[DMH+05] *Vasilios Darlagiannis, Andreas Mauthe, Oliver Heckmann, Nicolas Liebau, Ralf Steinmetz;* **Role Specialization to Support Peer-to-Peer Heterogeneity;** The 2005 Networking and Electronic Commerce Research Conference (NAEC 2005), Lake Garda, Italien, Oktober 2005, S.

Role Specialization to Support Peer-to-Peer Heterogeneity

Vasilios Darlagiannis[†] Andreas Mauthe[‡] Oliver Heckmann[†] Nicolas Liebau[†] Ralf Steinmetz[†]

[†] *Technische Universität Darmstadt*
Multimedia Communications (KOM)
Merckstr. 25, D-64283 Darmstadt, Germany
{Darlagiannis, Heckmann, Liebau, Steinmetz}@KOM.tu-darmstadt.de

[‡] *Lancaster University*
Computing Department
Lancaster, LA1 4YR, UK
andreas@comp.lancs.ac.uk

Abstract

Despite the fact that entities in Peer-to-Peer systems are assumed to have equal roles, as their name (peers) implies, in reality such systems are composed of heterogeneous populations. Therefore, their components cannot provide identical services at the same quality. In order to overcome this issue, a role based approach is suggested where the core roles have been identified by analyzing the mandatory functionality of the overlay network. The roles are employed in the design of Omicron, a two-tier overlay network architecture.

In this paper, the identified roles, their functionality, the related protocols as well as the utilized communication mechanisms are described. The general applicability of the approach is demonstrated with the implementation of a simulator framework where well-known systems have been developed. Related measurements are provided showing the superiority of the described techniques.

1 Introduction

Peer-to-Peer (P2P) networks are composed of peers with different physical capabilities and user behavior. Such networks can be composed of a highly heterogeneous peer population, ranging from small PDAs to powerful end-systems. Therefore, specialization according to individual's abilities is an advantageous policy to effectively deal with the intrinsic peer heterogeneity. Omicron [6] is a P2P overlay network that has been accordingly designed. In Omicron, peers are assigned with roles having particular criteria and requirements to make such a decision. The roles identified by Omicron are based on the core operations required for the basic functionality of the overlay network.

The common core functionality provided by the majority of structured overlay networks could be described by the following basic operations: (i) The *Routing* operation that requires the construction of a routing table for selecting the most "promising" neighbor to forward the queries. (ii) The *Indexing* operation that constructs and updates the necessary distributed data structures for replying to queries. (iii) The *Maintenance* operation that maintains the ideal network topology so that the theoretical upper bounds for the communication cost in worst case scenarios can be met. Moreover, Omicron suggests an additional function, that of *Caching* to offer more efficient services, though it is proposed as an optional functionality for systems dealing with non-uniformly popular resources.

Most of the identified roles need to communicate with other roles located either in remote peers or locally. Each individual role has different communication needs resulting in multiple different protocols composed by several messages. However, it is worthwhile to investigate the possible similarities that may be found in (by definition) different messages. Moreover, there is a need to enable communication between asymmetrically composed peers (assigned with different roles). Vital information should be delivered to each peer independently of the assigned roles.

Further, the deeper investigation of the identified roles can provide better understanding of the system operations. Studying the required functionality of each role, specific algorithms and mechanisms can be utilized to fulfill the requirements. Finally, the investigation of basic scenarios involving multiple peer roles is necessary in order to make the inter-role relationships and the way they co-operate to provide the targeted operations and services more comprehensive.

It should be noted that the goal of this paper is not to provide a validated set of protocols that can handle every potential use case including scenarios where peers behave maliciously. Rather, the paper aims to describe the necessary functionality in order to demonstrate the operation of an overlay network based on different roles as they are assigned to peers. The investigation of the necessary protocols, the interaction of the roles in different typical scenarios and the evaluation of the involved mechanisms is the main focus of this paper.

The rest of this paper is organized as follows. Section 2 provides an overview of Omicron and its architecture. Section 3 describes the core components of the architecture and Section 4 presents the common functionality required by the protocols of each role. Afterwards, in Section 5 the functionality expected from the core roles is described, followed by several common scenarios in Section 6. Section 7 presents some simulation results as obtained by the role-based architecture and the paper is concluded in Section 8.

2 Omicron Overview and Architecture

2.1 Overview

Omicron *(Organized Maintenance, Indexing, Caching and Routing for Overlay Networks)* is a P2P overlay network aiming to address issues of heterogeneous, large-scale and dynamic P2P environments. Its hybrid, DHT-based approach makes it highly adaptable to a large range of applications. Omicron deals with a number of conflicting requirements, such as scalability, efficiency, robustness, heterogeneity and load balance. Issues to consider in this context are:

Topology. The rational in Omicron's approach is to reduce the high maintenance cost by having a small and fixed node degree, thus, requiring small and fixed size routing tables (at least for the majority of peers), while still performing lookup operations at low costs. For this reason the usage of appropriate graph structures (such as de Bruijn graphs [7]) is suggested. However, while the small fixed node degree reduces the operational cost, it causes robustness problems.

Clustering mechanism. To address the robustness issue, clusters of peers are formed with certain requirements on their endurance. *Cluster endurance* $E_C(t)$ of cluster C is defined as the probability that at least one peer of the cluster will survive until time t. Endurable clusters increase the stability of the network despite the (un)reliability of the peers as they unexpectedly depart from the system. Clusters can be considered as an equivalent mechanism to the suspensions used in vehicles to absorb shocks from the terrain. In order to maintain cluster endurance, new peer join requests are directed to the least endurable clusters. When the endurance of a cluster gets below a certain threshold, a merging operation takes place between the critical cluster and neighbor cluster(s).

Roles. A unique feature of Omicron is the integrated specialization mechanism that assigns particular roles to peers based on their physical capabilities and user behavior. The specialization mechanism provides the means to deal with peer heterogeneity. This scheme fits the contribution of each node to its resource capabilities and aims at the maximization of the cluster efficiency by providing appropriate incentives to peers to take a certain role. As it can be observed from Omicron's name, four different core roles have been identified: *Maintainers (M), Indexers (I), Cachers (C) and Routers (R)*. Maintainers are responsible to maintain the overlay network topology, while Indexers handle the relevant indexing structures. Routers forward the queries towards their logical destination and Cachers reduce the overall routing workload by providing replies to popular queries. Roles are additively assigned, meaning that peers do not remove their older roles as they get new ones.

2.2 Two-tier Network Architecture

The suggested *two-tier* network architecture for Omicron is a major step towards accomplishing the fulfillment of the targeted requirements. It enables the effective usage of de Bruijn graphs by successfully addressing their shortcomings. In fact, the successful "marriage" of two different topology design techniques (in a combination of a *tightly structured macro level* and a *loosely structured micro level*) provides a hybrid architecture with several advantages.

Tightly structured macro level. Adopting the topological characteristics of de Bruijn graphs, the macro level is *highly symmetrical* enabling *simple routing* mechanisms. Composed of endurable components, it results in a

relatively *stable* topology with *small diameter* and *fixed node degree*.

2. **Loosely structured micro level.** On the other hand, the micro level provides the desirable characteristics to the macro level by following a more loosely structured topology with a great degree of *freedom* in the neighbor selection. This freedom may be invested on regulating and achieving a *finer load balance*, offering an effective mechanism to handle potential *hot spots* in the network traffic. Moreover, *locality-aware* neighbor selection may be used to maximize the matching of the virtual overlay network to the underlying physical network. Finally, *redundancy* may be developed in this micro level supplying seamlessly *fault-tolerance* to the macro level.

An example of the hybrid topology is illustrated in Figure 1. The structured macro level is a de Bruijn(2, 3) digraph. Two nodes (representing peer clusters) are "magnified" to expose the micro level connectivity pattern between them. Two different connection types are shown: inter-cluster connections and intra-cluster connections.

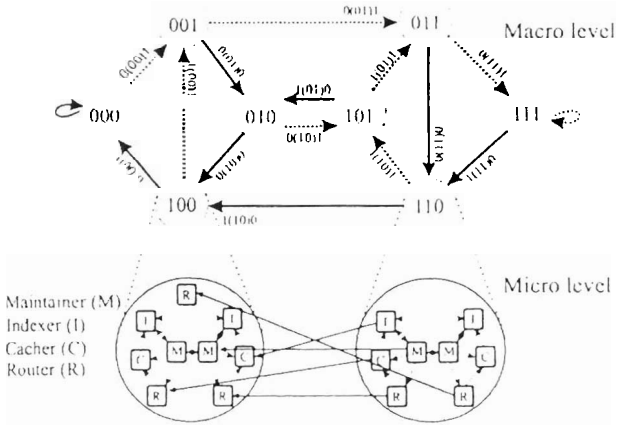


Figure 1: Omicron Overlay Network

3 Core Components

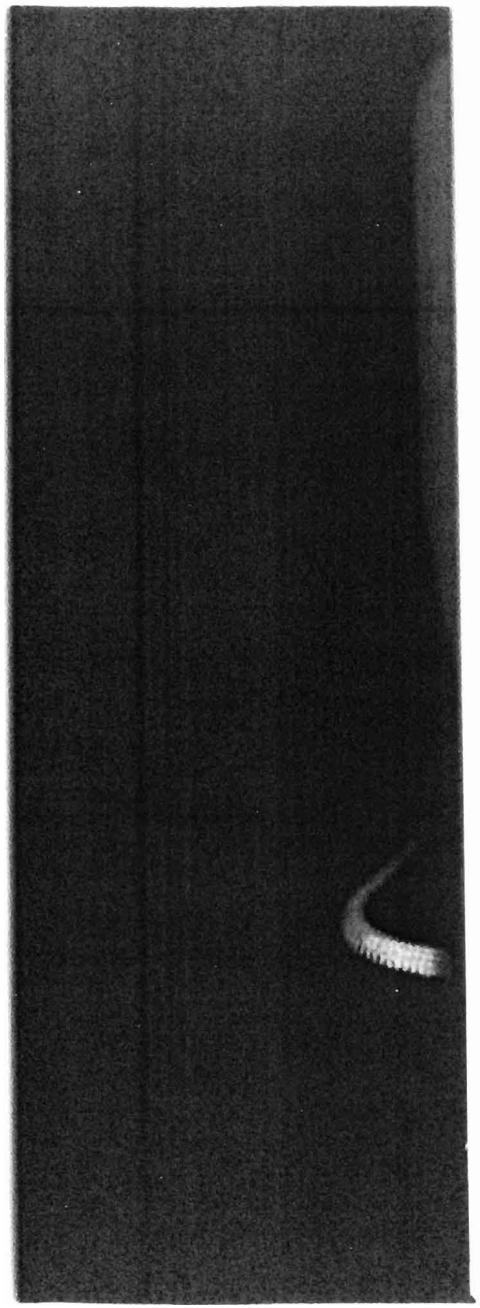
Before the presentation of the identified core roles, it is necessary to describe the basic components that are required for their realization. The basic components are divided in two types: (i) *common components* used in every DHT realization and (ii) *supplementary components* that are additionally introduced by Omicron.

3.1 Common Components

1. **Identification scheme.** One of the first tasks in designing an overlay network is the selection of an appropriate *identification scheme* that may uniquely characterize the involved entities. Peers are the most crucial entities that require unique identification. However, potentially more components might need identification, such as clusters, shared resources, services, etc. Usually the identifiers are randomly chosen from a large key space (address space). This selection may have a great impact on other design requirements, e.g., on the efficient routing mechanism, the evenly distributed workload, the accomplished security level or the supplied anonymity features. Advanced mechanisms might be required for the identification scheme to support the notion of strong or weak identities or even anonymity in a totaly decentralized way. Nevertheless, *hash-based* techniques are the most usual methods for generating global identifiers with a very high probability of uniqueness in a distributed way (without the need for a central authority).

2. **Routing tables.** The *routing tables* are vital components of the routing scheme. They should be efficiently structured and able to effectively represent peer's local view of the complete graph topology. Also, they should be sufficiently powerful to augment the efficient routing of every message independently of the destination despite the constraints of the limited local view. Constant size routing tables are preferable since their maintenance cost is not getting increased with the size of the system.

3. **Indexing structures.** In some cases the mistaken view can be encountered that structured P2P systems are designed to copy the advertised content and services at their system-specific location. Nevertheless, structured P2P systems are merely designed to hold indexing information about the actual



location of the advertised items. The structure of the employed indexing scheme is very crucial for the performance of the P2P system. Indices have to be designed in a way that allows for their re-distribution and re-organization at low cost. In fact, peers may join and leave the system dynamically so there may be frequent re-arrangements of the indexed and indexing content. For example, Chord is utilizing the *consistent hashing* mechanism [11] to accomplish this goal at low cost.

3.2 Supplementary Components

4. **ClusterMap.** *ClusterMap* is a newly introduced concept employed to summarize and describe the peers participating in a cluster[1]. ClusterMaps may be realized as tables collecting entries for each member peer. Every entry may hold information about other peers' GUID, its role in the system, the observed reliability and other useful information that could be used by each peer to construct its local routing table. In fact, ClusterMaps are supersets of Routing Tables, including the potential peers of clusters that may become neighbors of a particular peer. ClusterMaps are periodically disseminated to neighbor clusters as well as the cluster itself.

5. **Cache.** Caching is not usually found in the supplied functionality of existing tightly structured P2P systems. The main reason is that initially designers had assumed a set of requests with a uniformly distributed popularity. Though this might be a valid assumption for particular cases, the majority of empirical observations of widely deployed file sharing applications revealed the opposite. Therefore, these findings are considered in the design of our approach and an adaptive and effective caching mechanism is introduced. *Caches* have limited size and the optimal size value depends on the query popularity distribution and the observed peer behavior. It may be safely assumed that the low requirements of the caching mechanism are met by the great majority of the participating peers.

4 Protocol Basics

Several of the developed protocols for the particular aforementioned roles follow the hop-by-hop paradigm to deliver the messages to their destination. The developed communication mechanism uses a local message dispatcher on every peer. The message dispatcher examines several message fields in order to direct the incoming message to the appropriate instance that can handle it. Further details on the developed message dispatcher can be found in [4].

Inter-peer communication is implemented via a number of messages that constitute the necessary communication protocols. A number of common fields appear in each defined message. Table 1 provides the structure of an abstract message that defines the common fields. Every concrete message is derived from this abstract message structure.

Table 1. Common message structure

Field	Description
Name	Name of the message.
Scope	Defines the related overlay network.
Type	Relates the message to a specific role.
Style	Defines the way the message is forwarded towards its destination.
Initiator	Encapsulates the GUID of the peer that created the message.
Sender	Encapsulates the GUID of the peer that forwarded the message.
Destination	Encapsulates the GUID of the peer that should receive the message.
TTL	Time-to-live (optional).
Hops	Number of visited peers (optional).
Visited	List of traversed peers (optional).

The *name* field uniquely identifies each individual message (e.g. `UpdateClusterMap`, `LookupRequest`, etc.). The *scope* of the message defines the related overlay network. For example, an inter-cluster message has a different scope compared to an intra-cluster message. The *type* of the message specifies the role that should handle the incoming message, e.g. a Router, a Maintainer, etc. The *style* of the message defines the forwarding communication style, i.e., recursive or iterative.

The *initiator* field encapsulates the GUID of the message originator. Supple-

[1] Thus, this concept is required only in systems employing peer clustering mechanisms.

mentary information is provided in the *sender* field that encapsulates the GUID of the most recent peer that forwarded the specific message. The combination of these two fields provides the essential information to enable efficient communication schemes for query-reply based protocols. For example, a reply can be directly provided to the originator of any query without requiring to reversely traverse the same path that the query passed through. However, it should be noted that in particular cases the semantics of routing can change, e.g. involving the caching mechanism described in [3]. Thus, intermediate nodes may be visited in order to deliver the reply, too. The *destination* is set to the next closest peer in every hop based on the local routing table. Then, the message is forwarded to this peer using the underlying network.

Finally, some optional fields can provide additional useful functionality. For example, setting the *Time-to-Live* (TTL) to a maximum value can decrease the generated traffic (particularly useful when broadcasting mechanisms are utilized). However, it should be noted that the scope of the message is limited when the TTL field is actively used, which is inappropriate for e.g. `LookupRequest` messages. The following two fields (*hops* and *visited*) provide the means to construct non-oblivious messages. Thereby, it is possible to avoid cycles in the routing procedure, which is proven to be especially useful for random walk based communication mechanisms (that require single visit of peers). Appendix A provides detailed description of the developed messages to fulfill the requirements of the identified roles.

4.1 Inter-role Communication Model

Roles assigned to an individual peer can communicate only with similar roles in remote peers in order to keep their design as simple as possible. Moreover, since peers have different capabilities, different roles may be assigned to them. However, frequently there is a requirement that a role assigned to a peer must deliver some information to a remote peer that has not been assigned with the same role. For example, a Maintainer should provide an updated ClusterMap to a joining peer that has been assigned only with the Router role. In this case an additional mechanism is required to augment information delivery. A simple way to implement this mechanism is to assign an additional basic role to every peer, which provides the necessary message exchange, without altering the semantics of the identified roles. Such a role can fill the gap in the inter-role communication procedure.

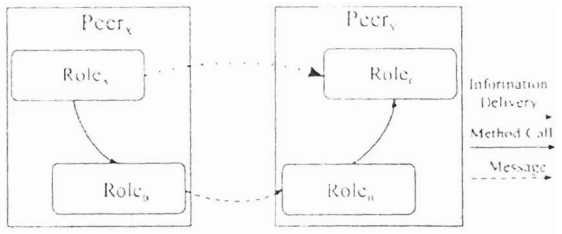


Figure 2: Inter-role communication mechanism.

The concept behind this mechanism is graphically illustrated in Figure 2. Assume that $Role_A$ assigned to $Peer_X$ must deliver some information $I_{A \rightarrow C}$ to $Role_C$ assigned to $Peer_Y$. Moreover, $Role_B$ has been assigned both to $Peer_X$ and $Peer_Y$. Assuming that $Role_B$ has been designed in a way that it can deliver the specific information $I_{A \rightarrow C}$ between different peers, an indirect communication takes place. $Role_A$ calls $Role_B$ of $Peer_X$, which delivers the information to its equivalent role in $Peer_Y$. Then, $Role_B$ calls a local method of $Role_C$ in order to pass the information to its destination.

4.2 Connector

From the previous discussion it becomes clear that a common role needs to be assigned by default to each peer. Despite the fact that the Router role is assigned to each peer, it is semantically inappropriate to be selected for this purpose. The aim of the common default role is to increase the inter-operability between peers assigned with different roles.

An important feature of this role is to offer the means to dynamically update the roles "installed" on each peer. Moreover, in terms of the adopted cluster-based approach, there is a need to provide a mechanism to dynamically update the local ClusterMaps of each peer as these structures evolve over time. In addition, a sufficiently rich mechanism is required to establish inter-peer connections. Two peers are considered as neighbors only if they have a mutually agreed established connection (which differs from the concept of TCP connec-

tions). Connections can be considered either as unidirectional or bidirectional based on the needs of the overlay network. Finally, peers should be able to periodically check the validity of the established connections. This can be implemented, i.e., using `Ping`/`Pong` messages[2]. The following list summarizes the required functionality.

1. Dynamic establishment and tear-down of inter-peer connections
2. Periodic checking of the validity of the established connections
3. Dynamical installation of new roles on peers
4. Dynamical updating of ClusterMaps on each peer.

5 Core Roles

In this section, the four core roles are described in more detail. The related protocol messages are provided in Appendix A. The particular roles have been identified after analyzing the common functionality that needs to be offered by most of the general purpose P2P overlay networks. Additional roles can be defined to provide advanced functionality and services (e.g. sreaming content or storing application-specific information [5]).

5.1 Router

Efficient routing is the most crucial functionality expected to be provided by a network system, which is the main aim of DHT-based structured overlay networks. The importance of efficient routing is investigated in [16] and [8], where the authors raise questions on the optimal routing that can be achieved. In terms of de Bruijn graphs, which appear to be one of the most promising network topologies for P2P systems, the complexity of the routing procedure increases logarithmically with the size of the system. This is formally denoted as $O(log(N))$, where N is the number of nodes, holds when the node degree is fixed. For cases where the node degree also increases logarithmically with respect to the network size the resulting routing complexity is discussed in [10]

$$D = O(\frac{log(N)}{log(log(N))}) \quad (1)$$

where D is the diameter of the de Bruijn digraph.

The optimal routing procedure in de Bruijn graphs has been investigated by other researchers as well. In particular, de Bruijn graphs have been studied in [15], aiming to combine short routing paths with fault-tolerant routing. Also, routing schemes for de Bruijn networks mostly applicable for parallel systems have been considered in [9], aiming to evaluate oblivious and non-oblivious mechanisms. Undirected de Bruijn graphs have been investigated in [13], focusing on the computational complexity of the routing algorithms. A pattern matching based approach for optimal routing is provided in [12]. Finally, Sivarajan and Ramaswani provide a shortest path solution for de Bruijn digraphs [18]. The simplicity of this solution makes it an interesting approach. In fact, this solution has been adopted and extended to fit the needs of our work.

5.2 Maintainer

The Maintainer is the most crucial role of the Omicron architecture and it is the role with the highest reliability requirements. Peers assigned with the Maintainer role constitute the backbone of the architecture. Their basic objectives are to maintain the targeted de Bruijn structure between neighbor clusters and to orchestrate the balanced organization of their cluster itself.

Depending on the observed peer lifetime, multiple Maintainers may be necessary to ensure continuous maintenance of the topology. Maintainers are responsible for providing updated ClusterMaps to the peers of the cluster and to the Maintainers of the neighbor clusters. Requests for participation by new members are also handled by Maintainers. As it has been described in [2], peers perform random walks to collect samples on the endurance of the clusters and direct new peers to the clusters that have the greatest need for them, resulting in a balanced network infrastructure.

Moreover, when the cluster population and subsequently the maintenance cost increase considerably, the cluster is divided in d new clusters (where d is the degree of the de Bruijn nodes), as long as the endurance requirements are met. Similarly, Maintainers trigger the cluster merging operation when their population is crucially low. The merging operation may be avoided if it is possi-

[2]The `Ping` and `Pong` messages form a mechanism to actively check whether the remote party is still available. Upon the reception of a `Ping` message, peers must promptly reply with a `Pong` message.

ble to migrate peers from neighbor clusters (without decreasing their endurance to a value less than a predefined threshold).

5.3 Indexer

Indexers are important roles that maintain efficient structures of the advertised resources and services. Their basic purposes are (i) to reply to incoming queries based on the indexed information, (ii) to store new resource and service advertisements and (iii) to synchronize the content of the local indexing structures with the content maintained by neighbor Indexers of the same cluster

The indexing mechanism can be as simple as a hash table where GUIDs are used as keys and owner peer addresses as values However, more advanced systems may need more complex structures such as rich metadata to provide advanced functionality, e.g., for range queries The investigation of such mechanisms is out of scope for this paper.

Moreover, depending on the type of the stored information, consistency can become a crucial requirement. In scenarios where a small number of Indexers suffices to provide the content, traditional consistency mechanisms can be efficiently applied [20].

5.4 Cacher

In the common design approaches of structured overlay networks, e.g. based on Distributed Hash Tables (DHTs) [1], queries are forwarded via intermediate peers towards the destination peer. This peer is responsible for the part of the DHT, which includes the globally unique identifier (GUID) characterizing the query. It is only the destination peer(s) that has the required information to reply to the query. Such design is suitable for evenly popular items since there is a non-ambiguous mapping of the resources to the system and the workload is evenly distributed. Thus, in such designs it is necessary to follow the whole path before it is possible to match the query.

The common core functionality provided by the majority of structured overlay networks is the one provided by the aforementioned roles (i.e routing, indexing and overlay network maintenance). Chord [19], Pastry [17] and Tapestry [21] are examples of structured networks that offer the aforementioned functionality However, Omicron [6] suggests an additional function, that of *caching* to offer more efficient services, though it is proposed as an optional functionality for systems dealing with non-uniformly popular resources [3]

The rationale behind the caching mechanism is described as follows. Since peers participate both in generating queries and routing them towards the destination, it may be advantageous to reuse the information gained from the replies they receive from locally generated queries. Thus, peers may directly provide the position of the requested resource instead of forwarding the query until it reaches the final DHT destination. Moreover, if peers monitor the popularity of forwarded requests, they could additionally consider caching the most popular of them, provided that they hold the necessary indexing information. A simple mechanism to develop such indexing knowledge is to modify the semantics of the routing procedure. For popular requests, intermediate peers may consider to store locally the incoming queries and generate identical ones (though originated at the intermediate peer) and forward them instead of the original queries. The received replies can be used both to reply to the stored pending queries and to populate the local cache with useful and popular information. However, the gathered information may be used for a maximum amount of time t_{Th} that depends on the peer uptime distribution.

A basic requirement for the caching mechanism is that it should not increase the signaling traffic generated by the peers. Therefore, Cachers differ from the other roles since there are no requirements for inter-Cacher communication. If someone the aforementioned requirement is released, proactive mechanisms can be explored that will populate the cached information at regular periods.

6 Inter-role Scenarios

In this section, some basic scenarios involving multiple peer roles are described in order to make the inter-role relationships and the way they co-operate to provide the targeted operations and services more comprehensive. In particular, three scenarios describing the necessary steps to complete a lookup request, a resource advertisement request and peer join request are discussed.

6.1 Resource Lookup Request

In this scenario, the Router role is the main entity orchestrating the required steps. Figure 3 shows the involved actions[1] The first step is triggered by an

[1] In the illustrated scenarios, cascaded oval shapes are used to indicate similar roles belonging to different (neighbor) peers Entities can communicate either by using message exchanging or by local method calls

incoming `LookupRequest` message *(step 1)*. The local message dispatcher directs this message to the Router, which checks whether the local cluster is responsible *(step 2)*. The cluster identifier and the requested key included in the `LookupRequest` message are compared to make this decision. If the comparison result is positive then the request may be forwarded to a neighbor peer in the same cluster that has been assigned with the Indexer role *(step 2A)*, unless the Indexer role is assigned to the local peer, too.

Afterwards, there are three options on how to proceed. The first alternative is followed if the Indexer role has been indeed assigned to the local peer. Therefore, the Indexer role is called and asked to checked whether the requested object has been published and is currently available *(step 3A)*. Then, the reply is provided back to the Router *(step 4A)*. The second option is followed if the Cacher role has been assigned to the local peer. Thereby, the Router checks whether the requested object is cached in the local cache *(step 3B)*. Then, the reply is provided back to the Router *(step 4B)*. The third option is followed either if the Cacher role is not assigned to the local peer or if the requested object is not in the local cache. In that case, the request is forwarded to the next peer that is closer to the targeted cluster *(step 5A)*. If the requested item is included either in the local index or cache, the requestor peer is contacted and supplied with the requested information *(step 5B)*, though.

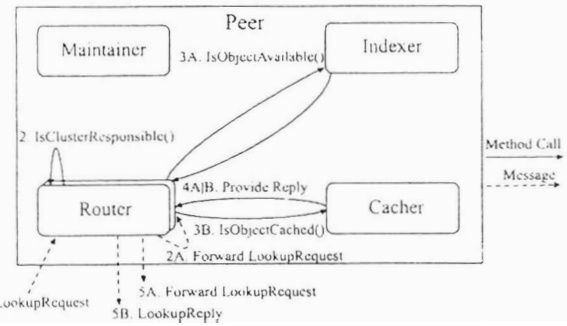


Figure 3: Lookup scenario steps.

6.2 Peer Join Request

The third scenario is intrinsically different from the previous scenarios. It is triggered by the reception of a `JoinRequest` message *(step 1)*. Such a message can be handled only by peers assigned with the Maintainer role. Thus, this request is always redirected by other peers to Maintainers of the same cluster. The scenario is graphically shown in Figure 4.

The Maintainer initiates a random walk randomly traversing selected neighbor Maintainers that are members of neighbor clusters *(step 2)*. After the appropriate cluster for accepting the new peer has been selected, the Maintainer must inform the joining peer. However, roles installed at a peer can communicate only with similar roles in remote peers. Since the new peer may not have been assigned with the Maintainer role, it is not possible for the Maintainer to accept the new peer. Thus, the *Connector* role has been introduced to fill this design gap. The Connector can provide three important pieces of functionality: (i) *connectivity functionality*, (ii) *new role installation functionality* and (iii) *ClusterMap updating functionality*.

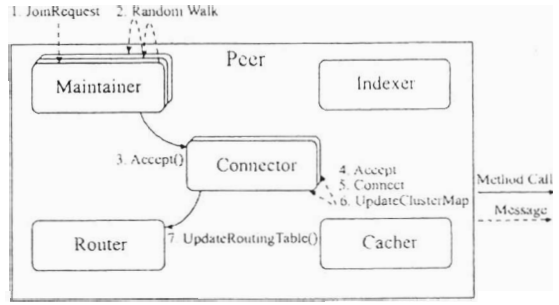


Figure 4: Peer join scenario steps.

Therefore, the Maintainer calls the Connector to accept the new peer *(step 3)*. Subsequently, the local Connector contacts the remote Connector and provides

the acceptance decision *(step 4)*. Then, the new peer requests from the Maintainer to be considered as a member of the same cluster *(step 5)*. Afterwards, the Maintainer provides the local and the neighbor ClusterMaps *(step 6)* to the new peer, which can populate its routing table with valid addresses *(step 7)*

6.3 Resource Advertisement Request

The second scenario, which is graphically shown in Figure 5 involves the Router and the Indexer. This scenario is triggered by the reception of a `PublishRequest` message *(step 1)*. Similar to the previous scenario, the message is directed to the local Router, which checks whether the local cluster is responsible *(step 2)*. The cluster identifier and the generated key for the advertised object included in the `PublishRequest` message are compared to make this decision. If the comparison result is positive then the request may be forwarded to a neighbor peer in the same cluster that has been assigned with the Indexer role *(step 2A)*, unless the Indexer role is assigned to the local peer as well. Then, the Indexer role is called and the local index is updated to include the newly advertised object *(step 3)*. The local Indexer sends `UpdateIndexRequest` messages to neighbor Indexers belonging to the same cluster *(step 4A)*. Based on the applied consistency policy, an additional action may periodically take place at this stage. Nevertheless, the Indexer confirms to the Router that the newly published object has successfully been advertised *(step 4)*, which subsequently provides the confirmation back to the originator of the `PublishRequest` message *(step 5B)*. Alternatively, if the local cluster is not the appropriate place to advertise the new object, then steps 3 and 4 are skipped and the Router directly forwards the request to the next peer that is closer to the targeted cluster *(step 5A)*

7 Results

In order to assess more accurately the performance of the role-based overlay network, a role-based simulator (described in [4]) has has been developed. Using this simulator, a number of experiments has been performed. Moreover, in order to demonstrate the general applicability of the role based approach, Chord [19] has also been been developed using the role based approach (where all of the required roles are assigned to every peer). The size of the evaluated

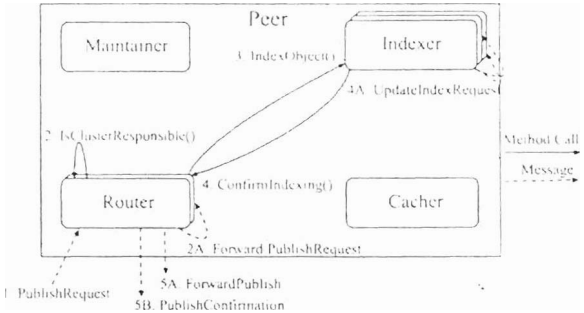


Figure 5: Resource advertisement scenario steps

P2P networks is ranging from 100 to 130,000 peers[4]. During the performed experiments the size of clusters ranges between 2 and 16 peers. The degree of the de Bruijn networks is mostly set to 2 and 4 connections per node. While it is possible to set it to any arbitrary integer, the interest of our research lies in fixed, low degree de Bruijn networks. Networks are constructed as peers join in the so-called bootstrap phase. When the targeted population is reached, peers perform operations and statistics are collected. Events are either randomly or periodically generated according to the needs of the particular experiment. Several random number generators are used to provide the appropriate input rates, e.g., uniform, normal, lognormal, Zipfian, etc.

7.1 Impact on Average Query Length

Scalability is the most critical requirement for most P2P applications. In terms of core overlay network operations, the cost of `LookupRequest(s)` is the metric we use to evaluate the scalability of the network. It should be noted that the cost both for `LookupRequest(s)` and `PublishRequest(s)` (as they are described in Appendix A) is identical. In the rest of this section we refer to

[4] It should be noted that not many widely available simulators have such a scale, to simulate network models of that scale and model detail

such messages as *queries*. The focus of the experiments is mostly on the way the communication cost increases as the network grows in size.

The average query length can be regarded as the most relevant metric to evaluate the scalability of a P2P overlay network. Moore's law defines lower bounds for k-regular graphs. In fact, de Bruijn graphs are asymptotically optimal graphs converging to this bound for sufficiently large graph order N and node degree k. As it is shown in [14]:

$$\mu_{DB} \approx D_{DB} - \frac{1}{k-1}, \tag{2}$$

where μ_{DB} is the average asymptotic distance in de Bruijn graphs and D_{DB} is the diameter.

The relevant collected measurements include experiments involving both the Omicron and the Chord networks. Initially, we consider only the scalability of the Omicron network with varying average cluster size between three values ($\overline{K}_1 = 4$, $\overline{K}_2 = 8$, $\overline{K}_3 = 16$). Figure 6 provides the graphical representation of the experiments. Both x-axis and y-axis are logarithmically scaled to provide a more comprehensive view. The average query length is constituted both of the inter-cluster routing (de Bruijn based) towards the targeted cluster and the intra-cluster step to reach an Indexer. In the particular experiments routing is performed by peers assigned only with the Router role. No Cachers are present in this experiment, therefore, Routers access directly an Indexer of the cluster to get the queried information. The close matching of the analytical approximation and the simulation results can be evidently observed from this figure. Moreover, it can be stated that, as the size of clusters grow exponentially, the average routing length decreases linearly. Therefore, as the cluster maintenance cost grows exponentially, it can be safely assumed that there is an optimal maximum value for the size of the clusters, beyond which the cost grows more quickly than the utility.

Beyond the absolute performance of Omicron, it is essential to compare it with a widely known system like Chord, which can be considered as a reference point. Figure 7 shows the related simulation results. It should be noted that both X and Y axes are logarithmically scaled to provide a more comprehensive view. As it can be observed from this figure, Chord outperforms the Omicron network configuration with average cluster size $\overline{K} = 8$ and node degree $k_1 = 2$. However, it should be noted that Chord's design requires a logarithmically increasing node degree. Though, by similarly increasing the node degree of Omicron, the average query length is getting significantly smaller than Chord.

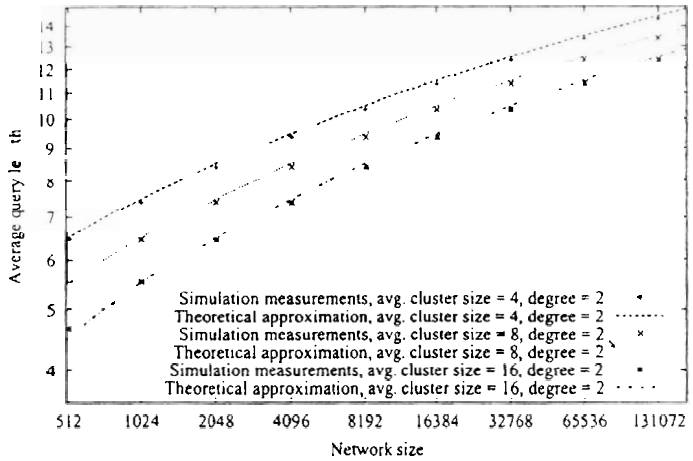


Figure 6: Routing scalability.

To analytically express this, if the diameter in Equation 2 is replaced by the formula of Equation 1 we get:

$$\mu_{DB} \quad \frac{log(N)}{log(log(N))} - \frac{1}{log(N)-1} \tag{3}$$

Chord's average length approximation is given by $\mu_{CH} = log(N)/2$. Therefore, for similar node degree, Omicron achieves smaller average query length. In fact, for the network order range described in Figure 7, by setting Omicron's node degree to $k_2 = 4$, Omicron outperforms Chord. Again here, it can be noted the good matching of the analytical approximation and the simulation results.

7.2 Impact of Caching on Routing

In this section the measurement observations of the simulative experiments are reported. Figure 8 displays the reduced routing communication cost in terms of required overlay traverse steps as the percentage of the communication cost

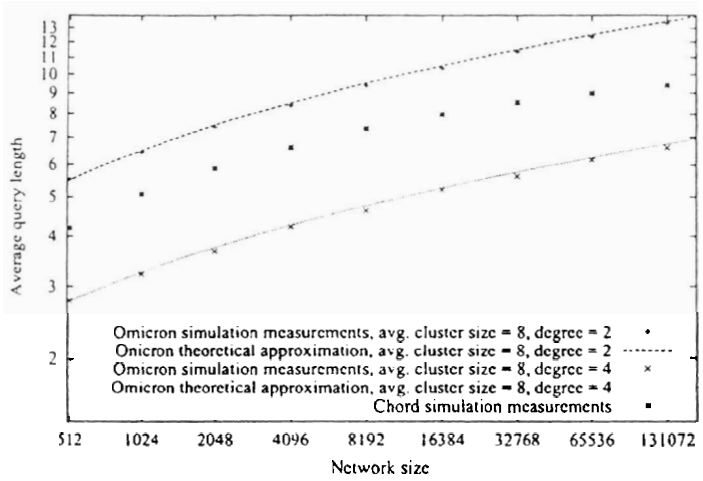


Figure 7: Routing scalability: Omicron versus Chord.

load.

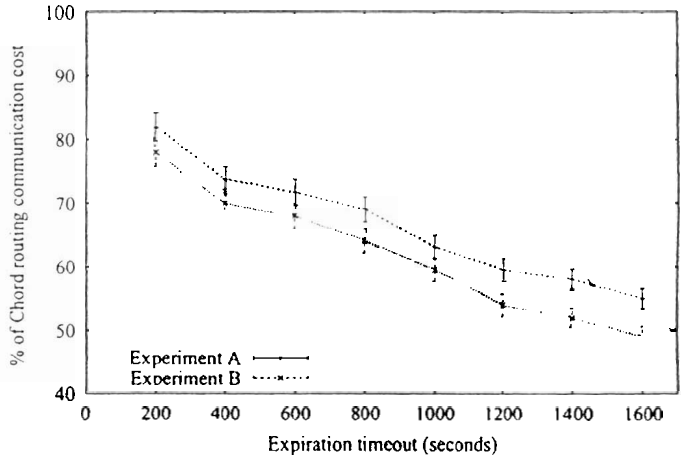


Figure 8: Cache routing load as a percentage of the original Chord network.

of the original Chord network, as a function of the *expiration timeout*. The expiration timeout is the time a cache entry is assumed to be valid with high probability, without the need to contact the original Indexer. The parameters of interest are the maximum cache size, the $FREQUENCY\_THRESHOLD$ which declares the number of times a query for a particular item should arrive in a peer so that it will be considered as popular and the reset time period of this counter. Two different experiments have been selected for demonstration:

1. Experiment A, where the $FREQUENCY\_THRESHOLD$ is 5, the maximum cache size is set to 80 and the frequency counter is reset every 200 seconds.

2. Experiment B, where the $FREQUENCY\_THRESHOLD$ is 3, the maximum cache size is set to 300 and the frequency counter is reset every 100 seconds.

We can observe that the total communication load for query routing can be considerably reduced using the caching mechanism down to 50% of the original

8 Conclusions

The aim of this paper is to provide a pragmatic approach to deal with the intrinsic heterogeneity of the participants in P2P systems. The followed approach is based on roles, which are identified based on the core functionality and operations required by the majority of the P2P overlay networks. The paper presents the required functionality and the related protocols of a role-based P2P overlay network architecture. In particular, small communities of peers are created where roles are appropriately assigned to peers. Then, these communities compose an efficient DHT in order to provide fast lookup operations. The structure of the DHT adopts the de Bruijn topology to minimize the maintenance overhead, while still provide a scalable solution to route the queries.

The role based approach is evaluated with simulations. The simulator has been developed by adopting the role based approach. Further, this technique is used to implement well-known overlay networks, such as Chord, in order to demonstrate its general applicability. The collected measurements show the advanced characteristics of Omicron in terms of each ability to efficiently handle

large-scale, heterogeneous and dynamic environments. Moreover, the extended design of Chord that adopts the caching mechanism of Omicron can provide much more efficient routing for scenarios where the query distribution is approximated by a lognormal distribution.

A Appendix: Protocol Messages

A.1 Connector Messages

Several messages are needed to provide the aforementioned functionality. A new peer may initiate its participation in a P2P system by submitting a `JoinRequest` message, which may be handled only by peers assigned with the Maintainer role. The first reply a new peer receives after the request to join is the `Accept` message. The `Accept` message includes a list of *PeerAddresses* that are candidate neighbors for the newly joined peer. The list may include only one peer, e.g., consider the case of Chord where only the ring successor suffices to establish the initial connection and then dynamically updates the fingers. Alternatively, it may include a list of peers, e.g., consider the case of Omicron where all the available Maintainers of the cluster may be provided to select one for the initial connection. The structure of the `Accept` message is given in Table 2. It extends the structure of the basic message provided in Table 1. In the rest of this chapter, each defined message is derived from the common message structure.

Table 2: `Accept` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
PeerAddresses	Provides a list of valid peers that can become direct neighbors.

In order to install new roles at peers, the `UpdateRoles` message is used. It includes a list of the new roles to be installed and it may be initiated only by peers that are assigned with the Maintainer role.

After receiving the `Accept` message a peer requires a connection from an active Maintainer. At this initial connection phase as well as at further points in time (periodically defined or event triggered) peers receive `UpdateClusterMap` messages that include updated ClusterMaps. Their information is essential for the correct operation of each peer, which updates its local routing tables and the established connections with currently active peers. An `UpdateClusterMap` message may be sent only by peers assigned with the Maintainer role. The structure of an `UpdateClusterMap` message is provided in Table 4

Table 3: `UpdateRoles` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1)
Roles	Provides a list of additional roles that can be installed on the peer.

Table 4: `UpdateClusterMap` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
ClusterMap	Provides an updated ClusterMap that can be used to update the local routing table and the established connections.

In addition, four more messages have been defined for the Connector role. The `Connect` and `Disconnect` messages are used to establish and teardown connections, respectively. The `Ping` and `Pong` messages are used to check the validity of the already established connections. Their structure is simple including the common message fields and optionally the local time a peer initiates the `Ping` message, which is copied in the `Pong` message as well.

A.2 Router Messages

This section provides a description of the messages that constitute the routing protocol. Table 5 describes the structure of the `LookupRequest` message. It includes a *key* representing the GUID of the queried item. `LookupRequest`

messages are forwarded towards the cluster whose GUID matches best with the included key.

Table 5: `LookupRequest` message structure

Field	Description
Common	Derived common message structure (cf. Table 1).
Key	Includes the GUID describing the queried item.

Table 6 describes the structure of the `LookupReply` message. It includes a *key* representing GUID of the queried item and the related indexing information (*IndexValue*). It is common technique to set the indexing information to merely the address of the owner of the related service or resource. However, the indexing information may include a long list of addresses for very popular items. Efficient mechanisms can be introduced in deployed systems to handle such scenarios. `LookupReply` messages are delivered directly to the originators of the queries.

Table 6: `LookupReply` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
Key	Includes the GUID describing the queried item.
IndexValue	Includes the GUID of the indexing information that matches the key.

Table 7 describes the structure of the `PublishRequest` message. It includes a *key* representing the GUID of the queried item and the *OwnerAddress* of the advertised service or resource that matches the provided key. The `PublishRequest` message is delivered to the cluster that matches best with the included key and subsequently, the Indexers update their local structures with the newly advertised item. Typically, advertisements have an expiration timeout. It is the responsibility of the owner to refresh the published information.

After successfully advertising an item, the responsible Indexer provides a confirmation to the originator of the advertisement using a

Table 7: `PublishRequest` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
Key	Includes the GUID describing the published item.
OwnerAddress	Includes address of the owner of the advertised service or resource that matches the provided key.

`PublishConfirmation` message. Its structure is described in Table 8

Table 8: `PublishConfirmation` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
Key	Includes the GUID describing the published item.
Confirmation	Confirms the successful advertisement of the published resource.

A.3 Maintainer Messages

This section provides a description of the messages that constitute the routing protocol. Table 9 describes the structure of the `SplitClusterRequest` message. This message is exchanged among the Maintainers of the same cluster to initiate the division of the cluster (as long as the endurance requirements are met) into d new ones, where d is the cluster degree (recall that cluster represent nodes in the constructed de Bruijn graph). The `SplitClusterRequest` message includes a field with a list of the proposed new *ClusterMaps*.

Maintainers that receive a `SplitClusterRequest` message check the validity of the suggested division and confirm it using a `SplitClusterConfirmation` message. The structure of this message is described in Table 10.

In certain scenarios it is necessary to merge existing clusters that do not fulfill the endurance requirements. In this case a Maintainer of the critical cluster sends a merging request to a neighbor cluster using a

Table 9: `SplitClusterRequest` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
ClusterMaps	Provides a list of suggested ClusterMaps by dividing the existing cluster members.

Table 10: `SplitClusterConfirmation` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
Confirmation	Confirms the suggested cluster division.

`MergeClusterRequest` message. The structure of this message is provided in Table 11.

Table 11: `MergeClusterRequest` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
ClusterMap	Provides the ClusterMap to be merged.

Maintainers that receive a `MergeClusterRequest` message process the included ClusterMap. If the particular cluster represents the best candidate among the neighbors it accepts the merging request and it confirms it using a `MergeClusterConfirmation` message. The structure of the confirmation message is provided in Table 12.

A.4 Indexer Messages

This section provides a description of the messages that constitute the routing protocol. Periodically or after a certain number of indexing structure updates, Indexers synchronize their locally stored indexing information to achieve a certain level of consistency. `UpdateIndexRequest` messages are used for this

Table 12: `MergeClusterConfirmation` message structure

Field	Description
Common	Derived common message structure (cf. Table 1).
Confirmation	Confirms the suggested cluster merging operation.

purpose. The structure of an `UpdateIndexRequest` message is given in Table 13. The included *index* can be the complete indexing structure. Alternatively, peers may select to exchange only recent differences in order to reduce the generated traffic load.

Table 13: `UpdateIndexRequest` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
Index	Provides an updated index of objects for which the relevant cluster is responsible.

Upon the reception of an `UpdateIndexRequest` message, Indexers check the validity of the information. If they accept the incoming information they provide a confirmation using an `UpdateIndexConfirmation` message. Otherwise, they reply using a valid `UpdateIndexRequest` message back to the originator of the received message. The structure of an `UpdateIndexConfirmation` message is provided in Table 14.

Table 14: `UpdateIndexConfirmation` message structure.

Field	Description
Common	Derived common message structure (cf. Table 1).
Confirmation	Confirms the valid update of the local Index.

Acknowledgements

The work on this paper is partly sponsored by the EU FP6 Network of Excellence E-Next (http://www.ist-e-next.net).

References

[1] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up Data in P2P Systems. *Communications of the ACM*, 46(2):43–48, 2003.

[2] Vasilios Darlagiannis. *Overlay Network Mechanisms for Peer-to-Peer Systems*. PhD thesis, Department of Computer Science, Technische Universität Darmstadt, Germany, June 2005.

[3] Vasilios Darlagiannis, Nicolas Liebau, Oliver Heckmann, Andreas Mauthe, and Ralf Steinmetz. Caching Indices for Efficient Lookup in Structured Overlay Networks. In *Proceedings of the Fourth International Workshop on Agents and Peer-to-Peer Computing*, July 2005.

[4] Vasilios Darlagiannis, Andreas Mauthe, Nicolas Liebau, and Ralf Steinmetz. An Adaptable, Role-based Simulator for P2P Networks. In *Proceedings of the International Conference on Modeling, Simulation and Visualization Methods*, pages 52–59, June 2004.

[5] Vasilios Darlagiannis, Andreas Mauthe, Nicolas Liebau, and Ralf Steinmetz. Distributed Maintenance of Mutable Information for Virtual Environments. In *Proceedings of the 3rd IEEE International Workshop on Haptic Audio Visual Environments and their Applications - HAVE'04*, pages 87–92, October 2004.

[6] Vasilios Darlagiannis, Andreas Mauthe, and Ralf Steinmetz. Overlay Design Mechanisms for Heterogeneous, Large Scale, Dynamic P2P Systems. *Journal of Networks and System Management*, 12(3):371–395, 2004.

[7] Nicolaas G. de Bruijn. A combinatorial problem. In *Proceedings of the Koninklije Nederlandse Academie van Wetenshapen*, pages 758–764, 1946.

[8] Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News - Special Interest Group on Automata and Computability Theory)*, 32:36–52, 2001.

[9] Frank Hsu and David Wei. Efficient Routing and Sorting Schemes for de Bruijn Networks. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1157–1170, 1997.

[10] Frans Kaashoek and David R. Karger. Koorde: A Simple Degree-optimal Hash Table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, February 2003.

[11] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM Press, 1997.

[12] Zhen Liu. Optimal routing in the de Bruijn networks. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 537–544, May 1990.

[13] Zhen Liu and Ting-Yi Sung. Routing and Transmitting Problems in de Bruijn Networks. *IEEE Transactions on Computers*, 45(9):1056–1062, 1996.

[14] Dmitri Loguinov, Anuj Kumar, Vivek Rai, and Sai Ganesh. Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience. In *Proceedings of ACM SIGCOMM'03*, pages 395–406, August 2003.

[15] Jyh-Wen Mao and Chang-Biau Yang. Shortest path routing and fault-tolerant routing on de Bruijn networks. *Networks*, 35(3):207–215, 2000.

[16] Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Routing algorithms for DHTs: Some open questions. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, 2002.

[17] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[18] Kumar Sivarajan and Rajiv Ramaswami. Lightwave networks based on de Bruijn graphs. *IEEE/ACM Transactions on Networking (TON)*, 2(1):70–79, 1994.

[19] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable Peer-to-Peer Lookup Service for Internet Applications. *IEEE Transactions on Networking*, 11(1):17–32, February 2003.

[20] Peter Triantafillou and Carl Neilson. Overlay Design Mechanisms for Heterogeneous, Large Scale, Dynamic P2P Systems. *IEEE Transaction in Software Engineering*, 23(1):35–55, January 1997.

[21] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.