces a overhead limited within $O(k')$.

The concept of CMP and the gained insights may have a positive impact on other fields of applications as well, e. g., for evaluating other approaches that deal with non-transitive connectivity, e. g., [38, 34]. Additionally, random number agreements are applicable in many distributed systems, e. g., for leader election or consensus. Therefore, it is likely, that other applications will benefit from the designs described in Section 4.4.

Interconnection of Heterogeneous Networks

The detected connectivity enables the efficient interconnection of heterogeneous networks. To achieve this, Chapter 5 introduces the Connectivity Domain Interconnection Protocol (CDIP). CDIP builds an unstructured relay overlay containing all relays and spanning the detected connectivity domains. Furthermore, CDIP uses a routing protocol called Tailored Routing (TRout) on the relays to route messages across heterogeneous networks by using the CIDs. Section 5.1 discusses appropriate scenarios and requirements while Section 5.2 reviews related work to find an appropriate starting-point for TRout's design. Finally, Sections 5.3 and 5.4 describe TRout's design and evaluate it thoroughly. TRout has the following features:

Identifier-based routing: TRout uses identifiers directly for routing. No additional mapping mechanism is necessary.

Anycast support: TRout supports anycast which enables several nodes to announce the same identifier. TRout will route messages sent to the closest node that announced the identifier[1].

[1] With respect to the virtual ring.

Scalable: The evaluation of TRout in Sections 5.4.3 and 5.4.4 show, that it is scalable in static and dynamic networks, i. e., the overhead required for stabilization grows in proportion to the network changes, especially with lifetime churn.

Modular: TRout's design consists of 3 modules: route maintenance, virtual ring maintenance, and anycast support. The route maintenance eases the handling of route failures and setup of new routes. This allows to use TRout as base for other DHT-inspired protocols.

Improved stabilization: TRout uses a parallel discovery mechanism to maintain its virtual ring. The evaluation in Section 5.4.4 shows that the parallel discovery mechanism has better performance than the linearization mechanism proposed recently.

Those features make TRout a suitable protocol for interconnecting heterogeneous networks for P2P applications. Additionally, with some modifications, TRout can be used in ad hoc networks like VRR. Some features, e. g., good scalability in the case of dynamic networks, make TRout also a good candidate for further investigation in scalable routing.

The CD middleware uses CMP and CDIP to provide seamless connectivity for P2P applications. In addition to the seamless connectivity, P2P applications can use the CIDs to support heterogeneous networks efficiently, e. g., for making better routing decisions, or for more robustness when relays fail. This opens up a new field of possibilities for tailoring P2P applications to heterogeneous networks. It is expected that heterogeneity will grow in the future, e. g., by network virtualization, and new specialized network types. This leads to highly heterogeneous networks. This thesis contributes important components that help handling this growing number of heterogeneous networks.

6.2 Review of the Claims

This section reviews the claims stated in Chapter 1 and verifies to which extent each of these claims hold:

Claim 1: *The CD middleware does not weaken the typical strengths of P2P applications.*

CMP and CDIP are designed carefully not to weaken any of the typical strengths of P2P applications. CMP is scalable and induces a low overhead, i. e., a peak bandwidth below 9 kbytes/sec, during its convergence time. It also handles joining, leaving or failing nodes smoothly, in less than 20 seconds in the considered worst-case scenarios, cf. Section 4.6.5. CDIP uses TRout for routing messages across connectivity domains. Like CMP, TRout is scalable (see Section 5.4.3), even in cases of churn, cf. Section 5.4.4. The further simulations in Section 5.4 suggest that the overhead induced by TRout grows sub-linearly with the number of relays, connectivity domains, and churn.

Claim 2: *The CD middleware can detect the connectivity provided by heterogeneous networks.*

As claimed the Connectivity Measurement Protocol can detect the existing connectivity in heterogeneous networks in typical scenarios. However, there are

some limitations as described theoretically in Sections 4.3 and 4.4, and simulative in Section 4.6.3. Briefly, connectivity detection is difficult in the unlikely scenario of having more relay than non-relay nodes.

Claim 3: *The CD middleware can provide seamless connectivity for P2P applications and requires only small modifications on existing applications.*
Using the CMP's CID agreement and detected relays CDIP provides seamless connectivity using the TRout routing, that provides anycast support. To route messages across heterogeneous networks, P2P applications just use a different address format, i. e., CDIP-addressing; no further changes are necessary.

6.3 Further Research

The concepts presented in this thesis just scratched the surface of possible use-cases of CMP and CDIP. The following sections discuss further research on each contribution:

Detection of Connectivity

CMP detects connectivity using an unstructured overlay. Some of the theoretical considerations assume this unstructured overlay to form a random overlay graph. Although, the presented discovery mechanisms in Section 4.5.5 return random samples with high probability, the exploration of CMP using additional approaches from related work is of interest. Related work usually combines random sampling with building an unstructured overlay at the same time. Therefore, the method of building unstructured overlay could be revised at the same time which could make CMP's design lighter.

CMP also suffers from many security issues stated in Section 4.7.3. While some of those issues are inevitable, some have a good chance of getting solved, e. g., preventing the mosquito attack [52] on the unstructured overlay.

Finally, even though CMP's default parameters are chosen carefully, they might need additional adjustment to serve a specific P2P application to its full extent. CMP, furthermore, introduces generalizations and extensions, i. e.,

- the integration of networks with scarce resources,
- naming connectivity domains, and
- CMP's support of new network paradigms,

as discussed in Section 4.7.1 whose exploration is left for further research.

Interconnection of Heterogeneous Networks

The Connectivity Domain Interconnection Protocol (CDIP) builds an unstructured relay overlay and uses TRout to route messages across connectivity domains. Although this is enough to provide seamless connectivity for P2P applications, it does not use the information provided by CMP to its full extent on both levels: the unstructured relay overlay and routing.

The unstructured relay overlay currently does not consider the heterogeneity of the connectivity domains, e. g., when a relay knows that a certain connectivity

domain has lower bandwidth capacities or less resources in general, it could connect to less relays inside this connectivity domain. Putting more structure to the unstructured relay overlay will certainly help to use the heterogeneous resources more efficiently and balance load.

The Tailored Routing (TRout) used by CDIP in the unstructured relay overlay has a potential to increase efficiency. First, TRout can be extended to support metrics on the routing paths, e. g., for balancing load in the unstructured relay overlay. Second, it could reduce stretch by adding additional overlay neighbors to relays in the same connectivity domain as stated in Section 5.1.2. Of course the amount of additional overlay neighbors need to be limited to ensure scalability.

Third, TRout could adapt its node identifiers (NIDs) to the unstructured relay overlay topology. This would reduce stretch of routes, delay of messages, and the routing-table size. Numerous concepts how this can be done are discussed in related work [42] for key-based routing schemes. As TRout is based on the key-based routing, those concepts can also be applied to TRout. In addition, the network coordinates in [42], enables non-relay nodes to choose closer relays in terms of message delay. Finally, TRout could also be replaced by compact routing [80]; this, however, would need a careful design of the compact routing scheme as most compact routing schemes are not adaptation-scalable.

Appendix A

Communication and Discovery

P2P applications commonly are deployed on homogeneous networks, mostly on the Internet. To extend P2P applications to work with heterogeneous networks, e. g., networks using different protocols, having different address-spaces, P2P applications need a unified way of communicate and discover other networked devices, or *nodes*, running the P2P application. The communication and discovery layer in the CMP/CDIP middleware provides an interface which implements this functionality.

The communication and discovery layer comprises a simple, asynchronous message-based communication interface and discovery interface to discover nodes already running the P2P application. Furthermore, different address types, e. g., IPv4, IPv6, TCP/UDP-Ports, or MAC addresses are encapsulated in a generic binary container called *address-set*.

Figure A.1 shows the placement of the communication and discovery layer. It provides two interfaces: one for exchanging messages between P2P nodes in heterogeneous networks and another one for discovering other nodes at random. Conceptually, the communication interface (de-)multiplexes messages send or received to modules that provide the functionality of the interface. The figure shows three exemplary implementations for TCP, UDP, and Bluetooth L2CAP. The discovery interface uses several mechanisms to discover other nodes in the local network, e. g., multicast. Furthermore it uses peer sampling to distribute locally discovered nodes to other nodes.

A.1 Data Types

This section provides introduces basic data types used by the communication and discovery layer: a address-set for generic encapsulation of arbitrary addresses, and a link context comprising address-sets associated with a communication link.

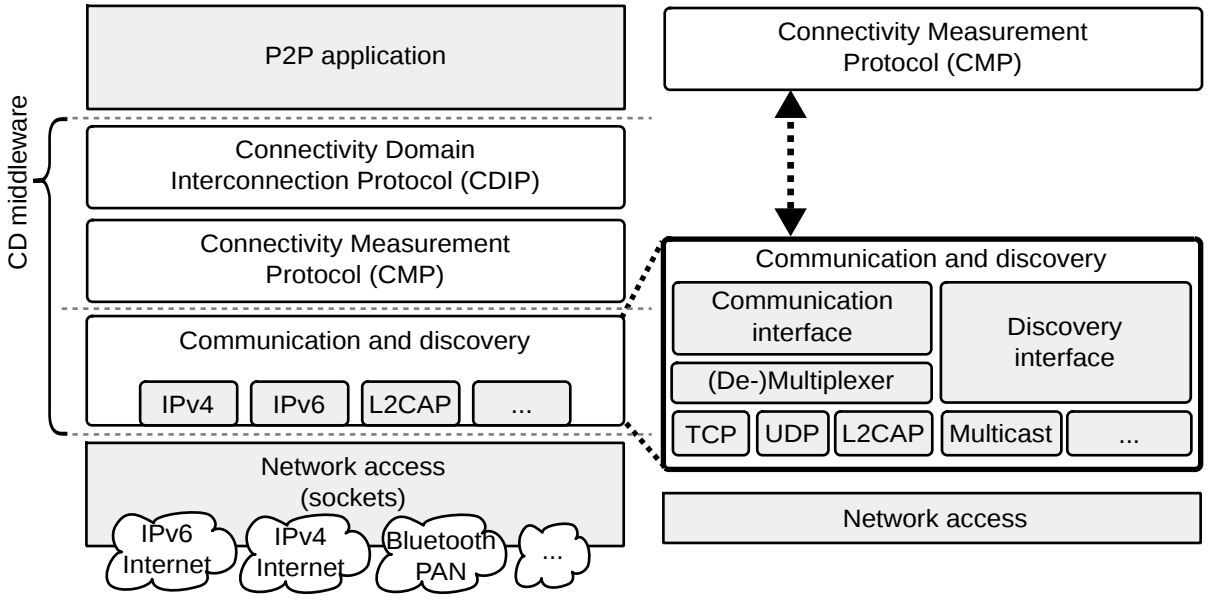


Figure A.1 – The communication and discovery layer in the P2P middleware

Address Set

A address-set, is a container for addresses of any kind. An address in an address-set should be *atomic*, i. e., the address can not be separated into sub-addresses, e. g., a TCP server has an address-set comprising the IP address and port as separate addresses.

All addresses in an address-set are stored as pure binary data using a type-length-value (TLV) encoding scheme:

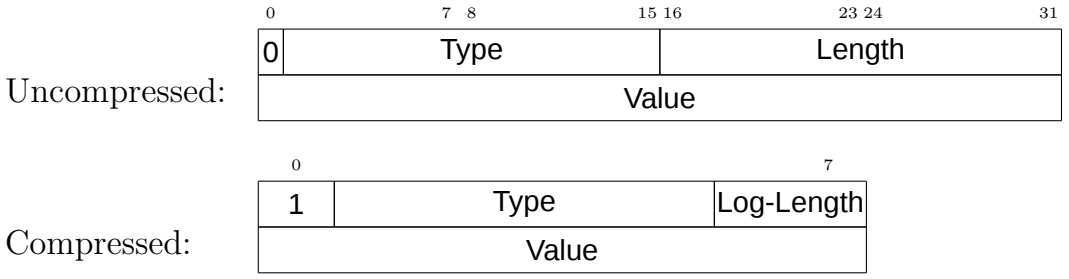
Type	Length	Value(s)

The type and length fields have 16 bits. Both fields are unsigned 16-bit integers encoded in network-order (big endian). The length fields indicates the length of the value, and the type uniquely identifies the address type. A address-set is encoded using a sequence of pairwise different TLVs (set property). Thus a address-set comprising an IPv4 address and TCP port, e. g., `10.11.12.13:14`, is encoded as follows:

`0x0800`	`0x0004`	`0x0a 0x0b 0x0c 0x0d`
`0x0006`	`0x0002`	`0x00 0x0e`

The order of the TLVs is not of importance. The total size of this address-set is 14 bytes. Compared to the length of the values, i. e., 6 bytes a lot of overhead is introduced by the type and length fields.

To reduce this overhead, the first bit of the type field indicates that the type-length header is compressed. Both formats are encoded as follows:



In the compressed form, 32 address types can be defined in the Type-field. The Log-Length holds the logarithmic length of the value to a base of 2. Thus, a length of 1,2,4, and 8 bytes is possible. An IPv4 address plus TCP port has a compressed size of 8 bytes. In both cases the address type fields need to uniquely identify the address type used in the P2P applications. The precise numbers are left as implementation specifics.

Due to the type-length-value format the P2P application can work with these address-sets without knowing the real addresses encoded in them. Thus, can be treated like a black-box. However, the P2P application can transfer them to other nodes and perform operations like union and intersection. Thus, if X, Y are address-sets, $X \bigcap Y$ denotes the intersection, $X \bigcup Y$ denotes the union of X and Y.

Link Context

When a node receives messages, a node creates a link-context that describe the source and the used destination address-set to send the first message. These address-sets are called local and remote initial address-sets (IAS), and denoted by $addr(x, y)$, where x is the local, and y the remote node. An example, for an IAS, are IP addresses and TCP ports of the local and remote server socket. Nodes exchange the IAS to allow other nodes to send messages to other nodes.

Furthermore, it comprises a address-set used for communicating with the remote node. In case of TCP, the address-set would comprise the local/remote TCP-ports and IP addresses after the connection has been established. Those address-sets are denoted by $caddr(x, y)$, where x is the local, and y the remote node. The interfaces introduced in the following sections use these data types for message exchange and node discovery.

A.2 Communication Interface

The communication interface provides rudimentary functions to send and receive messages unreliably. The difference to, e. g., Berkeley sockets is that the functions use address-sets to send data. Furthermore, when receiving messages the interface provides an link-context to return messages and managing the address. The following specifies the interface in detail:

send(*message*, *address-set* [, *maxDelay*, *strategy*]): This function sends a message to another node using the given *message* and *address-set*. The remaining parameters are optional.

The *maxDelay* parameter denotes that the message should leave the device's output buffer after *maxDelay* milliseconds. The main use of the parameter is to

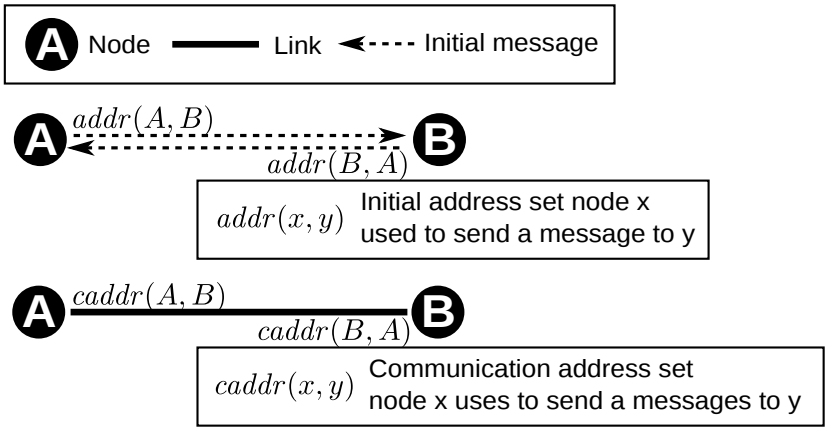


Figure A.2 – Establishing a link between nodes using the initial address-sets

priorize messages accordingly in local buffers on the device. If the device cannot comply with the maximum delay, the message is dropped.
The *strategy* parameter states the selection of the addresses in the address-set to send the message. Two modes are differentiated:

1. *random*: The random strategy selects one possible address from the address-set and sends the message. Thus, only single message is sent.
2. *multicast*: The multicast strategy sends the message to all adresses that can be extracted from the address-set. For example, when the address-set comprises two IP addresses and one TCP port, the message would be sent to both IP addresses using the same TCP port. Thus, duplicates of the same messages are sent to several addresses. If the *strategy* parameter is omitted, this is the default strategy.

The function sends the message *unreliable*, i. e., the message might get drop locally or in the network. Furthermore, there is *no flow control*. Congestion inside the network leads to loss of messages. The function may drop a message when it exceeds the maximum transfer unit (MTU) of the underlying network.

receive(callback): This function binds a callback for received messages. Whenever a message has been received, the callback function is called and the link-context and the message itself is passed, i. e.,

receive_callback(*message*, *link-context*)

monitor(*link-context*, $T_{link-timeout}$, callback): This function monitors a link using keep-alive messages. When no messages were received $T_{link-timeout}$ seconds, the callback is called and the link-context is passed:

monitor_callback(*link-context*)

If a link has no activity

A node receiving a message can respond to it by using the communication address-set included in the link-context. Using this interface nodes can exchange messages

using different network protocols, e. g., UDP/TCP over IPv4/6, Bluetooth RFCOMM, any many more.

Figure A.2 shows the establishment of a link between two nodes Ⓐ and Ⓑ. In the first step, both nodes use the initial address sets (IAS) $addr(A, B)$ and $addr(B, A)$ to send a message to each other. Subsequently, both nodes receive the message with the appropriate link-context. Finally, the nodes use the address-sets $caddr(A, B)$ and $caddr(B, A)$ to exchange messages over the *link* between them. The following sections provide implementation notes to this interface.

Implementation notes

The communication interface multiplexes all its method to several communication modules, i. e., implementations that support several protocols, e. g., TCP, UDP, or Bluetooth L2CAP. This section provides some implementation notes how to implement such modules exemplary for a unreliable datagram protocol, e. g., UDP, and, a reliable stream-based protocol, e. g., TCP. The main problem to solve in both cases is the transmission of the initial address sets (IAS). These address-sets are needed to provide a valid link context.

UDP Communication Module

To provide the functionality of the communication interface for a datagram protocol, i. e., UDP, the implementing module must take care of creating a link-context. Let X and Y be nodes. When node X sends a message to node Y, it knows the IAS $addr(X, Y)$ of node Y. To let node Y know its own IAS, node X needs to sent the IAS $addr(X, Y)$ to Y in the first message. This is important, as middle-boxes, like NAT routers, may change the destination address, so the IAS, can't be extracted from the packet. Furthermore, node Y needs to know the IAS of node X for the link-context. To this end, X includes any information needed to build the IAS in the first message, i. e., the server-port to construct the IAS together with the IP source address. When the first message is received by Y, the communication addresses $caddr(Y, X)$ of the link-context on Y, in case of UDP, is the same has the IAS, i. e., $caddr(Y, X) = addr(Y, X)$.

TCP Communication Module

In contrast to the UDP datagram module, the TCP communication module use the IAS to establish a new TCP connection to a node representing the link. Let X and Y be nodes, then node X uses the IAS $addr(X, Y)$ to establish a TCP connection to the link. Like the UDP module, it includes IAS $addr(X, Y)$, and server-port of node X for the link-context. When the first message is received by Y, the communication addresses $caddr(Y, X)$ is a pointer to node Y memory containing the socket-description of the TCP connection. To consider the *maxDelay* parameter in the send function, it is possible to schedule messages in an earliest-deadline-first (EDF) scheme into the TCP output buffer, so delay bound are met. Additionally, a limited number of concurrent TCP connections to the same node may be established to concurrently send messages.

To bootstrap an overlay network, i. e., finding another node that is running the P2P middleware/application, the following describes the necessary discovery functionality.

A.3 Discovery Interface

The discovery interface provides a generic interface to discover other nodes running the P2P middleware/application. Two key mechanisms are provided in the interface: first, a node can advertise that it wants to be discovered for a certain time. Second, an advertisement of a node may be discovered by another node. The communication and discovery layer holds a list of advertisements of other nodes and disseminates them randomly among known nodes. Thus, the discovery of an advertisement is a *random sampling* from all advertisements.

The following specifies the discovery interface in detail:

advertise(*time-to-live* [, *address-set*]): This function advertises that the node wants to be discovered by the given type string for a certain time (*time-to-live*) using the given address-set. The address-set is optional. If omitted, the IE address-set discovered by the node itself is used.

discover(discover_callback): This function discovers address-sets announced by other nodes. When an address set has been discovered by the function it calls the callback function and passes the type and the discovered address-set, i. e.,

discover_callback(*address-set*)

The address-set can be used to send messages to a new overlay neighbor.

Implementation notes

The discovery interface can be implemented using a variety of mechanisms. First of all, nodes in the local network can be discovered using multicast packets, Zeroconf (mDNS) or the Bluetooth service discovery protocol (SDP). All these mechanisms provide address-sets of nodes nearby.

Appendix B

Implementation

This chapter contains additional notes on implementation of the Connectivity Domain Measurement (CMP) and extended Virtual Ring Routing (eVRR) protocols.

B.1 Connectivity Measurement Protocol (CMP)

This section contains several implementation specifc details to the Connectivity Measurement Protocol (CMP).

Message Formats

This section proposes a binary formats for CMP's messages. CMP uses two message types: status update and triangle check messages. The following sections describe the formats.

Status Update

CMP sents status updates in each of its round when changed. Figure B.1 shows the binary format of the state update message. The first 32-bit of the message contains CMP's version, state, and flags. The first 4 bits denote CMP's version, for the first version the field is allocated with 0001. The state-field encodes CMP's current state. The allocations are as follows: HOLD =0000, AGREEING =0001, STABLE =0010. The CPA, CID Proposal Availibility bit is set to 1 if the message contains a valid CID proposal, and 0 otherwise.

Triangle Check

CMP uses triangle checks to detect if a node is a relay. To do this, CMP sends triangle check messages containg a hop count, a nonce, a source node identifier and a via underlay address. Figure B.2 shows the binary format of a triangle check message. The hop-field is a 8-bit integer and represents the hop count and the nonce-field contains a 24-bit integer. The src-field contains a 128-bit node identifier. Finally, the via-field contains an underlay address of variable size. The presence of the optional fields, src and via, is indicated by the message length.

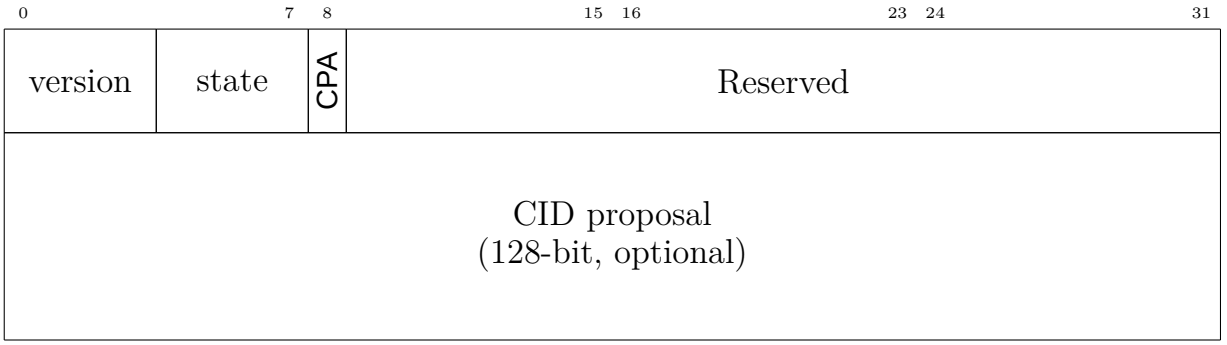


Figure B.1 – CMP's status update message format.

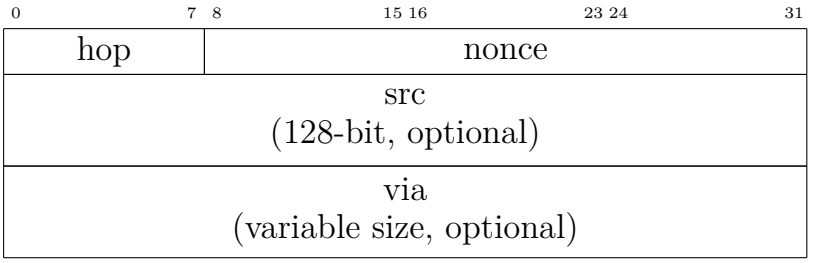


Figure B.2 – CMP's triangle check message format.

B.2 Tailored Routing (TRout)

This section gives hints for implemented Tailored Routing (TRout). It is the technical documentation for each of the modules TRout comprises and serves as guideline that helps implementing TRout. To understand this documentation its highly recommended to read the description of TRout's mechanisms in Section 5.3.

Route Updates and Queue Module

The route update queue collects updates to be sent. For this purpose the route update queue comprises a list of updates for each physical neighbor. When the queue is flushed, i. e., the updates should be sent, it assembles all route updates in one message for each physical neighbor, sends the message, and clears the list. Figure B.3 shows the binary format of a route update. Each message sent to a physical neighbor contains at least one of these updates. The route update starts with the update and route type. The update type may hold the following values, handled by the route maintenance module:

- Update (`0x01`) denotes a "usual" route update
- Teardown (`0x02`) denotes that the route should be torn down
- Forward (`0x03`) denotes that the route should be forwarded to the node with the destination id
- Notify (`0x04`) denotes that the update should be deliviered to the node that announced the destination id. It informs this node about a change in the network topology.

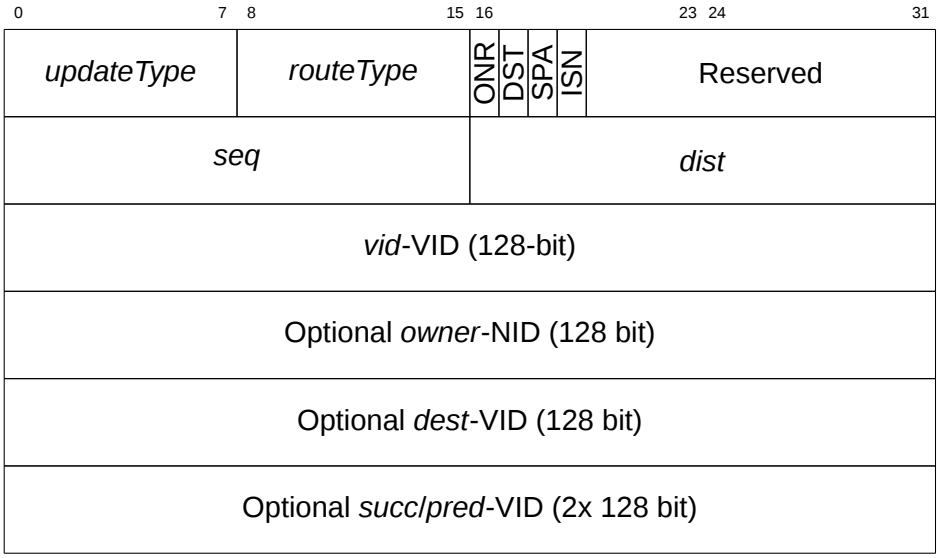


Figure B.3 – Route update record

– USED,UNUSED (0x08,0x09) denotes a notification that a node that a route is used/not used.

Additionally, the virtual ring module processes the following route types:

– DISCOVER-PRED,SUCC (0x10,0x11) denotes that the route update should be forwarded towards a closest predecessor or successor.

Finally, the anycast extension module processes the following route types:

– ANYCAST-ANNOUNCEMENT (0x20) denotes that the update should be forwarded along a path closest to the virtual identifier (VID) until it finds a route already present with a shorter route.

The update type field is followed by the route type. TRout currently knows only two types of routes: virtual ring (0x01) and anycast routes (0x02). The next word of 16-bits contains flags. These flags indicate whether optional fields exist in the route update. The ONR-bit indicates whether the *owner*-NID field is available. The DST-bit indicates whether the *dest*-VID field is available. The SPA-bit indicates whether the successor/predecessor-VIDs are available. The ISN-bit indicates whether the sequence number should be increased on in an *Notify* route update. The sequence number *seq* of the route and the distance *dist* to the node that announced the route follow after the flags. Measure of the distance is the the number of hops. The next field, the route's virtual identifier (VID), denotes the identity of the route. This completes the mandatory section of the route update. The presence of the next fields are indicated by the flags mentioned before. The first field contains the owner of the update, the *owner* node identifier NID. The second optional field holds the destination-VID *dest*. TRout uses this field to explicitly forward a route to a specific node. The third optional fields are successor- and predecessor-VID fields *succ*/*pred*. These fields contain the best-known successor-

and predecessor VIDs of the route VID. Linearization uses these two fields for optimizations.

The route update queue module has 3 basic methods:

queue.*send*(*nid*,*update*): queues a route *update* to be sent to the node with the node identifier (NID) *nid*.

queue.*clear*(*nid*): clears the route update queue of the node with the node identifier (NID) *nid*.

queue.*flush*(*nid*): packs the route updates for the node with the node identifier (NID) *nid* into a message, sents it to the node, and flushes the queue.

The contents of a route update anticipate some of the module's functionality. In the following the mechanisms are explained in further detail.

Route Maintenance Module

The route maintenance module eases setup and failure handling of routes. It draws its primary idea from the destination-sequenced distance vector (DSDV) protocol (cf.. Section 2.3.2). The main idea behind DSDV is, that each node announces routes identified by an identifier (VID) and distance together with a sequence number. When a node receives distance vectors from its neighbors it adapts the distance of an entry, if the distance is smaller, or, the sequence number is greater. This principle solves the problems that come with distance vector (DV) protocols: count-to-infinity problem and routing loops. This is the case, because a worse route is only accepted by other nodes when the origin node has chosen a new sequence number. Because routes become only longer when the network topology changes. Thus, the sequence number should be increased at least on every topology change to converge to a stable state.

The route maintenance module provides functional tools to build routes between nodes:

1. *Route forwarding:* route forwarding allows a node A to forward a route from node X to node Y. For this purpose, node A needs to know the route to Y as well for forwarding. After forwarding node Y knows a route to node X—most likely with node A as one intermediate node.
2. *Route usage recording:* Each node keeps a record of the routing-table entries referenced by its physical neighbors, i. e., a node knows that it is the next-hop in the routing-table entry of a physical neighbor. This allows efficient route *teardowns*.
3. *Automatic route teardowns:* By using the route usage record, a node can *teardown* routes easily by notifying the physical neighbors about a invalid route when a link fails. The notified nodes spreads the message to other nodes that use the same route to remove the entry as well. This continues until all entries of the route are removed from all routing-tables. At the same time, the destination node is informed about the failure, so it can re-establish a new route.
4. *Route notifications:* route notifications notify nodes that announced a route about topology changes or other events.

Name	Description
x.*vid*	Identifier (VID)
x.*owner*	Owner node identifier (NID, optional)
x.*type*	Route type (ring, or anycast)
x.*dist*	Distance to node that announced the route
x.*seq*	Sequence number
x.*nexthop*	The physical node the update came from
x.*usedby*	Neighbors using this entry
x.*unused*	Number of rounds the entry was not used
x.*neighbor*	Flag: indicates a entry from a physical neighbor
x.*changed*	Flag: indicates a change
x.*forwarded*	Flag: indicates a forwarded entry
x.*forwardednew*	Flag: indicates a newly forwarded entry
x.*succ*, x.*pred*	Predecessor / successor VIDs (optional)

Table B.1 – Contents of a routing-table entry x of TRout

Before describing these mechanisms in further detail, the following describes contents of a routing-table entry.

Routing-Table Entry

The contents of TRout's routing-table entries are a common denominator for all routing-table construction mechanisms, namely, discovery, linearization, and anycast extension implemented in TRout. Table B.1 summarizes the contents of a routing-table entry. It comprises 5 sections. The first section describes the fields of a typical distance vector protocol: the *vid*-field denotes the identifier of the route, the *type*-field denotes the route type, i. e., if the route is part of the virtual ring or the anycast extension. The *dist*-field denotes the number of hops to the node that announced the route. The *seq*-field denotes the sequence number in terms of the DSDV-principle, finally, the *nexthop*-field denotes node identifier of a physical neighbor closer to a node that announced the route.

The second section stores a usage record of the entry. For this purpose, the *usedby*-field denotes the subset of physical neighbors of a node X whose routing-table contains the same route with node X's in the *nexthop* field. The *unused*-field denotes the number of rounds the *usedby* set was empty *and* the entry has not been changed.

The third section stores several flags. The *neighbor*-flag denotes whether the route was announced from a physical neighbor, i. e., when $dist = 1$. The *changed*-flag indicates a change of the entry during a protocol round. The *forward*-flag indicates that the route has been established using route forwarding. The *forwardnew*-flag indicates that the entry is new and has been *forwarded*, and is ready for linearization.

The fourth section stores the closest successor and predecessor identifiers for the route identifier. Those two fields are only used for optimizing linearization.

Basic Routing-Table Management

For routing-table management, the route maintenance module provides several methods to add, and remove VIDs owned by the node to its routing-table. Furthermore, it has methods that allow the query of routing-table entries by VID and owner-NID. The following describes these methods:

routes.*addId*(*vid*,[*owner*],*type*) → entry: adds a new entry the routing-table of the given VID, type, i. e., ring or anycast, and owner. Its purpose is to add new routes to be announced by the node with the given VID to the routing-table. As this is a new entry, the *changed*, *forwarded*, *forwardednew*-flags are set to *true*, the distance *dist* is set to 0. The method also informs listeners about the newly added entry.

routes.*removeId*(*vid*,[*owner*]): removes a routing-table entry with the given VID, and owner-NID. The method also informs listeners about the removed entry. If one of the physical neighbors uses the route of this routing-table entry, teardown messages are sent to completely remove the route from the network.

routes.*get*(*vid*,[*owner*],[*ensure*]) → entry: returns the routing-table entry for the respective VID and owner-NID, or *nil* when the entry is not available. If *ensure* is *true*, a new routing-table is allocated and returned. If the owner is not given, it returns the entry that has an unspecified owner (*nil*).

routes.*getAll*(*vid*) $\rightarrow \{x \in \mathsf{R} | x.\mathit{vid} = \mathit{vid} \wedge x.\mathit{dist} = 0\}$: returns a set of routing-table entries with the given VID. This includes VIDs with different owners.

These methods allow routing-table access from the other modules, e. g., adding or removing new VIDs, and querying specific routing-table entries.

Neighbor Management

The route maintenance keeps track of physical neighbors and manages the changes in the routing-tables when the physical neighborhood changes. In case of the

```
neighbors := neighbors\{nid};
foreach x ∈ R:
    if x.nexthop = nid: remove(x);
foreach x ∈ R:
    if x.dist = 0: continue; /* do not remove my own entries */
    if ¬ nid in x.usedby: continue;
    notify(x.id, x.owner);
queue.clear(nid);
increaseSeq();
notifyListeners();
```

Algorithm B.1 – The routes.*removeNeighbor*(*nid*) method's pseudo-code. R denotes the routing-table. neighbors the set of NIDs of the physical neighbors.

unstructured relay overlay, this is the case, when a new overlay link is established or fails.

routes.*addNeighbor*(*nid*): informs the route maintenance module that the node as a new neighbor with the given NID. In consequence the route maintenance increases the sequence number of all routing-table entries and informs listeners about the new neighbor.

routes.*removeNeighbor*(*nid*): informs the route maintenance module that a node's neighbor left the network or failed. Consequently, all routing-table entries that use this neighbor as next-hop are removed from the routing-table. Additionally, the nodes that announced routes which included the neighbor are notified about the failure, so they can update their sequence number for the route to restore it if required. Subsequently, all sequence numbers of the node are increased and the listeners informed about the removed neighbor. Algorithm B.1 describes this method in further detail.

routes.*getNeighbors*() $\rightarrow$ {nid1, ..., nidn}: returns the set of known NIDs of all neighbors.

These methods are primarily called from the unstructured relay overlay, when new links are established or fail.

The route maintenance module has a primary *process*(*nid*,*u*) method that demuliplexes a incoming route update u from a node with a node identifier (NID) *nid*. The following uses the notation $process_x(nid, u)$ for the demultiplexed handling of a route update u with type x, i. e., $u.routeType = x$. Furthermore, the notation $x_y \leftarrow z$ denotes that field y in x gets replaced with the value of z.

Route Update Processing

When the route maintenance module receives a "usual" route update from another node, or module, the route maintenance module processes the route update as follows:

routes.*process*$_{\text{Update}}$(*nid*,*u*): The method, first checks whether the route update is on the blacklist, i. e., has been removed recently. If this is the case, the route update is discarded. If not on the blacklist, the method searches for an entry e for the respective VID $u.vid$ and owner $u.owner$. If not available, the routing-table entry e is created using the routes.*get* method. Subsequently, the route update is adapted to the routing-table entry, one of the following conditions are true:

1. The routing-table entry is *new*.
2. The sequence number e.*seq* of the routing-table entry is *smaller* than the sequence number u.*seq* of the route update.
3. The sequence numbers of routing-table entry and route update are equal, but the distance u.*dist+1* of the route update is smaller than the distance of the routing-table entry.

The adaptation of the route update basically involves copying the fields from the route update to routing-table entry. However, some fields need special attension. First, method assigns the distance field e.*dist* of the routing-table entry with the

distance of the route update plus 1, because the route update traversed one more hop. The e.*neighbor* of the routing-table entry is set *true*, when $u.dist$ equals 0, i. e., is a physical neighbor. The $e.nexthop := nid$ is set to the node identifier the route update came from. Finally, the $e.changed := true$ flag is set, because the routing-table entry has been changed, and, the $e.forwarded, e.forwardednew$-fields are set *true*, when the route update came from a physical neighbor. To inform the physical neighbors about the usage of the route update, the method sends USED/UNUSED-route updates to the respective physical neighbors. Furthermore, it informs listening modules about the changed or new entry. Algorithm B.3 describes this method in further detail.

The route update processing already includes notifications about the usage of the route updates to the physical neighbors. These records are used to remove orphaned or inconsistent routing-table entries from the network. The following introduces the necessary mechanisms to do this.

Route Usage Records and Blacklist

When a route update has been adapted by a node, i. e., updated its routing-table entry with the update, it informs the physical neighbor about this using USED and UNUSED route updates. This allows a consistency check and the removal of unused entries from the network. Furthermore, the route maintenance holds a blacklist of recently removed entries to inhibit the re-adoption of old routing-table entries

```
if isBlacklisted(update):
    | queue.send(nid, u|updateType ← UNUSED);
    | return;
e ← get(u.vid,u.owner,true);
if (u.seq > e.seq) ∨ ((u.dist + 1) < e.seq ∧ u.seq = e.seq) ∨ (e is new):
    | /* Inform neighbor when if next-hop changes */
    | if u.nexthop ≠ e.nexthop:
    |     | queue.send(e.vid, e| updateType ← UNUSED);
    | copy contents from u to e;
    | e.dist ← u.dist + 1;
    | e.changed ← true;
    | e.nexthop ← nid;
    | e.neighbor ← (u.dist = 0);
    | e.forwarded ← e.forwarded ∨ e.neighbor;
    | e.forwardednew ← e.forwardednew ∨ e.neighbor;
    | /* Inform neighbor that route update is used */
    | queue.send(nid, u|updateType ← USED);
else:
    | queue.send(nid, u|updateType ← UNUSED);
```

Algorithm B.2 – The routes.$process_{\text{UPDATE}}(nid,u)$ method's pseudo-code.

in routes.*process*$_{\text{UPDATE}}$. The following methods implement the functionality of the route usage records and blacklist:

routes.*process*$_{\text{Used}}$(*nid*,*u*): queries the entry according to the update u from the routing-table using the routes.*get* method. When found, it adds the node's identifier *nid* of the physical neighbor to the *usedby*-field in the routing-table entry. If not found, it initiates a teardown of the route to the neighbor. This ensures that the neighbor wipes invalid routes from its routing-table.

routes.*process*$_{\text{Unused}}$(*nid*,*u*): proceeds like routes.*process*$_{\text{USED}}$, but removes the node identifier f rom *usedby*-field in the routing-table entry, and does *not* initiate a teardown.

routes.*consistencyCheck*(): re-sends USED-updates of all routing-table entries to the corresponding next-hop, physical neighbors. If any inconsistency is detected, the physical neighbors will respond with a teardown of the route. This ensures validity of routes after checking consistency.

routes.*addToBlacklist*(*entry*): adds routing-table entry to the blacklist. If a entry with the same *vid* and *owner* is already blacklisted, it overwrites the entry when *entry* has a greater sequence number *seq*.

routes.*isBlacklisted*(*u*): checks whether the route update is on the blacklist. This is the case, when the blacklist contains a entry with the same *vid*, *owner*, and a *greater* sequence number *seq*.

routes.*maintenance*(*ttl*): cleans up the routing-table when unused entries are found. Every call to this method increases the *unused*-field of each routing-table entry that has not been changed and is not used by any physical neighbor. When called every protocol round, the *unused*-field counts the rounds the routing-table entry is not used. If it exceeds the time-to-live *ttl*, then the method asks listing modules, whether the entry is still needed. If this is not the case, it removes the entry. At the same time it clears the blacklist from entries older than *ttl* rounds.

Route Teardown and Notifications

Route teardowns are used when nodes withdraw their announced routes, i. e., by calling *removeId* and when nodes leave the network or fail. To ensure the validity of the routes, the teardown mechanism removes routing-table entries in the network that rely on a certain routing-table entry on a node. This mechanism highly relies on the route usage record maintained with each routing-table entry. To notify nodes about a change in its announced routes, e. g., for increasing the entry's sequence number, the notify mechanism forwards a notification route update to the announcing node.

routes.*remove*(*entry*): removes a routing-table entry. If the routing-table entry is still part of a active route, routes that depend on this entry are torn down using a teardown route update. Furthermore, it informs the next-hop neighbor that the routing-table entry has been removed using a UNUSED-route update. Algorithm B.3 describes this method in further detail.

routes.*notify*(*entry*,*incSeqNo*): sends a notify route update to the node that announced the route the entry belongs to. A respective routes.*process*$_{\text{NOTIFY}}$(*nid*,*u*)

method handles the notify route updates and forwards them until it reaches the announcing node. If *incSeqNo* is set *true*, the sequence number of the routing-table entry on the announcing node is increased.

routes.*teardown*(*nid*,*entry*): sends a teardown message to a physical neighbor with node identifier *nid* for the routint-table *entry*. A respective routes.$process_{\text{TEARDOWN}}(nid,u)$ handles the teardown route updates and removes the routing-table entry with the *remove*-method. This method subsequenly initiates further teardowns until all affected nodes remove the routing-table entries of the nodes relying on the entry, i. e., is on the route to the annuncing node, from the network.

Route Forwarding

Route forwarding is an important building-block for the linearization algorithm. Briefly, linearization tries to forward "better" routes to other virtual nodes. The functionality is implemented using two methods:

routes.*forward*(*entry*,*toEntry*): fowards a route update extracted from the routing-table *entry* along the route to the node that announced *toEntry*.

routes.$process_{\text{Forward}}$(*nid*,*u*): calls the routes.$process_{\text{UPDATE}}$ method to add the route to node's routing-table and forwards the route update further along the path to *toEntry*. If a shorter route is found, i. e., a routing-table entry that has a smaller distance to the same destination, the forward route update will be replaced with the data extracted from this routing-table entry.

Listener Interface

The route maintenance module notifies the virtual ring maintenance and anycast extension module about new, changed, or deleted routing-table entries. Furthermore, it asks the modules if a specific routing-table entry is still

routes.*registerListener*(*listener*): registers a module that listens to routing-table changes.

listener.*entryUpdate*(*entry*,*newEntry*): is called by the route maintenance when a routing-table entry has been added or changed.

listener.*entryRemove*(*entry*): is called by the route maintenance when a routing-table entry has been removed.

listener.*entryNotify*(*entry*): is called when a notify route update has been received.

Inform listeners;
foreach nid **in** *none.entry*usedby:
  | *teardown*(nid,*entry*);
queue.*send*(*none.entry*nexthop, $entry_{|updateType \leftarrow \text{UNUSED}}$);
addToBlacklist(*entry*);
Remove entry with *vid* and *owner* from routing-table;

Algorithm B.3 – The routes.*remove*(*entry*) algorithm in pseudo-code.

listener.*entryNeeded*(*entry*) → *bool*: is called to check whether the entry is still needed. Implenenting modules need to return *true* when the entry should not be removed.

The next section describes the virtual ring maintenance which implements this interface and registers itself to the route maintenance module.

Virtual Ring Maintenance

The virtual ring maintenance module builds a virtual ring using the route maintenance module. First, it adds a routing-table entry with its own node identifier NID to the routing-table using the routes.*addId*-method. Then it employs two mechanisms to establish virtual links to its neighbors: discovery and linearization. The mechanisms are triggered by TRout's main-loop to vring.*maintenance*:

vring.*maintenance*(): maintains the virtual ring routing-table. TRout's calls this method each T_{round} seconds from its main-lopp. First, the method sends its own VIDs to its physical neighbors if changed. This way, physical neighbors get to know each other. Subsequently, it calls the maintenance methods of the selected TRout-mode *mode*, i. e., DISCOVERY or LINEARIZATION. Finally, it increases the stable-counter and resets the change flags in the virtual ring table vringTable. Algorithm B.4 describes this method in further detail.

The *stable*-counter counts the number of rounds, no changes affected the virtual links to the virtual ring neighbors. The DISCOVERY-mode uses this counter to reduce the number of discovery messages sent each round.

As mentioned before, the virtual ring maintenance module reacts on notifications from the route maintenance module. For this purpose, it implements the *listener*

```
/* Iterate over the NIDs of my neighbors */
foreach nid in routes.getNeighbors():
    /* Inform phyical neighbors about my ring-VIDs */
    foreach e in vringTable:
        if (¬e.dist = 0 ∧ e.changed): continue;
        e.usedBy ← m.usedBy ∪ {nid};
        e.unused ← 0;
        queue.send(nid, e|updateType ← UPDATE);
/* Call maintenance of the specific mode */
if mode = DISCOVERY: vring.maintainDiscovery();
if mode = LINEARIZATION:vring.maintainLinearization();
/* Increase stability counter and reset flags */
stable ← stable + 1;
resetFlags();
```

Algorithm B.4 – Pseudo-code of vring.*maintenance*(). The main maintainance method for constructing the virtual ring. Called each T_{round} seconds in TRout's main-loop.

interface of the route maintenance module and registers itself to get notifications about routing-table changes. The virtual ring maintenance module keeps a ascending, ordered list of routing-table entries that belong to the ring in vringTable, i. e., *routeType*=RING. This list is kept up-to-date using the implementation of the *listener*-interface. Furthermore, it builds virtual links to virtual ring neighbors. More formally, the virtual ring maintenance module implements the *listener*-interface as follows:

listener.*entryUpdate*(*entry*, *newEntry*): returns immediately when $entry.routeType \neq$ RING, i. e., not a virtual ring routing-table entry, or $newEntry = false$, i. e., not a new entry. Otherwise, the method inserts *entry* into the virtual ring routing-table vringTable at the appropriate position, so the table stays in ascending order. If the *entry* denotes a route to a new virtual ring neighbor, it forwards its own routing-table entry, i. e., $dist = 0$ and $vid =$ nid to the *vid* of this entry. This way, the new virtual neighbors learns a route to the node. Subsequently, the method calls vring.*linearizationNotify*() or vring.*discoveryNotify*() depending on TRout's *mode*, to handle the new virtual neighbor.

listener.*entryRemove*(*entry*): does almost the same as listener.*entryUpdate*(*entry*, *newEntry*) only the other way round. It first calls vring.*linearizationNotify*() or vring.*discoveryNotify*() depending on TRout's *mode*, to handle the removed virtual neighbor. Then, it removes the entry from vringTable. Because a node or link failure usually causes the removal of a virtual neighbor, the method increases all sequence numbers of virtual ring routing-table entries using the routes.*increaseSeq*-method.

listener.*entryNotify*(*entry*): resets the stable-counter when $entry.routeType =$ RING.

listener.*entryNeeded*(*entry*) $\rightarrow$ *bool*: returns *true*, if the *entry* denotes leads to a virtual neighbor or *false*, if not, or when $entry.routeType \neq$ RING. Thus, unused routes to not leading to a virtual neighbor are removed from the routing-table.

Using these preparations leaves 2 methods, i. e., for maintenance and handling change notifictions, for each of TRout's modes, i. e., DISCOVERY and LINEARIZATION to implement. These two mechanisms do not build a virtual ring, but a virtual ordered path or line connecting all virtual nodes beginning with the lowest VID and ending with the highest VID. The virtual ring can be builtby connecting the ends of the path to a ring.

Discovery

The discovery mechanism finds virtual line neighbors by iteratively searching and discovering closer virtual neighbors, i. e., closer successor on the left, and predecessor on the right. Two methods implement this meachnism:

vring.*maintainDiscovery*(): sends discovery route updates towards the D_α randomy chosen neighbors. To inhitbit that discovery route updates are sent to a physical neighbor more than once, it keeps a set of physical neighbors considered already in the process. Furthermore, the stable-counter is used to lineary delay sending new discovery routes. It delays new discovery route updates by the number of rounds the virtual neighborhood is stable and no notification has been

received. This highly reduces signalling overhead when there are no changes in the network topology. Algorithm B.5 describes this method in further detail.

vring.*process*$_{\text{Discovery-Succ/Pred}}$(*nid*,*u*): handles the discovery route updates sent by *maintainDiscovery*. It adds a routing-table entry that leads towards the announcing node and forwards the discovery route update to a better successor, or predecessor if possible. If the node is the closest one, it forwards a route to itself back to the node that sent the discovery message. Thus, a virtual link is established between them. The pseudo-code in Algorithm B.7 shows this method in further detail.

The discovery mechanism adaptively reduces the number of updates sent by using the stable-counter. To ensure that TRout adapts quickly in case of network changes, the stable-counter needs a reset, whenever the virtual neighborhood changes. In this case, as explained before, the *notifyDiscovery* is called:

vring.*notifyDiscovery*(*entry*): informs virtual neighbors about changes in the network. For this purpose, it uses the routes.*notify* methods to send a notification route update to the predecessor and successor. The route maintenance module then calls the listener.*entryNotify* method of the virtual ring maintenance which resets the stable-counter to zero.

This completes the description of the virtual ring maintenance with the discovery mechanism. The next section describes the linearization mechanism to build the virtual ring.

```
/* Delay sending discovery messages when stable. */
discoveryDelay ← discoveryDelay + 1;
if discoveryDelay ≤ stable/2: return;
discoveryDelay ← 0;
/* Send discovery messages */
foreach e in vringTable:
    /* Not the node identifier of this node? -> continue */
    if e.dist ≠ 0: continue;
    /* Send discovery route updates to D_α neighbors chosen randomly. */
    R ← getNeighbors();
    foreach i in (1, ..., D_α + 1):
        if |R| = 0: break;
        r ← "random neighbor from R";
        R ← R\{r};
        queue.send(r.nextHop, e_|updateType ← DISCOVERY-PRED);
        queue.send(r.nextHop, e_|updateType ← DISCOVERY-SUCC);
```

Algorithm B.5 – The pseudo-code of vring.*maintainDiscovery*().

Linearzation

Linearization is a self-stabilizing algorithm introduced by by M. Onus et al. [71] that sorts a graph where each vertex holds an identifier to a straight line sorted ascendingly by the identifiers.Self-stabiliting means, that the algorithm can recover from *any* state. A problem of using linearization to build a virtual path is concurrency [40]. The original algorithm was designed for iterative processing of graphs. In a distributed setting, linearization is performed in parallel on many nodes. This leads to a high amount of redundant steps in the worst-case. To lower the redundancy, TRout uses a simple method: it only linearizes left- and right-routes if the linearization neighbor is closer to the VID than the VIDs linearized before. TRout uses the *succ*, *pred*-fields in the route update for this purpose. Like the discovery meachnisms, two methods are responsible for linearization:

vring.*maintainLinearization*(): first, finds a pair (a,b) of forwarded routing-table entries in vringTable, so that a.*vid* < b.*vid*. Second, it checks whether one of these entries has been changed or could profit from knowing the other route by checking the respective *succ*/*pred*-fields. If the case, it forwards routes when both are left neighbors and a is a newly forwarded entry, or when both are right neighbors and b is a newly forwarded entry. This implements the linearization algorithm. When forwarded, the method adapts the *succ*/*pred*, and *forwarded-New* fields. The pseudo-code in Algorithm B.6 shows this method in further detail.

```
foreach aIndex in (0, ..., vringTable.size − 1):
    /* Find left-neighbor */
    a ← vringTable[aIndex];
    if ¬a.forwarded ∨ a.dist = 0: continue;
    foreach bIndex in (aIndex + 1, ..., vringTable.size − 1):
        b ← vringTable[bIndex];
        /* Find closest linearization neighbor */
        if ¬b.forwarded: continue;
        ifa or b have changes and provide a new succ/pred to eachother:
            forward ← false;
            /* Newly forwarded entry, and a,b left neighbors? -> forward */
            if a.forwardednew ∧ (a and b are left-routes):
                a.forwardednew ← false; forward ← true;
            /* Same as above, only mirrored */
            if b.forwardednew ∧ (a and b are right-routes):
                a.forwardednew ← false; forward ← true;
            /* Forward routes */
            if forward:
                Set new predecessor/successor entries for a and b;
                routes.forward(a,b); routes.forward(b,a);
```

Algorithm B.6 – The algorithm of vring.*maintainLinearization*() in pseudo-code.

vring.*notifyLinearization*(*entry*): initiates a re-linarization with its left/right-route depending if the entry itself is a right or left route. This is done by setting the *forwardNew* and *changed*-flags of the routing-table entry to *true*. Subsequently, vring.*maintainLinearization* will re-linearize the routing-table entry in the next protocol round.

Finding the Next-Hop

When the virtual path is complete, routing can be done greedily, as the virtual path represents a convex address space. The next-hop to a given VID is always denoted by the *nextHop*-field of the routing-table entry in vringTable that is closest to the given VID. Because the vringTable also includes routes that do not lead to a virtual neighbor, but also routes traversing the node, short-cuts are taken naturally. The next section adds anycast support to TRout using the virtual ring maintaince module.

```
/* Add route to node where the discovery came from */
routes.process_UPDATE(nid,u_|updateType ← UPDATE);
/* Get source/destination virtual ring routing-table entry indexes */
srcIndex ← indexOf(none.uvid); dstIndex ← indexOf(none.udest);
/* If one of both entries are not available-> skip */
if srcIndex=-1 ∨ dstIndex=-1: return;
/* Get source/destination routing-table entries */
src ← vringTable[srcIndex]; dst ← vringTable[dstIndex];
/* Check for successor / predecessor */
if (u.updateType = DISCOVERY-SUCC) ∧ (srcIndex + 1) < vringTable.size:
    | dst ← vringTable[srcIndex+1];
if (u.updateType = DISCOVERY-PRED) ∧ (srcIndex − 1) ≥ 0:
    | dst ← vringTable[srcIndex-1];
/* The node is the closest one? -> forward my route to source, stop */
if dst.dist = 0:
    | src.forwarded ← true;
    | routes.forward(dst.vid, src.vid);
    | notifyDiscovery(src);
    | return;
/* Split horizon */
if dst.nextHop = nid: return;
/* Route message */
newRoute ← src_|updateType ← u.updateType;
newRoute.dest ← dst.vid;
src.usedBy ← src.usedBy ∪ {dst.nextHop};
queue.send(dst.nextHop,newRoute);
```

Algorithm B.7 – The vring.$process_{\text{DISCOVERY-SUCC/PRED}}$ (nid,u) algorithm in pseudo-code.

Anycast

The anycast extension uses the virtual ring routing to add additional routes that lead to anycast VIDs. Each node may announce additional, arbitrary anycast VIDs. These announcements are forwarded along the routing paths of the virtual ring until they hit the node with the closest NID to the annouced VID. To differentiate the routes the anycast extension assigns the *owner*-field with the NID of the announcing node. This makes each route unique. If a subset of nodes annouce the same VID the resulting routes intersect at a certain point. In this case, only the shortest (or the one with the lowest owner-NID to break ties) announcement is forwarded to the virtual node with the closest NID.

When routing to an anycast VID, the virtual ring routing is used, until a anycast route is found. Then, the message follows this path to the announcing node. Because the anycast extension only announces the shortest-route to the VID using the virtual ring routing, close nodes announcing the same VID are preferred. This reduces stretch.

Bibliography

[1] The universal plug and play forum. http://upnp.org, 2011. Last visted on 21. November 2011.

[2] The utorrent bit-torrent client. http://utorrent.com, 2011.

[3] Y. Afek, E. Gafni, and M. Ricklin. Upper and lower bounds for routing schemes in dynamic networks. In *30th Annual Symposium on Foundations of Computer Science 1989.*, pages 370–375. IEEE, Oct. 1989.

[4] R. Baldoni, M. Platania, L. Querzoni, and S. Scipioni. Practical uniform peer sampling under churn. In *Proceedings of the 2010 Ninth International Symposium on Parallel and Distributed Computing*, ISPDC '10, pages 93–100, Washington, DC, USA, 2010. IEEE Computer Society.

[5] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[6] I. Baumgart. P2pns: A secure distributed name service for p2psip. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 480–485, Washington, DC, USA, 2008. IEEE Computer Society.

[7] P. Baumung. Tram: Cross-layer efficient application-layer multicast in mobile ad-hoc networks. In *Wireless Communications and Networking Conference, 2007.WCNC 2007. IEEE*, pages 4069 –4073, Mar. 2007.

[8] BitTorrent. The bittorrent website. http://www.bittorrent.com/, June 2011.

[9] R. Bless, C. Hübsch, S. Mies, and O. Waldhorst. The Underlay Abstraction in the Spontaneous Virtual Networks (SpoVNet) Architecture. In *Proc. Next Generation Internet Networks (NGI)*, pages 115–122, Krakow, Poland, Apr. 2008.

[10] B. Bollobás and W. F. de la Vega. The diameter of random regular graphs. *Combinatorica*, 2(2):125–134, 1982. Springer.

[11] S. P. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: design, analysis and applications. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 13-17 March 2005, Miami, FL, USA*, pages 1653–1664. IEEE, 2005.

[12] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications ACM*, 16(9):575–577, 1973.

[13] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual Ring Routing: Network Routing Inspired by DHTs. In *Proc. ACM SIGCOMM 2006*, pages 351–362, Pisa, Italy, 2006.

[14] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica. ROFL: Routing on Flat Labels. In *Proc. ACM SIGCOMM 2006*, pages 363–374, Pisa, Italy, 2006.

[15] M. C. Caesar. *Identity-based routing*. PhD thesis, EECS Department, University of California, Berkeley, Sept. 2007.

[16] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like P2P systems scalable. In *Proceedings of SIGCOMM 2003*, pages 407–418. ACM Press, 2003.

[17] S. Consortium. SpoVNet: An Architecture for Supporting Future Internet Applications. In *Proc. 7th Würzburg Workshop on IP: Joint EuroFGI and ITG Workshop on Visions of Future Generation Networks (EuroView 2007)*, Würzburg, Germany, July 2007.

[18] L. Correia, H. Abramowicz, M. Johnsson, K. Wuntsel, and J. Silva. *Architecture and Design for the Future Internet: 4WARD EU Project*. Signals and Communication Technology. Springer, 2011.

[19] C. Cramer and T. Fuhrmann. Isprp: a message-efficient protocol for initializing structured p2p networks. In *Proceedings of the 24th IEEE International Performance Computing and Communications Conference, IPCCC 2005, April 7-9, 2005, Phoenix, Arizona, USA*, pages 365–370, 2005.

[20] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. Plutarch: An Argument for Network Pluralism. In *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pages 258–266, Karlsruhe, Germany, 2003.

[21] L. D'Acunto, J. Pouwelse, and H. Sips. A measurement of nat & firewall characteristics in peer-to-peer systems. In L. W. Theo Gevers, Herbert Bos, editor, *Proc. 15-th ASCI Conference*, pages 1–5, P.O. Box 5031, 2600 GA Delft, The Netherlands, June 2009. Advanced School for Computing and Imaging (ASCI).

[22] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. E. Sturgis, D. C. Swinehart, and D. B. Terry. Epidemic algorithms for replicated

database maintenance. *Operating Systems Review*, 22(1):8–32, 1988. ACM Press.

[23] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 10.1007/BF01386390.

[24] J. R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, 2002. Springer-Verlag.

[25] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631 (Informational), May 1994. Obsoleted by RFC 3022.

[26] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In O. Cheong, K.-Y. Chwa, and K. Park, editors, *Algorithms and Computation, 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, volume 6506 of *Lecture Notes in Computer Science*, pages 403–414. Springer, 2010.

[27] J. Eriksson, M. Faloutsos, and S. V. Krishnamurthy. Peernet: Pushing peer-to-peer down the stack. In M. F. Kaashoek and I. Stoica, editors, *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, volume 2735 of *Lecture Notes in Computer Science*, pages 268–277. Springer, 2003.

[28] J. Eriksson, M. Faloutsos, and S. V. Krishnamurthy. DART: Dynamic Address RouTing for scalable ad hoc and mesh networks. *IEEE/ACM Transactions on Networking*, 15(1):119–132, 2007.

[29] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29:251–262, August 1999.

[30] D. Fernholz and V. Ramachandran. The diameter of sparse random graphs. *Random Structures and Algorithms*, 31(4):482–516, 2007. Wiley.

[31] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32:374–382, April 1985.

[32] R. W. Floyd. Algorithm 97: Shortest path. *ACM Communications*, 5(6):345–, June 1962.

[33] B. Ford. Scalable Internet Routing on Topology-Independent Node Identities. Technical Report MIT-LCS-TR-926, Massachusetts Institute of Technology, 2003.

[34] B. Ford. *UIA: A Global Connectivity Architecture for Mobile Personal Devices.* PhD thesis, Massachusetts Institute of Technology, September 2008.

[35] L. R. Ford. Network flow theory, rand corporation, santa monica, ca. 1956.

[36] P. Francis and R. Gummadi. IPNL: A NAT-Extended Internet Architecture. In *Proc. ACM SIGCOMM 2001*, pages 69–80, San Diego, CA, USA, 2001.

[37] G. N. Frederickson and R. Janardan. Designing networks with compact routing tables. *Algorithmica*, 3:171–190, 1988. Springer.

[38] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica. Non-transitive Connectivity and DHTs. In *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*, pages 55–60, Berkeley, CA, USA, 2005. USENIX Association.

[39] T. Fuhrmann. Scalable routing for large self-organizing networks. In A. K. Bandara and M. Burgess, editors, *Inter-Domain Management, First International Conference on Autonomous Infrastructure, Management and Security, AIMS 2007, Oslo, Norway, June 21-22, 2007, Proceedings*, volume 4543 of *Lecture Notes in Computer Science*, page 234. Springer, 2007.

[40] D. Gall, R. Jacob, A. W. Richa, C. Scheideler, S. Schmid, and H. Täubig. Time complexity of distributed topological self-stabilization: The case of graph linearization. In A. López-Ortiz, editor, *LATIN 2010: Theoretical Informatics, 9th Latin American Symposium, Oaxaca, Mexico, April 19-23, 2010. Proceedings*, volume 6034 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2010.

[41] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Transactions on Computers*, 31(1):48–59, 1982.

[42] B. Heep. *Effizientes Routing in strukturierten P2P Overlays*. KIT Scientific Publishing, Feb. 2012.

[43] M. Holdrege and P. Srisuresh. Protocol Complications with the IP Network Address Translator. RFC 3027 (Informational), Jan. 2001.

[44] C. Hübsch, C. Mayer, S. Mies, R. Bless, O. Waldhorst, and M. Zitterbart. Reconnecting the Internet wirh ariba: Self-Organizing Provisioning of End-to-End Connectivity in Heterogeneous Networks. In *Software Demonstration at ACM SIGCOMM 2009*, Barcelona, Spain, 2009.

[45] C. Hübsch, C. Mayer, S. Mies, R. Bless, O. Waldhorst, and M. Zitterbart. Using Legacy Applications in Future Heterogeneous Networks with ariba. In *Proceedings of IEEE INFOCOM*, San Diego, CA, USA, May 2010. Demo.

[46] C. Hübsch, C. P. Mayer, S. Mies, R. Bless, O. P. Waldhorst, and M. Zitterbart. Reconnecting the Internet with ariba: Self-Organizing Provisioning of End-to-End Connectivity in Heterogeneous Networks. *ACM SIGCOMM Computer Communication Review*, 40(1):131–132, jan 2010. SESSION: Best poster & demo abstracts from ACM SIGCOMM 2009.

[47] S. Jaiswal, A. L. Rosenberg, and D. F. Towsley. Comparing the structure of power-law graphs and the internet as graph. In *12th IEEE International*

Conference on Network Protocols (ICNP 2004), 5-8 October 2004, Berlin, Germany, pages 294–303, 2004.

[48] S. Janson, C. Lavault, and G. Louchard. Convergence of some leader election algorithms. *Computing Research Repository (CoRR)*, abs/0802.1389, 2008.

[49] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23:219–252, August 2005.

[50] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25(3):8, 2007.

[51] E. Jennings and L. Motyckova. A distributed algorithm for finding all maximal cliques in a network graph. In *LATIN '92, 1st Latin American Symposium on Theoretical Informatics, São Paulo, Brazil, April 6-10, 1992, Proceedings*, volume 583 of *Lecture Notes in Computer Science*, pages 281–293. Springer, 1992.

[52] G. P. Jesi and A. Montresor. Secure peer sampling service: The mosquito attack. In *Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, WETICE '09, pages 134–139, Washington, DC, USA, 2009. IEEE Computer Society.

[53] G. P. Jesi, A. Montresor, and M. van Steen. Secure Peer Sampling. *Computer Networking*, 54:2086–2098, August 2010.

[54] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS 2000, pages 565–, Washington, DC, USA, 2000. IEEE Computer Society.

[55] S. Kniesburges, A. Koutsopoulos, and C. Scheideler. Re-chord: a self-stabilizing chord overlay network. In *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures*, SPAA '11, pages 235–244, New York, NY, USA, 2011. ACM.

[56] A. Korman and D. Peleg. Dynamic routing schemes for general graphs. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 619–630. Springer, 2006.

[57] D. V. Krioukov, K. C. Claffy, K. R. Fall, and A. Brady. On compact routing for the internet. *Computing Research Repository (CoRR)*, abs/0708.2309, 2007.

[58] K. Kutzner and T. Fuhrmann. Using linearization for global consistency in ssr. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007.*, pages 1–5, Mar. 2007.

[59] L. lab. The p2p next streaming client. http://www.livinglab.eu/, June 2011.

[60] S. limited. The skype website. http://www.skype.com/, June 2011.

[61] J. C.-P. W. M. Abolhasan, B. Hagelstein. The babel homepage. http://www.pps.jussieu.fr/ jch/software/babel/, May 2011.

[62] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10*, pages 260–272. Springer, 2004.

[63] D. Malkhi, S. Sen, K. Talwar, R. F. Werneck, and U. Wieder. Virtual Ring Routing Trends. In *Proceedings of the 23rd international conference on Distributed computing*, DISC'09, pages 392–406, Berlin, Heidelberg, 2009. Springer-Verlag.

[64] C. P. Mayer, S. Mies, C. Hübsch, and O. P. Waldhorst. Ariba Virtual Network Substrate. http://www.ariba-underlay.org, 2009.

[65] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems: 1st International Workshop, IPTPS 2002. Revised Papers*, volume 2429/2002, pages 53–65, Cambridge,MA, USA, Mar. 2002.

[66] S. Mies, O. Waldhorst, and H. Wippel. Towards End-to-End Connectivity for Overlays across Heterogeneous Networks. In *Proc. Future-Net 2009, co-located with IEEE ICC 2009*, Dresden, Germany, June 2009.

[67] S. Mies and O. P. Waldhorst. Next Steps in Overlay Construction on Dynamic Heterogeneous Networks. Presentation at 4th GI/ITG KuVS Workshop on The Future Internet, Dec. 2009.

[68] S. Mies and O. P. Waldhorst. Autonomous Detection of Connectivity. In *Proceedings of the 11th International Conference on Peer-to-Peer Computing (IEEE P2P'11)*, pages 44–53, Sept. 2011.

[69] S. Mies and H. Wippel. Providing End-to-End Connectivity Across Heterogeneous Networks. Presentation at 8th Würzburg Workshop on IP: Joint EuroFGI and ITG Workshop on Visions of Future Generation Networks, July 2008.

[70] H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, New York, NY, USA, 2008.

[71] M. Onus, A. W. Richa, and C. Scheideler. Linearization: Locally self-stabilizing sorting in graphs. In *Proceedings of the Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007.

[72] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Computer Communication Review*, 24:234–244, Oct. 1994.

[73] B. Pittel. On spreading a rumor. *Journal on Applied Mathematics*, 47(1):213–223, 1987. SIAM.

[74] T. P.-V. project. The p2p-vpn website. http://www.p2pvpn.org/, June 2011.

[75] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[76] S. C. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: a public dht service and its uses. In R. Guérin, R. Govindan, and G. Minshall, editors, *Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Philadelphia, Pennsylvania, USA, August 22-26, 2005*, pages 73–84. ACM, 2005.

[77] U. Schmid, B. Weiss, and I. Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM Journal on Computing*, 38(5):1912–1951, 2009.

[78] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos. Power laws and the as-level internet topology. *IEEE/ACM Transactions on Networking*, 11(4):514–524, 2003.

[79] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, EW 11, New York, NY, USA, 2004. ACM.

[80] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy. Scalable routing on flat names. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 20:1–20:12, New York, NY, USA, 2010. ACM.

[81] Z.-L. Z. Sourabh Jain, Yingying Chen. Viro: A scalable, robust and namespace independent virtual id routing for future networks. In *Proceedings of the 30th IEEE International Conference on Computer Communications (IEEE INFOCOM 2011)*, pages 2381–2389. IEEE, 2011.

[82] T. B. special interest group. Bluetooth protocol specifications. https://www.bluetooth.org/Technical/Specifications/adopted.htm, June 2011.

[83] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663 (Informational), Aug. 1999.

[84] M. Steiner, T. En-Najjary, and E. W. Biersack. Long term study of peer behavior in the kad dht. *IEEE/ACM Transactions on Networking*, 17:1371–1384, October 2009.

[85] W. R. Stevens. *UNIX Network Programming: Networking APIs: Sockets and XTI*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1997.

[86] V. Stix. Finding all maximal cliques in dynamic graphs. *Computational Optimization and Applications, Springer*, 27:173–186, 2004.

[87] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM 2001*, pages 149–160, San Diego, CA, USA, 2001.

[88] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 189–202, New York, NY, USA, 2006. ACM.

[89] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger. On unbiased sampling for unstructured peer-to-peer networks. *IEEE/ACM Transactions on Networking*, 17:377–390, April 2009.

[90] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '01, pages 1–10, New York, NY, USA, 2001. ACM.

[91] N. Tölgyesi and M. Jelasity. Adaptive peer sampling with newscast. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, pages 523–534, Berlin, Heidelberg, 2009. Springer-Verlag.

[92] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques. 3106:161–170, 2004. Springer Berlin/Heidelberg.

[93] US-CERT. United States Computer Emergency Readiness Team. Vulnerability note vu no. 347812: Upnp enabled by default in multiple devices. http://www.kb.cert.org/vuls/id/347812, 2011. Last visted on 21. November 2011.

[94] A. Varga and R. Hornig. An Overview of the OMNeT++ Simulation Environment. In *Proc. 1st Int. Conf. on Simulation Tools and Techniques for Communications, Networks and Systems (Simutools 2008) Workshops*, pages 1–10, Marseille, France, 2008.

[95] A. Vázquez, R. Pastor-Satorras, and A. Vespignani. Large-scale topological and dynamical properties of the Internet. *Physical Review E*, 65(6):66130, 2002.

[96] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[97] C. Werle, S. Mies, and M. Zitterbart. On Benchmarking Routing Protocols. In *Proceedings of the 17th IEEE International Conference on Networks (ICON2011)*. IEEE, Dec. 2011.

[98] H. Wippel. Optimierung der Konnektivität von Overlays in heterogenen Netzen. Diplomarbeit, Institut für Telematik, Universität Karlsruhe (TH), 2008.