Universität Hannover
Institut für mikroelektronische Schaltungen und Systeme
Prof. Dr.-Ing. E. Barke

Semantic Web Services

Diplomarbeit
von
Mario Schlosser

Dezember 2002

Universität Hannover
Institut für mikroelektronische Schaltungen und Systeme
Prof. Dr.-Ing. E. Barke

Semantic Web Services

Diplomarbeit
von
Mario Schlosser

Betreuer: Dr. Stefan Decker (Stanford University)

Erstprüfer: Prof. Dr. E. Barke
Zweitprüfer: Prof. Dr. W. Nejdl

Erklärung

über die Anfertigung der Diplomarbeit gemäß Diplomprüfungsordnung von 06.10.2001, § 27 (5)

Ich versichere, die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Hannover, den 1. Dezember 2002
Mario Schlosser

Contents

Chapter 1

Einleitung

Web Services sind Softwaremodule, die über ein Netzwerk angesprochen und aktiviert werden können. Semantic Web Services stellen ihre nächste Entwicklungsstufe dar: Semantic Web Services zeichnen sich durch eine formalisierte Beschreibung ihrer Fähigkeiten aus, die durch Rechneralgorithmen verarbeitet und verstanden werden können. Dies ermöglicht die Anwendung von automatischen Such- und Planungsalgorithmen auf Semantic Web Services. Algorithmen für Semantic Web Services sollen es auf diese Weise automatisiert ermöglichen, in einer großen Menge von Serviceangeboten gezielt ideal für die Bewältigung einer Aufgabe geeignete Services zu identifizieren, formalisierte Beschreibungen komplexer Aufgaben (zum Beispiel von Workflows) auf eine Aufrufsequenz einfacherer, vorhandener Web Services abzubilden, oder die Qualität von Web Services anhand ihres Verhaltens automatisiert zu bewerten.

Diese Arbeit beschreibt Algorithmen im Kontext von Semantic Web Services. Peer-to-Peer Netzwerke als Infrastruktur für ein Netz aus Service-Providern ermöglichen eine effiziente und verteilte Suche von Services, basierend auf einer formalisierten Beschreibung des gesuchten Services. Der im Rahmen der Arbeit entwickelte HyperCuP-Algorithmus organisiert Peers in einem P2P-Netzwerk in eine deterministische Netzwerktopologie, die sich sehr effizient durchsuchen läßt. Der Algorithmus ermöglicht dabei die Skalierung des Netzwerkes auf hohe Zahlen von teilnehmenden Peers. Gleichzeitig wird die Anzahl der Nachrichten, die zwischen Peers zum Aufbau der Topologie ausgetauscht werden muss, auf einem niedrigen Niveau gehalten.

Die Arbeit beschreibt weiterhin einen Algorithmus zur Quantifizierung der Vertrauenswürdigkeit von Peers in einem P2P-Netzwerk. Peers, die sich in einer für das Funktionieren des Netzwerkes schädlichen Weise verhalten, werden durch den Algorithmus identifiziert und ihr Einfluss auf das Netzwerk stark reduziert.

Zur Simulation der Algorithmen wird ein allgemeines Modell eines P2P-File-Sharing-Netzwerkes entwickelt und beschrieben.

Chapter 2

Introduction

Web Services are software modules which provide some kind of service and which can be accessed and invoked via a network.

Semantic Web Services are considered to be the next step in the evolution of Web Services. In addition to Web Services, Semantic Web Services feature formalized and machine-understandable descriptions of their capabilities which allow them to be understood and processed by automatic algorithms. Semantic Web Services are envisioned to enable sophisticated tasks such as automated discovery of services by matching specified service requests with a large pool of service providers, automated composition of services to instantiate high-level descriptions of complex tasks by a sequence of calls to simpler services, and service monitoring to evaluate the quality of work provided by services.

This thesis presents algorithmic building blocks for evolving existing Web Services standards into Semantic Web Services technology. It introduces Peer-to-Peer (P2P) Networking as a new paradigm for distributed service discovery. The HyperCuP algorithm and protocol presented in this work organizes providers of Web Services into a large-scale P2P network, a Web of Services. The network enables efficient discovery of Web Services matching formalized ontology-based service requests. The HyperCuP protocol ensures the scalability of the network to millions of service providers while reducing search and discovery times as well as reducing the administrative message overhead required to build and maintain the network.

The Web of Services is enhanced by a distributed algorithm for trust management in a P2P network. The algorithm presented is capable of identifying and banning malicious peers in a file-sharing P2P network from causing damage to the network by sharing inauthentic files. The algorithm can be used in a Web of Services to prevent malicious service providers from disrupting network operations by offering damaging services.

To simulate the algorithms, a general model of P2P networks is developed and described.

Chapter 3

Semantic Web Services

Web services are self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces. – Web services are loosely coupled, communicating directly with other Web services via the Internet using standards-based protocols. – Web Services are fundamental building blocks on the way to large-scale distributed computing. – Web Services are the next big thing.

Definitions for the term "Web Services" provided by IBM, Microsoft and Bill Gates. In fact, as of 2002, Web Services have received much attention, and several research and industry efforts have been spawned aiming at developing and deploying Web Services technology. This chapter will provide an overview of existing Web Services industry standards, formulate a long-term vision for Web Service research and introduce an algorithmic perspective for achieving long-term vision goals.

A Web Service is a software module that is accessible via the Web, providing some kind of service. A short-term vision for the evolution of Web Services is provided by industry efforts on the deployment of Web Service technology: Web Service standards aim at enabling systems and programs to communicate and interact via the Internet. A company may provide Web interfaces, say, for its ordering and billing systems, enabling other companies to invoke and use its services via the Internet.

The long-term vision for the evolution of Web Services, however, envisions the Internet to be a huge pool of peers in which every participant is able to offer some kind of service. In an automated fashion, descriptions of complex tasks are submitted to the Web of Services, potentially helpful services are identified on the fly and in a distributed manner, and afterwards composed to a complex service capable of finding a solution to the problem described. The query "Plan a vacation in Hawaii" organizes a vacation, based on the preferences and financial situation of the user, the situation of water currents and the volcanic activity at the destination,

uses peer reviews of potentially interesting hotels to evaluate and choose a hotel, and finally yields a flight, car and hotel booking and a detailed holiday schedule in order to optimize fun, educational outcome and relaxation at the same time. All by using a weather forecast Web Service, crawling hotel Web Services, invoking booking Web Services of airlines and many others.

3.1 A Web Services Scenario

As an example from a future world of Web Services, consider the following scenario. Company A wants to buy goods from company B. First, an internal automated service of company A realizes a shortage on a specific product and decides to place an order with company B. Company A's buying service automatically invokes company B's ordering service and places the order via the Internet. Additionally, company A's buying service contacts an insurance company's insurance service to purchase an insurance for the transaction. A fourth Web Service of a financial institution is accessed and invoked to execute the financial transactions between the companies. Finally, the service of a logistics company is used to ship the goods.

The scenario itself is by no means new or futuristic. However, in such a process today there is little degree of automation: Companies do use electronic exchanges of information and goods in e-business processes, but those are usually static, programmed once, hard to modify, and based on proprietary protocols. Despite already increasing the efficiency of business processes, such a model exhibits many drawbacks:

What if services of two companies are built on different platforms and protocols? Web Services are to be able to communicate via networks such as the Internet. Thus, their communication among each other will be built upon the usual networking protocols. Additionally, Web Services require a common, formalized way of accessing and exchanging information among each other. As when calling a function in a program and passing parameters to the function call, communicating with a Web Service requires some way of binding data passed to some meaning or interpretation. A protocol for *service communication* is needed.

What if a new player enters the game? In the example, company A relies on buying goods from other companies in the market. If the business process described takes place with a fixed set of players, company A will never be able to explore the market for cheaper prices or better service – at maximum it could do so off-line, and then reprogram its business process as soon as a competitor of company B offers its service at superior terms. However, such a manual and off-line process of looking for business

partners is tedious and prohibitively expensive in a dynamic environment with many new players popping up and vanishing in every single moment. Hence competitors of company B require a way of publishing and advertising their Web Services by virtue of *service publication*. Also, service advertisements have to be found by company A in some way, requiring means of *service discovery*.

What if a new player on the market offers an innovative insured shipping service? The power of Web Services will be their diversity: As with today's Internet, an open Web Services infrastructure could be the basis for a diverse range of services, offered by a diverse range of service providers. However, as with today's Internet, too, service consumers then face the problem of understanding what a particular Web Service actually does – and even more so, having algorithms and machines understand it: The problem of searching information on an internetwork has not been fully solved yet (Google's [22] mission statement aims at solving the "AI-complete" search problem in general, i.e., being able to answer any question automatically by crawling the Internet [27]), and it is just as complicated in the world of Web Services where a service is searched to work on a particular specified task. Hence it is vital that a commonly agreed upon way of *service description* be established. Otherwise the company computers in the given Web Services example may never be able to understand the upsides of an innovative new Web Service entering the market, just because its description slightly differs from previous, though semantically similar services.

What if a company's business process changes? Complex tasks require breaking down the task into smaller subtasks, individually executable. Web Services could provide a means of efficiently executing a complex task by having many Web Services interoperate and cooperate. If a company's business process changes, it may just need to update its complex, composed service in one little detail, by replacing a service building block by another, in a process of *service composition*.

What if a competitor deliberately offers a company a dysfunctional service? An entirely open Web of Services introduces issues of security: Web Services are not certified by any central authority to provide good service. A means of *service monitoring* is required to track down malicious service providers which provide bad quality or even dangerous services.

Figure 3.1 groups the Web Service activities identified above into an abstract protocol stack.

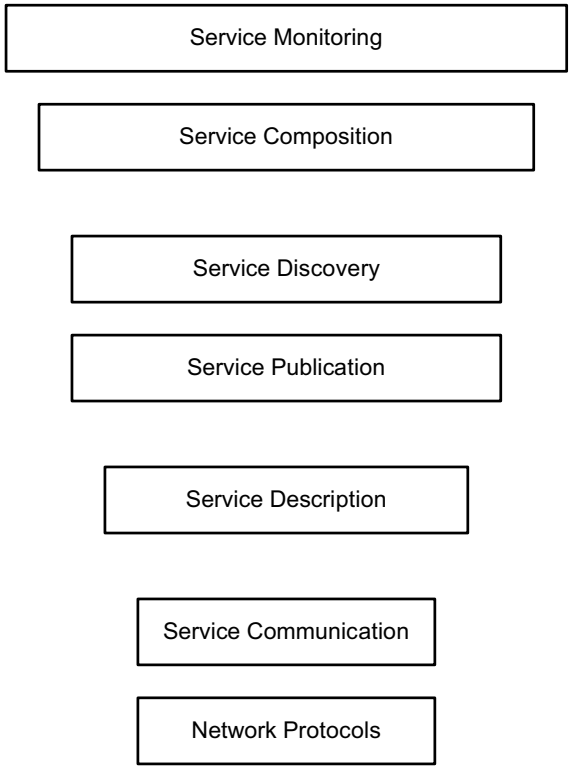

Figure 3.1: Web Services Protocol Stack

3.2 A Web Services Protocol Stack

Current industry efforts have started addressing the construction of a unified Web Services stack. Four efforts in this direction – SOAP, WSDL, WS-BPEL, UDDI – shall be described briefly in the following.

3.2.1 Web Services Communication – SOAP

SOAP (Simple Object Access Protocol [12]) is a simple XML-based messaging protocol which has been proposed to exchange messages among Web Services. A SOAP message is an XML (Extensible Markup Language, a formalized way of encoding information in a tree-like document structure [49]) document which follows a pre-defined format of encoding message content, specifying sender and recipient, content and control fields of the message. For example, if a Web Service intends to invoke the execution of another Web Service, it encodes the parameters of the call (such as the product and volume to be bought from a selling service) in a SOAP document and sends the message over the Internet. SOAP messages may be transmitted via HTTP.

3.2.2 Web Services Description – WSDL

The Web Service Description Language [14] is a language proposed by the World Wide Web Committee (W3C) to describe Web Services.

A service is described as a set of endpoints operating on messages. To this end, WSDL specifies an XML grammar which states how to describe messages and operations of a Web Service in a standardized format, using standardized data types.

A Web Service in the definition of WSDL consists of ports, each of which can be accessed by a number of specified operations. Operations in turn are invoked by sending messages whose exact format is also determined by the WSDL document. Hence a Web Service is fully characterized by messages it is able to process and generate. Additionally, WSDL documents may contain pointers to more detailed, natural language information on the service.

WSDL documents are mostly used manually: During the implementation of a Web Service which will access another Web Service upon its invocation, the WSDL document describing the Web Service to be accessed may be used to automatically generate and program the interface to the service. WSDL descriptions do not contain much more than basic technical information on a service. The description of messages processed by the service does not expose any information on the dynamics or real-world side effects of the service.

To summarize, WSDL addresses the problem of automated execution of and communication with Web Services. SOAP is proposed as unified messaging protocol for Web Services, and a Web Service publishing a WSDL document on its ports and operations offered mandates itself to respond to SOAP-requests for invocation of these ports and operations.

3.2.3 Web Services Discovery – UDDI

UDDI (Universal Description, Discovery and Integration) is a standard proposed by companies such as Sun, IBM and HP. The standard specifies a Web Services description format and an architecture for a Web Services description repository. Hence UDDI addresses the issue of discovering Web Services. In extension to WSDL service descriptions, UDDI service descriptions focus on meta-data of the service: E.g., its business domain and pricing may be described.

Technically speaking, each service description in UDDI consists of a businessEntity element, akin to a White Pages element describing the contact information for a business. A businessEntity describes a business offering a service by name, categorization, services offered (businessService elements) and contact information for the business. A businessService element describes a service offered by the business using a name, categorization and multiple bindingTemplate elements. This can be considered to be analogous to a Yellow Pages ele-

ment that categorizes a business. A bindingTemplate element in turn describes the kind of access the service requires (phone, mailto, http, ftp, fax etc.), key values and tModelInstances. tModelInstances are used to describe the protocols, interchange formats that the service comprehends, i.e., the technical information required to access the service. Here, UDDI makes the connection to the previously described WSDL standard: UDDI explicitly recommends the use of WSDL documents to specify details of the service. WSDL documents are among the information stored in a service's tModel.

Next to a standard for describing meta-information of services, UDDI also describes an architecture for a service description repository. Such a UDDI server is a centralized server system which is capable of storing and retrieving service descriptions. The UDDI standard contains a specification of an API to access a UDDI server and publish, modify or remove a Web Service description. A service description encompassing BusinessInfo, ServiceInfo, BindingTemplate and tModel can be published in a UDDI repository, and the repository can be crawled by anybody looking for a particular service. Nowadays, public UDDI servers are operated by Sun, IBM, HP and SAP.

To summarize, UDDI proposes a way of describing services on a meta-level, and of storing and retrieving service descriptions stored at a UDDI server. It does not deal with matchmaking between service requests and descriptions – at maximum, UDDI may help to semi-automate the process of Web Services discovery: Users manually browse a UDDI server's service repository and search for services required by them.

3.2.4 Web Services Composition – WS-BPEL

The Web Services Business Process Exeution Language [19] has been proposed by IBM and aims at providing a means of composing Web Services. Building on WSDL, it specifies a grammar for creating a complex Web Service by composing other Web Services.

WSDL descriptions of existing Web Services are regarded as interfaces to these services. Operations exposed by these interfaces can be linked by WS-BPEL statements into a Web Service workflow. A WS-BPEL document thus closely resembles a state diagram in which states are tied to operations on Web Services interfaces and transitions between states are linked to conditions based on the outcome of Web Service operations. An entire workflow encompassing many different Web Services can be created – albeit manually: WS-BPEL does not deal with automatically composing or discovering Web Services. Known services are composed in order to create a more complex service.

To summarize, WS-BPEL provides a standardized format to describe the composition of Web Services into a more complex Web Service, again accessible via ports and operations. It does not deal with automatically generating compositions. It also does not cope with issues such as fail-safety (i.e., replacing a malfunctioning service on the fly with a backup service) or execution monitoring (i.e., halting a service execution if it takes too long or produces wrong results).

3.2.5 Shortcomings of Current Web Services Technology

WSDL, WS-BPEL, SOAP and UDDI are Web Service related standards. Making full use of their capabilities enables a Web Service infrastructure in which Web Services can be described, discovered, invoked and composed, apparently tackling all issues identified in Figure 3.1. On taking a closer look, these standards allow for describing a Web Service in a standardized format and publishing it in a UDDI repository – however, UDDI service descriptions merely provide some meta-data on the service and are often even natural language based, making it extremely difficult for automated algorithms to understand the exact semantics and capabilities of a service and choosing the correct service in order to complete a task. A WSDL document on the service does not help much here, it merely contains information on the syntax of accessing services, again, the information does not suffice for an automated algorithm to choose a service in order to reach a particular goal. The standards allow for composing services and creating more complex services – however, the WS-BPEL workflow models resulting from such compositions are executed in a deterministic, non-flexible way, with no means of simply declaring a particular service in the workflow as abstract and leaving it to a WS-BPEL execution engine to dynamically select a service at run-time. Or, moreover, to replace an existing service in the workflow dynamically by a new service featuring, for example, better quality of service. Thus, the question is: Are all Web Services use cases discussed in Section 3.1 already made possible by these protocols and standards?

In fact, WS-BPEL, WSDL, UDDI and SOAP provide solutions to low-level infrastructure problems in the context of Web Service communication, description, discovery and composition. Leveraging the power of Web Services, for example enabling *automated* Web Service discovery and composition, requires connecting the Web Service domain to research areas that deal with searching, processing and combining information. Referring to the envisioned Web Services protocol stack in Figure 3.1, WS-BPEL, WSDL, UDDI and SOAP are approaches to the problem domain of Web Services from a technical perspective. They provide basic infrastructure for communicating with, describing, discovering and composing Web Service. However, the algorithmic perspective is not touched by any of these languages and protocols.

Hence so far, these standards do not provide any means of re-planning a Web Services workflow on the fly in case a service fails. They are not able to choose the best service among a large set of available and semantically similar services. They cannot determine the exact capabilities of a service and decide whether the service might be of any use for solving another service's task in combination with further services.

3.3 Semantic Web Services

Figure 3.2 links the envisioned Web Services stack to Computer Science research areas which may contribute algorithmic excellence for an automated world of Web Service operations.

To motivate the introduction of an algorithmic perspective: Where does the current Web Services protocol stack miss important features?

Understanding service semantics. WSDL does provide a way of describing services – albeit on a low, rather technical level. Services are not defined in terms of their capabilities or semantics, only in terms of their syntax, i.e., in form of message types and operation formats. Automated service discovery requires a way of encoding operational service semantics in a machine-readable and processable way: An algorithm attempting to locate a service for solving a specific task has to be able to "understand" what a service does. Research in the *Semantic Web* area may provide valuable input: The Semantic Web will be addressed in Section 3.3.1.

Dynamic service publication and discovery. UDDI describes an infrastructure for publishing and reading service descriptions. Once a way of enhancing service descriptions with service semantics has been found (as mentioned in the previous bullet point), UDDI may as well be used with more expressive service descriptions. However, does it really solve the problem of discovering services in an up-to-date way? Today's Internet is marked by two major properties: Diversity and dynamics. New and extremely heterogeneous information pops up and disappears at amazing pace. A monolithic service publication architecture as UDDI, a centralized server that attempts to capture all services currently available on the global Internet, seems depreciated and obsolete. Centralized servers require permanent update of their information – for performance reasons, UDDI would not be able to mimic every single service's current pricing situation or activity status to choose an idle service when a request comes in, for example. Centralized servers are single points of failure – once a UDDI server fails, an entire network of services is threatened. Research on distributed storage and discovery of information in *Peer-to-Peer Systems* may provide an interesting alternative to centralized

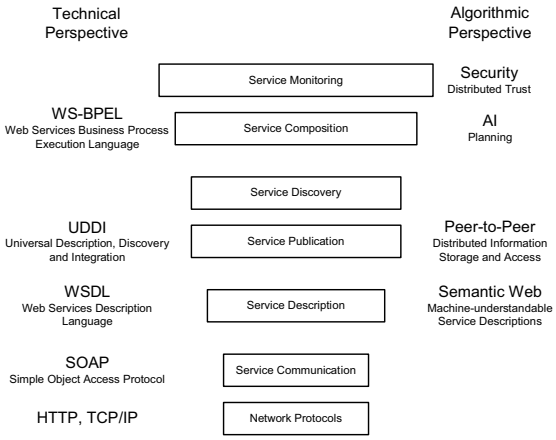

Figure 3.2: Web Services Protocol Stack – Technical and Algorithmic Perspective

service discovery as in UDDI. P2P and its suitability for Web Services discovery will be introduced in Section 3.3.2 and addressed in detail in Chapter 4.

Planning service compositions. Artificial Intelligence research has spawned results on planning tasks based on formal task descriptions [17]. Planning algorithms accept a goal formalized in a planning language (such as Golog [28]) as input, derive a plan which establishes preconditions and effects of the goal and use a repository of prefabricated service modules to instantiate and execute the plan. Planning algorithms may well be used in the domain of Web Services – however, one of the major challenges is to describe the capabilities and semantics of a Web Service in a way that may be understood by a planning algorithm. Again, *Semantic Web* research may provide valuable input.

Monitoring and trusting services. The diversity of the Web has been advertised as one of the reasons why enabling services on the Web is an attractive option. However, the openness of the Web also creates problems: Services may provide bad quality results or, worse, attempt to cause damage. In a Web of Services, some authority is required to ban malicious service providers from subverting the network. The authority may be a centralized system storing and processing evaluations of Web Services – yet such a system creates the same kind of problems as a centralized service discovery infrastructure, as discussed above. In Chapter 5, a solution is presented in which the entire Web of Services becomes an authoritative body evaluating each other's services.

3.3.1 Semantic Web

On the Internet, large quantities of data are directly available for human consumption. However, so far most of the information available is intended for merely exactly that, human consumption: It is yet very hard to built automated agents, which support humans in processing and filtering information, due to the unstructured representation of data on the Web in form of HTML pages, images, natural language text etc. The solution to this problem comes in form of formally defined, linked data on the Web. Machine algorithms require data to be stored in formalized syntax in which terms used are associated with predefined semantics. Semantic Web research aims at defining knowledge representation methods which can be used in such manner.

A crucial role in these efforts is played by ontologies. Ontologies, to state a popular definition, are a formal, explicit specification of a shared conceptualization [46] of a domain. (For a brief introduction to ontologies, refer to Section 4.7.1.1.) Essentially, ontologies capture knowledge about a particular domain in a machine-processable way. On the Semantic Web, they are used as a common vocabulary to mark up content: While it is hard for an algorithm to understand what a Web site is about by attempting to understand its natural language content, the algorithm will be able to process meta-data attached to the site which uses vocabulary defined in an ontology – since such vocabulary has been a priori associated with a commonly agreed upon meaning.

Web Services may inherit in two important ways from Semantic Web research. First, sophisticated tasks such as Web Service description, discovery and composition require algorithms which are capable of understanding the semantics of a particular service, i.e. what it actually *does*. The Semantic Web approach of adding formalized, ontology-based meta-data to a Web site may be used here. An approach linking Web Services to ontologies to accomplish this will be covered in Section 4.7. Second, Semantic Web research deals with devising algorithms that are capable of using formal content (or service) descriptions for matching [37], composition [32], learning [3] and other high-level tasks based on formal descriptions. While this work will not explicitly deal with service composition, Section 4 introduces an infrastructure for Semantic Web Services in which services can be efficiently discovered based on formal descriptions, also enabling efficient composition algorithms in a later step.

3.3.2 Peer-to-Peer Networks

Peer-to-peer (P2P) networks are a novel paradigm of storing and accessing information on a network: As opposed to centralized storage systems or client-server approaches where single servers index data in a large-scale system, P2P networks simply connect the data sources in a

network. If a particular data item is searched, no single spot in the system is asked – instead, a query message is broadcasted among all data sources in the network, and sources capable of answering the query respond.

As compared to centralized storage of information, this approach exhibits some interesting properties:

Diversity and Equality. Any peer is able to share any content in the network, and all peers have equal exposure to the network. Content on a P2P network is searched dynamically, by asking as many peers in the network as possible for requested content. The participation of peers in the network may be very dynamic, the peer "audience" may change rapidly – yet at all times, many different peers sharing very diverse content populate the network.

Dynamics. In P2P networks, information is always discovered and downloaded fresh from a source of information – centralized storage systems require updating information stored at a central server when the cached information has changed. For example, the search engine Google crawls less popular sites in periods of up to three months – a change in content on these sites in between the time of two crawls will not be realized by the search engine, and it will not be considered during searches for the sites.

Redundancy. P2P networks do not have any single point of failure. Content is stored at many peers, and popular content is replicated automatically throughout the network (by virtue of peers deliberately downloading and storing content from other peers). If networks are organized in a centralized manner, usually single points of failures such as a centralized server exist: These servers may fail and thus disrupt the entire system operations, they may be attacked and brought down and they are a bandwidth bottleneck since they form a core building block of the system. In reality, P2P networks feature a good deal of redundancy and invulnerability against node failures – however, unorganized P2P networks such as the Gnutella network turn out to be surprisingly vulnerable against particular node failures and attacks. This issue will be addressed in Chapter 4.

Censorship resistance. Due to the lack of a central authority, any peer can participate in a P2P network and provide content. While certainly enriching the amount of information available on the network, the total lack of control over shared content and peer behavior, however, also imposes problems of security and authenticity on the network. These issues will be discussed in Chapter 5.

3.3.3 Putting Everything Together

Using Semantic Web technology and peer-to-peer networking combined with existing Web Services technology may allow for the construction of a flexible Web of Services: A large-scale P2P network of Semantic Web Service providers, each of which has marked up its Web Service with machine-processable information on its exact capabilities, can be queried for instantiating a complex task description. The task description is broadcasted throughout the network, and peers which can possibly contribute to solving the task respond. In the next chapters, a P2P infrastructure for such a Web of Services will be presented, and an algorithm for managing trust in such a network will be described.

Chapter 4

Distributed Service Discovery

Service discovery requires crawling a large pool of service providers to identify a provider that is offering a particular service. In highly dynamic domains where service providers appear, disappear and change status frequently, centralized service repositories which store a large number of service descriptions and which can be queried in order to find a particular service are hard to install. In this chapter, service providers will be organized into a peer-to-peer network. A query for a particular service can be issued at any point in the network, will be propagated among potentially interesting service providers and instantaneously answered by many different providers. The network can scale up to millions of peers, cope with peers frequently logging on and off and does not exhibit any single and vulnerable point of failure (such as a central server).

However, P2P networks evolving in an unorganized manner suffer from serious scalability problems, limiting the number of nodes in the network, creating network overload and pushing search times to unacceptable limits. In this chapter, these problems are addressed by imposing a deterministic shape on P2P networks: A graph topology is proposed which allows for very efficient broadcast and search, and a broadcast algorithm is described that exploits the topology to reach all nodes in the network with the minimum number of messages possible. An efficient topology construction and maintenance algorithm will be provided which, crucial to symmetric peer-to-peer networks, does neither require a central server nor super nodes in the network. Nodes can join and leave the self-organizing network at any time, and the network is resilient against failure. Moreover, the scheme can be made even more efficient by using a global ontology to determine the organization of peers in the graph topology, allowing for efficient concept-based search.

4.1 Why P2P Networks Do Not Scale

Beyond the pure Gnutella-style P2P networks [21], other network types have evolved. This section will discuss their upsides and downsides and introduce the approach to P2P search presented in this work. Section 4.1 introduces different types of P2P networks and how they attempt to provide efficient access to distributed information. Also, the basic idea behind the approach to P2P search presented in this work will be outlined. Section 4.2 describes the P2P network topology used in this approach and its suitability for efficient broadcast and search. Section 4.3 presents a distributed algorithm, dubbed HyperCuP, which is capable of maintaining the graph structure efficiently, and elaborates the algorithm on a detailed example. Section 4.4 discusses simulation results of the algorithm. Section 4.5 embeds the topology presented into a larger class of graph topologies and describes applications of the algorithm in this context. Section 4.6 presents an implementation of the algorithm on the popular P2P infrastructure JXTA. In Section 4.7, an extension of HyperCuP is presented by the use of ontologies for partitioning the network. Finally, Section 4.8 covers related work, and Section 4.9 concludes the chapter.

4.1.1 Scale-free Networks

Currently deployed P2P networks such as Gnutella [21] are mostly peer-to-peer networks in the genuine sense: A random coupling of peers on top of a transport network such as the Internet. However, these P2P networks suffer from serious drawbacks, effectively hampering their deployment on a larger scale and for more mission-critical purposes than file sharing (which is the most popular use case of P2P these days). It has been observed that the node degree distribution of P2P networks with "uncontrolled evolution" mimics a power-law distribution (Section 4.4). These networks (so-called exponential or scale-free networks) exhibit drawbacks, including:

Scalability. As P2P networks grow to large numbers of peers, content cannot be searched efficiently any more. On Gnutella-style networks, information is searched by broadcasting query messages all over the network. This consumes many messages, produces overhead traffic due to messages delivered several times to identical peers and reaches many peers that are not capable of contributing anything to the resolution of the query issued.

Lack of search guarantees. Since searching by broadcast merely reaches a random set of peers in the network due to the random coupling of peers, it does not exhibit any guarantees on the results of the search process. Content that is actually available in the network may not be visible to many peers since messages get dropped or do not

travel that far, graph properties such as the diameter of the network (the shortest path between the two most distant nodes) deteriorate.

Vulnerability to attacks. Scale-free P2P networks rely on some nodes with high node degree, i.e. many links to other peers. If these nodes are struck by denial-of-service attacks and go down, the network can be partitioned quickly (see Section 4.4).

Uncontrolled evolution of a P2P network occurs when no scheme is imposed on the way nodes join and leave the network. Any peer can join and leave the network at any time and via any peer already in the network. Such networks are likely to grow to become exponential networks. However, they are theoretically symmetric (though not in practice due to the power-law node degree distribution): No peer has to execute more complex tasks than any other peer in the network. Searching is usually carried out as broadcast, normally with a hop-count horizon to avoid flooding the network with queries. More efficient search techniques [53] such as a distributed version of iterative deepening search can be deployed, too.

4.1.2 Centralized Server Networks

Napster [33] introduced a central server which indexed content available on the system's peers. If a peer searches for information, it simply issues a query to the central server which will provide pointers to peers that are able to provide the information. Despite the improvements on search performance, this marks the return to a centralized system with all the downsides involved, such as introducing a single point of failure, bandwidth bottlenecks, problems of keeping indexed information up-to-date etc.

4.1.3 Super-Peer Networks

Conceptually spoken, super-peer networks occupy the middle-ground between centralized and entirely symmetric P2P networks: They introduce hierarchy into the network in the form of super-peers, peers which have extra capabilities and duties in the network. Super-peers index content that is present on a particular number of leaf peers in the network. Queries are broadcasted among super-peers which forward them to the appropriate leaf peers. Though to a certain extent inheriting the search performance of a centralized P2P approach as well as defying the downsides of an entirely centralized system, super-peer networks loose symmetry: Some peers have additional duties. Thus, super-peer networks should be carefully engineered in order to work well [52]. Distributed algorithms which are capable of maintaining a super-peer network in a state which allows for efficient searching are yet to be described. [25] is a large-scale super-peer network: Peers in the network can opt to become super-peers and do more work than others. Still, there are no guarantees on the outcome of search processes, and still, the topology of the network may become inefficient due to its uncontrolled evolution.

4.1.4 Deterministic Topologies

Nodes in a network have a limited view of the network: They have a set of neighbors which determines their scope. Operations on P2P networks become ineffective primarily due to this fact. During message broadcasting, messages reach a single peer several times since more than one path of the network topology leads to this peer. Peers do not know where in the network a specific content that has been asked for might be located, hence their search technique is restricted to simple broadcasting, i.e. asking all their neighbors to forward the query to whomever they like to.

Deterministic topologies address this issue – not by giving any node a global view of the network, which would mean centralization, but by maintaining a deterministic topology of the network which is known to all nodes. Therefore, nodes at least have an idea of what the network beyond their scope looks like. They can use this globally available information to reach locally optimal decisions while routing and broadcasting search messages. The information on the topology is packaged in a protocol that is used to police peers joining and leaving the network: Instead of allowing peers to join and depart without any restrictions, peers are made to connect to specific peers already in the P2P network upon joining – in a way that maintains the topology in the desired state at virtually every moment in time.

4.2 Searching Semantic Web Services

In this work, peers in a P2P network are organized into a hypercube topology. In the context of Semantic Web Services, this allows for efficient distributed discovery of requested Web Services.

4.2.1 Organizing Peers into a Hypercube Graph

Figure 4.1a depicts a hypercube for a base $b = 2$, a topology that has been studied before in the area of multiprocessor machines [38], but under different assumptions (multiprocessor machines do not have to deal with a highly varying number of nodes in the topology). A complete hypercube graph consists of $N = b^{d_{max}}$ nodes and is defined by the fact that all nodes have $(b-1) \cdot (d_{max})$ neighbors, $(b-1)$ in each 'dimension' – where d_{max} is essentially the number of dimensions spanned by the cube (in Figure 1, the cube has three dimensions, and d_{max} is 3). The network diameter, defined as the shortest path between most distant nodes in terms of node hops, is $\Delta = log_b N$. As visible, this structure is symmetric, i.e. no node incorporates a more prominent position than others. This is crucial for load balancing in the network: Every node can become the source of a broadcast (the root of a spanning tree of the network), yet the load will always be shared equally. The topology provides redundancy –

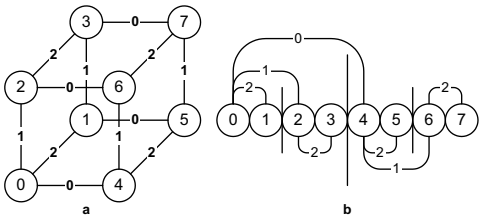

Figure 4.1: a. Hypercube graph b. Serialized notation (links incomplete)

its connectivity (the minimum number of nodes to be removed in order to partition the graph) is optimal, i.e. equal to $nodedegree - 1$. Power-law networks such as Gnutella can easily be partitioned by bringing down highly connected nodes in the network through denial of service attacks, the hypercube topology is far less vulnerable to such attacks. The hypercube base$b$ can be chosen to adjust the network diameter and node degree. Note at this point that the construction algorithm that will be described in Section 4.3.1 works well with node numbers that are not equal to those in complete hypercubes, allowing for any number of peers in the network. To describe the topology of a graph $G = (V, E)$, some definitions shall be stated. In the following, the descriptions will use hypercubes with a binary base as examples. Edges in the graph are labeled: Node Y is dubbed i-neighbor of node X if node Y is X's neighbor along dimension i. For example, in Figure 4.1, node 5 is the 2-neighbor of node 4. Node 5 is also dubbed 4's neighbor along dimension 2. Edges in the graph are undirected, i.e. node 4 is also 5's 2-neighbor. Edge labels start at $i = 0$. The maximum dimension of a graph is termed d_{max}.

4.2.2 Broadcast and Search Algorithms

Based on this terminology, a broadcast scheme can be defined which guarantees that nodes receive a message exactly once. It is guaranteed that exactly $N - 1$ messages are required to reach all nodes in a topology. Furthermore, the last nodes are reached after$\log_b N$ forwarding steps. Any node can be the origin of a broadcast in the network, satisfying a crucial requirement. The algorithm works as follows: A node invoking a broadcast sends the broadcast message to all its neighbors, tagging it with the dimension of the link on which the message was sent. Nodes receiving the message restrict the forwarding of the message to those links leading to higher dimensions. As an example, refer to the serialized notation of the network graph in Figure 4.1b (for clarity, only the links used in the example are depicted – however, one can just copy all links in 4.1a into this notation to arrive at the full picture): Node 0 sends a broadcast – at first to all its own neighbors, viz. nodes 4, 2 and 1. Node 4 receives the message on a link tagged as a dimension 0 link, i.e. it forwards the message only to its 1-

and 2-neighbors, namely 6 and 5. At the same time, node 2 which has received the message on a dimension 1 link forwards it to its 2-neighbor, node 3. In the third forwarding step, node 6 relays the message to node 7, again its 3-neighbor. The characteristic path length in this scheme can be calculated as

$$L = \frac{1}{N-1} \cdot \sum_{i=1}^{log_b N} \frac{(b-1)^{log_b N - i + 1}}{(log_b N - i)!} \cdot \prod_{j=0}^{log_b N - i} (i+j) \tag{4.1}$$

which is about $0.5 \cdot log_b N$. The equation is discussed in Section 4.3.2.2.

The above scheme constructs a spanning tree over the hypercube graph. A drawback of the scheme is that the spanning tree is imbalanced: In the binary hypercube, the number of messages sent during a broadcast varies between $log_2 N$ and 1 from node to node, i.e., the workload of nodes participating in the broadcast is different from node to node. However, a balanced spanning tree can be constructed with a similar algorithm as described in [23]. The spanning tree is still greedy, i.e., its height is $O(log_2 N)$, and it can be constructed in a distributed manner. In the scheme in [23], every node invests the same amount of work (or rather, number of messages sent) into a broadcast.

A search in a hypercube is essentially a broadcast with a time-to-live, i.e. a broadcast with a limited scope. The maximum number of nodes is reached with a given number of messages. A search can also be referred to as broadcast with limited dimension horizon: If a broadcast is started on dimension d instead of on dimension 0, only nodes that can be reached by dimension hops of dimensions larger than d will receive the search message.

An algorithm for shortest-path routing on the hypercube can also be specified: A node at position $\vec{p}_i$ attempting to route a message to position $\vec{p}_j$ sends the message to its neighbor on a dimension which is marked as 1 in $\vec{p}_i \oplus \vec{p}_j$. This neighbor node executes the same algorithm. Subsequently, all binary digits will be 'corrected'. For example, a node at position 010 attempts to send a message to position 101. A path that can be constructed by the above rule is: $010 \rightarrow 110 \rightarrow 100 \rightarrow 101$. The maximum length of such a path is equal to $\log_2 N$ on the binary hypercube since the maximum Hamming distance between any two positions corresponds to the number of binary digits of the position vectors, i.e.,$\log_2 N$.

4.3 Building and Maintaining Hypercube Graphs

In the following, a distributed algorithm is outlined which allows peers in a P2P network to build a hypercube topology. Here, the major challenges in P2P networks are as follows: To maintain network symmetry, crucial for P2P networks, any node in the network should be

allowed to accept and integrate new nodes into the network. Furthermore, joining and leaving the network are to consume a reasonable amount of message transmissions to limit the traffic imposed on the transport network. Clearly, a joining node should not have to register with all nodes in the network, i.e., the protocol is to beat a message number of$O(n)$ for node joins and removals.

4.3.1 A Distributed Topology Construction and Maintenance Algorithm

In the following, a construction and maintenance protocol of a binary hypercube P2P topology will be described. The section will also feature a walk through an example by having 9 peers joining a network, and one peer leaving during the process, to elaborate on the basic idea of the construction and maintenance algorithm. The construction and maintenance algorithm is based on the notion that nodes in an evolving hypercube graph take over responsibility for more than one position in the hypercube. The idea is to have the hypercube topology of the next biggest complete hypercube graph already implicitly present in the current topology state. Upon arrival of new nodes, the complete hypercube topology unfolds as needed. Upon removal of nodes, other nodes jump in to cover the positions previously covered by the node that left the topology, prepared to give these positions up again as new nodes join. Since the complete hypercube topology is implicitly preserved, the broadcast and search algorithms do not have to change either – still, every peer receives a broadcast message exactly once.

An algorithm which follows this general scheme will be presented in the following. The algorithm follows a framework: Nodes joining the topology are allowed to ask any already integrated node for integration. Then, the following steps are carried out.

1. **Integration dimension selection.** The node that is to carry out the integration selects a dimension along which to integrate the joining node.

2. **Integration champion node appointment.** The node that has been asked by the joining node for integration does not necessarily have to integrate the new node itself: It might be necessary to pass on the integration responsibility to a neighbor node which then becomes the integration champion node for the joining node. Also, in some cases several integration champions will carry out the integration in cooperation.

3. **Node integration.** The integration of the new node is carried out: The new node is assigned one or more positions on the hypercube and connected to its new neighbors.

4. **Node departure.** If a node leaves the topology, it is to carry out a departure protocol. Essentially, its former neighbors are connected to new neighbors, and the topology is modified such that it always closely resembles a hypercube topology.

The algorithm attempts to build a complete hypercube topology. However, if there are less than 2^d peers in the network, there are not enough peers to build a hypercube topology of dimension d. In this case, peers in the network acquire responsibility for more than one position on the hypercube topology. This usually means that such a peer is connected to additional neighbors, viz. to all those peers that are neighbors to one of the positions on the hypercube that the peer currently covers. When a new peer joins the topology, a peer that currently covers more than one position on the hypercube passes on one (or more) of its positions to the new node. The new node is then connected to the neighbor nodes of these positions. Analogously, if a node departs from the network, it assigns the responsibility of covering the positions it used to cover itself to some of its neighbor nodes. Since all these processes take place in a distributed way, simple rules are required based on which peer joins and departures are carried out.

As the most basic rule, an ordering on the dimensions in the hypercube is assumed: Lower dimensions correspond to larger distances. For example, peers 4, 2 and 1 are connected to peer 0 in Figure 4.1, yet peer 4 is farthest from peer 0, and peer 1 is closest to peer 0, since peer 4 is a neighbor on dimension 0, whereas peer 1 is linked to peer 0 by a dimension 2 link. In figuring out which peers have taken over responsibility for which vacant positions on the hypercube, the simple rule is applied that if a position is vacated by a departing peer, the closest peer takes over. For example, if peer 1 leaves the network in Figure 4.1, peer 0 takes over peer 1's position. This idea will be algorithmically formalized below.

4.3.1.1 Data Structures

A node V stores two types of data structures: A position vector $\vec{p}_v$ and a cover map vector $\vec{c}_v$. The position vector denotes the actual position of a node on a binary hypercube. The cover map reflects the current coverage of other positions by the node: A 1 in the cover map at position d means that the node is covering a position along dimension d. Cover maps and position vectors are encoded in d_{max} bit long numbers which limits the maximum number of nodes in the network to $2^{d_{max}}$. However, d_{max} can simply be set to 64, for example (the bit width of a long integer) – in which case the maximum number of nodes in the network is 2^{64}, a number that will probably not be exceeded even in massively large-scale P2P networks.

For each of its neighbor nodes W, a node stores a tuple $W = (\vec{p}_w, addr_w)$, where $\vec{p}_w$ denotes the neighbor node's position vector and $addr_w$ the neighbor node's transport network address to which it can be sent messages. These tuples are assembled in a node's neighbor set N.

The distance between two positions on the hypercube (i.e., between two nodes located at these positions) is expressed as the Hamming distance between their position vectors, i.e.,

$$Hd(\vec{p}_x, \vec{p}_y) = \|\vec{p}_x \oplus \vec{p}_y\| \tag{4.2}$$

For example, the distance between node 0 ($\vec{p}_0 = (000)^T$) and node 7 ($\vec{p}_7 = (111)^T$) in Figure 4.1 is $Hd((000)^T, (111)^T) = \|(000)^T \oplus (111)^T\| = \|(111)^T\| = 3$. (The vector notation for numeric examples will be omitted for brevity in the following.)

The dimensionality of a link between two nodes is expressed by

$$Lcd(\vec{p}_x, \vec{p}_y) = f(\vec{p}_x \oplus \vec{p}_y) \tag{4.3}$$

where function $f(\vec{b})$ returns the position of the most significant bit set to 1 in the binary vector $\vec{b}$. For example, the link dimensionality between nodes 0 and 6 in Figure 4.1 would be $Lcd(000, 110) = f(000 \oplus 110) = f(110) = 0$. (The most significant bit is the leftmost bit, and the enumeration of bits starts with index, or dimension, 0.)

Immediate neighbors are nodes which would also be neighbors if the hypercube topology was complete, i.e., all nodes were present. A node V is an immediate neighbor of a node W if and only if

$$Hd(\vec{p}_v, \vec{p}_w) = 1 \tag{4.4}$$

In Figure 4.1, nodes 0 and 1 are immediate neighbors, nodes 4 and 7 are no immediate neighbors (two hops are required to reach node 7 from node 4). Clusters within the hypercube are sets of positions on the hypercube: A cluster of hypercube positions are all positions that are located within a certain distance from a particular position within the cluster. To specify a cluster, state any position within the cluster plus a maximum and minimum dimension. For example, the cluster of positions $\{0, 1, 2, 3\}$ on the hypercube in Figure 4.1 is determined by a maximum dimension of 2 and a minimum dimension of 1: If one starts from any position 0, 1, 2 or 3 and follows an arbitrary number of dimension 1 and/or 2 hop, one will always end up on any other position out of 0, 1, 2 or 3. Similarly, positions 0 and 1 form a cluster described by a maximum and minimum dimension of 2.

Link sets are used to tag links between nodes: This operation is not part of the algorithm, yet link sets serve well to explain the operations of the algorithm and will be made use of in the example walk through in Section 4.3.4. A node can have extended neighbors $Y = H(X) = \{x_0, x_1, \ldots\}(X)$, where H is termed neighbor link set, and it denotes the sequence of dimension i hops one would have to follow in the complete hypercube graph to reach

node Y from node X and vice versa. In the example in Figure 4.1, the neighbor link set $\{0,1\}$ leads from node 1 to node 7 and back, i.e. $1 = \{0,1\}(7)$ and $7 = \{0,1\}(1)$. Note that edges in the graph are undirected, thus the commutative property of link sets always holds.

The algorithm consists of the steps described in Section 4.3.1. It is executed as soon as a new node attempts to join the HyperCuP P2P network by contacting any node which then becomes the integration champion node. It is also executed by a node which leaves the network.

4.3.1.2 Integration Dimension Selection

The algorithm is kicked off when a new node contacts a node, say node V, which is already integrated into the hypercube topology, and requests integration into the topology. It is not important how the new node became aware of node V in the first place – a real-world implementation of the protocol may choose to elect some nodes which are always up and whose transport network (e.g., IP) addresses are published on the Internet. In fact, this is the approach taken in the JXTA implementation of the protocol, to be described in Section 4.6.

Again, the basic idea of the algorithm is that there will seldomly be exactly 2^d nodes in the network which could form a complete hypercube topology. Thus, there will be vacant positions on the hypercube grid, and these positions have to be filled by newly joining nodes. At first, node V therefore searches for vacant positions in its neighborhood to select an integration position for the new node. An integration position will be in the immediate neighborhood of node V (since this is the scope of the network that node V has) at a one-hop distance, thus it is sufficient for node V to choose an integration dimension along which the new node is to be integrated. This dimension specifies the hop to be taken from node V's position to the actual integration position.

Node V is referred to as the initial integration champion node: The newly joining node expects node V to take the necessary steps to integrate the new node into the topology. At first, the integration champion node itself checks if it currently covers any additional positions. If so, it sets the integration dimension d_{int} equal to the position of the most significant 1 in its own cover map. In this case, node V opts to integrate the new node on one of its own currently covered positions.

If node V's cover map consists of all 0s, the integration champion node V checks if any of its neighbor nodes currently has a vacant neighbor position. The node identifies non-immediate neighbors by checking the Hamming distances to its current neighbor nodes. All non-immediate neighbors are connected to the integration champion node by a temporary

link since they only cover the actual neighbor position of the integration champion node in place of a missing node. (Otherwise, they would be node V's immediate neighbors and have a Hamming distance of 1 to node V.) If the integration champion node detects a non-immediate neighbor, it sets the integration dimension d_{int} as follows:

$$d_{int} = min_{\forall w \in N_v}(Lcd(\vec{p}_v, \vec{p}_w)) \tag{4.5}$$

for a d_{int} on which node V does *not* have an immediate neighbor. This policy is implemented to balance the resulting graph (also see Section 4.3.3): Nodes are to be integrated on vacant dimensions as low as possible to fill up these dimensions first. As an example, if there are 4 nodes 0, 1, 2, 3 in the network, a 2-dimensional hypercube can be built – there is no reason for building a 3-dimensional graph where nodes 1, 2, 3 are 0-, 1-, 2-dimension neighbors of node 0. Hence the reason for the policy to fill up low vacant dimensions first is to build the most "dense" hypercube topology at any time: If there are N nodes, the maximum dimension used in the graph should be $\lceil \log_2 N \rceil$. Section 4.3.2 will quantify what happens if this is not the case, Section 4.4 will present simulation results which show that this goal cannot always be satisfied perfectly.

The integration position of the new node is computed as the node's own position $\vec{p}_v$ with the digit at position d_{int} inverted (this reflects the dimension hop along dimension d_{int} to reach the integration position from node V's position):

$$\vec{p}_{int} = (p_v^0, p_v^1, \ldots, p_v^{d_{int}-1}, \overline{p_v^{d_{int}-1}}, p_v^{d_{int}+1}, \ldots, p_v^{d_{max}-1}) \tag{4.6}$$

4.3.1.3 Integration Champion Node Appointment

The actual integration champion node is identified. It is set to be node V's neighbor node with the smallest Hamming distance to the integration position. The integration control is forwarded to the selected node to have it carry out the integration. It is possible that the initial integration champion, node V, is not the integration champion which will actually carry out the integration. If node V does not have own vacant positions to be assigned to the new node and selects a position covered by one of its non-immediate neighbors as integration position, the respective non-immediate neighbor node is to carry out the integration and to give up 'its own' position.

4.3.1.4 Node Integration

During the integration of the new node, it will be assigned one (or more!) positions on the hypercube. It is on the integration champion node to inform all future neighbors of the new node in order for them to link to it. Two types of future neighbors for the new node are distinguished:

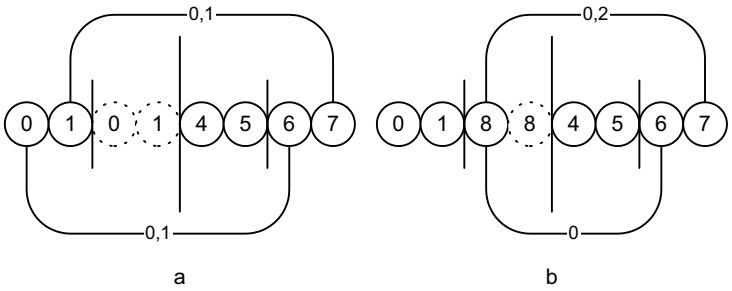

Figure 4.2: Construction Example

Prospective neighbors. Nodes which are neighbors of one of the positions that the new node will take over are prospective neighbors: They used to link to the node which is about to give up a position on the hypercube to the new node, i.e., they used to be connected to the integration champion node.

Integration champions. Nodes which currently cover one of the positions that the new node will take over are integration champion nodes: To modify the definition of an integration champion node given above, several integration champions can actually exist during an integration process. It has already been stated that the initial integration champion may pass on the responsibility to integrate the new node to another node, and it may also happen that this responsibility is passed on to *several* other nodes. This is the case if the new node takes over a set of positions on the hypercube (instead of only one position) whose coverage is partitioned among several nodes. Since all these nodes then give up positions during the integration process, they all become integration champions.

In the example in Figure 4.2, nodes 0 and 1 currently cover positions along their 1-dimensions, thus serving as 0-neighbors for nodes 6 and 7 (only the links between these nodes are depicted, all other links are omitted). Also note the link sets tagging these links – node 0 can be reached from node 6 by a dimension 1 and a dimension 0 hop, thus leading to the link set $\{0, 1\}$. Node 0 then starts integrating node 8 on one of the vacant positions. Following the general rule that vacant positions are always covered by nodes which are closest to them, node 8 is to take over the position covered by node 1, too. Hence both node 0 and node 1 act as integration champion nodes, passing on temporarily covered positions to node 8 and informing the prospective neighbors 6 and 7 of the change in their neighborhood.

Formally, the first elected integration champion node (it was elected after the integration position had been chosen as described in Section 4.3.1.2, thus it may actually already be the second node in a row which deals with the integration of the new node) identifies and notifies other integration champion nodes. If node V at position $\vec{p}_v$ is integrating a new node at

position $\vec{p}_{int}$, any node W in the network is integration champion if it is closer to node V than node V is to the integration position $\vec{p}_{int}$, formally if

$$\|\vec{p}_w \oplus \vec{p}_v\| < \|\vec{p}_{int} \oplus \vec{p}_v\| \tag{4.7}$$

The reason for this step is that there can be no node closer to the integration position $\vec{p}_{int}$ than node V, due to the fact that it is currently covering the position $\vec{p}_{int}$. However, there may be other vacant positions close to position $\vec{p}_{int}$ which are currently covered by other nodes than node V. This is the case when node V has one or more neighbors (say, node W) along higher dimensions than the integration dimension, i.e. $Lcd(\vec{p}_v, \vec{p}_{int}) < Lcd(\vec{p}_v, \vec{p}_w)$: All these nodes will also cover a position in the cluster in which $\vec{p}_{int}$ is located since they are closer to one of the vacant positions there than node V. For example, in Figure 4.2, both nodes 0 and 1 cover one position each in the cluster where node 8 will be integrated. Node 1 is closer to the position it covers than node 0 – from node 0, a path consisting of a dimension 1 and dimension 2 hop leads to the position covered by node 1, from node 1 itself just one dimension 1 hop is sufficient. Node 1 is closer to node 0 than node 0 is to the integration position, satisfying the condition for node 1 also being an integration champion node.

Node V does not necessarily have to be linked to all mandatory integration champion nodes. Hence it carries out a limited broadcast on the hypercube: All nodes which satisfy Equation 4.3.1.4 have to be informed, and every node of this set can be reached by one or more hops of dimension $d_{int} + 1$ at minimum. Thus, node V sends out a broadcast with a minimum dimension of $d_{int} + 1$ – all nodes receiving the broadcast are appointed integration champion nodes, and they carry out the steps described in the following.

The new node's integration position has been computed as $\vec{p}_{int}$. However, it still lacks an initial cover map $\vec{c}_{int}$. The cover map is set to

$$\vec{c}_{int} = (c_v^0, c_v^1, \ldots, c_v^{d_{int}-1}, 0, 1, 1, \ldots, c_v^{d_{max}-1}) \tag{4.8}$$

The new node does have a neighbor along dimension d_{int} – the integration champion node(s). Also, it does not have any neighbors on higher dimensions since otherwise its new position would not have been covered by the integration champion node(s). Thus, its cover map is filled with 1s on dimensions higher than d_{int} and has a 0 on dimension d_{int}. For dimensions lower than d_{int}, the cover map is simply a copy of node V's cover map. In general in a binary hypercube, nodes which are linked on dimension i perceive the same neighbor situation for dimensions lower than i: If there is at least one node in the cluster reachable by a dimension $d < i$ hop, this node will cover all positions in that cluster since it is closest to the vacant

positions in the cluster. If there is no node in the cluster, it is missing, and all nodes outside of the cluster will regard it as missing. For example, in Figure 4.2, nodes 0 and 1 both perceive the cluster along their dimension 1 as missing and the cluster along their dimension 0 as not missing since it is populated by nodes. Hence V's cover map is copied into the new node's cover map for dimensions lower than d_{int}.

By the broadcast, all integration champion nodes have now been informed of the fact that they have to integrate a new node on the selected integration position. Hence they all carry out an identical integration algorithm, to be described in the following.

Each integration champion node updates its own cover map by flipping bit d_{int} – each of them now has a neighbor along dimension d_{int} (the new node) and does not cover any position along this dimension any more.

Since an integration champion node gives up covered position(s) to the new node, it will also pass on links to nodes that it used to link to by itself. To determine which nodes will become neighbors of the new node, two lists are computed: The list $L_{champion}$ of positions that remain covered by the integration champion node, and the list L_{new} of positions that will be covered by the new node after the integration has been carried out.

$$L_{champion} = (\vec{p}_{champion,0}, \vec{p}_{champion,1}, \ldots, \vec{p}_{champion,2^{\|\vec{c}_{champion}\|-1}}) \tag{4.9}$$

$$L_{new} = (\vec{p}_{new,0}, \vec{p}_{new,1}, \ldots, \vec{p}_{new,2^{\|\vec{c}_{new}\|-1}}) \tag{4.10}$$

Here, $\vec{c}_{champion}$ denotes the cover map of an integration champion node, and $\vec{c}_{new}$ denotes the cover map of the new node.

The positions are computed as follows: A node's actual position $\vec{p}_v$ is regarded as root position, where 1s in its cover map are considered as "don't cares" in this position vector which can be either 0 or 1. Hence the number of covered positions is only determined by the weight $\|\vec{c}_v\|$ of (the number of 1s in) the corresponding cover map. As an example, if a node is located at position $\vec{p} = 011$ and its cover map is $\vec{c} = 101$, its list of covered position is $L = (011, 010, 111, 110)$. This operation is spelled out as

$$L_v = \vec{p}_v \times \vec{c}_v \tag{4.11}$$

First, all prospective neighbors of the new node will be identified. Nodes whose neighbor position is a position that will be covered by the new node have to become a neighbor of the new node. Note that the integration champion node has to be linked to all these nodes by

definition: It is currently covering a superset of the new node's position(s) and thus currently has to be connected to all of its future neighbors. This is ensured by the departure protocol, to be described below.

Thus, for each position $\vec{p}_i \in L_{new}$, the integration champion node computes all neighbor positions along dimensions $0 \leq d < d_{int}$ where $c^d_{champion} = 0$ as

$$\vec{p}_{neighbor(p)} = \vec{p} \oplus \vec{d} \tag{4.12}$$

where $\vec{d}$ is a vector filled with 0s and a single 1 at dimension d. For each neighbor position $\vec{p}_{neighbor(p)}$, the integration champion identifies the node W in its set of neighbors which is closest to this position. Actually, the algorithm is searching for the node *right on* position $\vec{p}_{neighbor(p)}$ since this node would be the immediate neighbor of position $\vec{p}$ and thus a future neighbor of the new node. However, other nodes in the neighbor set of node V may be covering, too, hence the immediate neighbor of position $\vec{p}$ may be missing and another node which is identified as the node closest to the position is covering it. This is node W, the node at least *closest* to position $\vec{p}_{neighbor(p)}$.

Thus, node W will become a neighbor of the new node. It is yet to be determined if, at the same time, node W also stops being a neighbor of node V: To find out, for each position $\vec{p}_i \in L_{champion}$, the integration champion node computes all neighbors positions along dimensions $0 \leq d < d_{int}$ following the same algorithm as described above. The positions computed that way are the neighbor positions of any of the hypercube grid positions that remain covered by node V after the integration is finished. If, out of all nodes in the neighbor set of node V, node W is the node with the smallest Hamming distance to any such position, it will remain a neighbor of the integration champion node since it is either an immediate neighbor of one of node V's covered position or itself is covering a position that is a neighbor position of one of node V's covered positions. If node W does not have the overall smallest Hamming distance to any such position, it can be removed as a neighbor of node V: It is entirely passed on as a neighbor to the new node and stops being a neighbor of node V. In any case, node W has been identified as a neighbor for the new node which it is informed about by a message sent by the integration champion node. Upon reception of this message, node W will connect to the new node and become its neighbor. In doing so, node W also informs the new node about its transport network address and its position on the hypercube, $\vec{p}_w$.

In the computation of the neighbor position, only dimensions d are considered which are not marked as 1 in the integration champion node's cover map. The reason is that due to the way the new node's cover map is constructed (by copying the integration champion's cover map for dimensions $d < d_{int}$) these dimensions are still covered by the new node itself, hence

along these dimensions the node is "its own" neighbor.

Finally, the new node is entered into the list of neighbors by the integration champion node.

4.3.1.5 Node Departure

If a node leaves the network, it is to carry out a departure protocol to keep the topology in a clear state. This is vital since all nodes carry out algorithmic steps independently from each other and by minimizing communication among them: Thus, it is important that all nodes can reliably expect the topology to be in a state that the algorithm expects it to be in, i.e., with the basic rule of nodes covering vacant positions closest to them on the hypercube grid valid everywhere.

If a node departs from the network, it vacates one or more positions on the hypercube grid. The coverage of these positions must be taken over by other nodes.

First, the departing node (say, node V) selects a buffering dimension, i.e., the dimension along which it chooses nodes that will take over its positions. The buffering dimension d_{buf} is set to the highest link dimensionality in the set of node V's neighbors:

$$d_{buf} = max_{\forall W \in N_v}(Lcd(\vec{p}_v, \vec{p}_w)) \tag{4.13}$$

In the binary hypercube, one or two nodes will thus become the buffering nodes of the departing node V, i.e., node V will assign the responsibility for covering its positions to one or two other nodes in the network which are selected to be those nodes at link dimensionality d_{buf} from node V. The nodes are chosen based on the basic rule of a vacant position being covered by the node closest to it: By maximizing the buffering dimension d_{buf}, node V selects the nodes closest to its own position to become its buffering nodes. The buffering nodes are assembled in the list L_{buf}:

$$W \in L_{buf} \Leftrightarrow W \in N_v \wedge Lcd(\vec{p}_v, \vec{p}_w) = d_{buf} \tag{4.14}$$

For all nodes W in L_{buf}, their position $\vec{p}_w$ is known (since node V always knows the exact positions of every one of its neighbor nodes). The algorithm also requires an estimation of these nodes' cover maps $\vec{c}_w$ which is computed in the following way: For dimensions $0 \leq d \leq d_{buf}$, the buffering nodes' cover maps are identical to that of node V. This is due to the fact that the buffering nodes are located at link dimensionality d_{buf} from node V and thus have these parts of their cover maps in common (nodes see the same neighbor situation along dimensions lower than the link dimensionality between each other, see above). For dimensions $d > d_{buf}$, the nodes' cover maps are computed by pairwise comparing all node positions in L_{buf}:

$$\forall \vec{p}_i, \vec{p}_j \in L_{buf} c_i^x, c_j^x = 0 \Leftrightarrow Lcd(\vec{p}_i, \vec{p}_j) = x \tag{4.15}$$

If the link dimensionality between two node positions is x, then both nodes have a neighbor along dimension x and therefore do not cover any position along this dimension. Hence the corresponding digit in their cover maps is set to 0. Before running the algorithm, all estimated local cover maps $\vec{c}_w \in L_{buf}$ are initialized with 1s for dimensions $d > d_{buf}$.

For all nodes W in L_{buf}, their covered positions are computed. As described above, the combination of cover map $\vec{c}_w$ and position vector $\vec{p}_w$ – both are known now for every buffering node – yields the list of positions covered by node W. Most importantly, the root position that is used to compute a buffering node's covered positions is *not* its own position, $\vec{p}_w$, but

$$\vec{p}_w^{cover} = (p_w^0, p_w^1, \ldots, p_w^{d_{buf}-1}, \overline{p_w^{d_{buf}}}, p_w^{d_{buf}+1}, \ldots, p_w^{d_{max}-1}) \tag{4.16}$$

I.e., the actual position of node W is projected into the buffering cluster along dimension d_{buf} by flipping bit d_{buf} in the position vector $\vec{p}_w$. This is precisely the root position which will be taken over by the buffering node W. Note that this might not be the only position taken over by the buffering node W in the cluster to be buffered: If node W has vacant dimensions $d > d_{buf}$, it will take over additional positions. These positions, however, are computed with the help of node W's cover map, as described above, to $L_w = \vec{p}_w^{cover} \times \vec{c}_w$.

For all nodes W in L_{buf}, the algorithm then iterates through their covered positions L_w. These positions represent positions that used to be covered by the departing node and are now transferred to the buffering node W. Thus, for each position $\vec{p}_i \in L_w$, the integration champion node computes all neighbor positions along dimensions $d < Lcd(\vec{p}_w, \vec{p}_i)$ where $\vec{c}_w = 0$ as

$$\vec{p}_{neighbor(p)} = \vec{p}_i \oplus \vec{d} \tag{4.17}$$

where $\vec{d}$ is a vector filled with 0s and a single 1 at dimension d. This is similar to the algorithm executed when a new node joins the topology: The buffering node has to be connected to the neighbor nodes of the positions it will cover from now on. Again, only neighbor positions along dimensions $d < Lcd(\vec{p}_w, \vec{p}_i)$ from a covered position are considered since higher dimensions have already been checked when d is decreased from d_{max} to 0 during the checking process.

For each neighbor position $\vec{p}_{neighbor(p)}$, the departing node identifies the node X in its set of neighbors which is closest to this position. This node has to be linked to buffering node W as its new neighbor.

When the algorithm has finished (i.e., iterated through all buffering nodes, their covered positions and the neighbor positions of the covered positions), all positions currently covered by the departing node are assigned to one of the buffering nodes. Each buffering node is informed to which new nodes it has to connect in order to implement the new covering scheme. Then, the departing node is allowed to finally leave the network. The buffering nodes connect to all their new neighbors, again informing them about their position vectors and transport network addresses, and thus fixing the hypercube topology after the departure of node V.

4.3.1.6 Broadcast and Routing in Incomplete Hypercubes

The algorithms described in Section 4.2.2 can be used unchanged in any state of the HyperCuP topology. Nodes simply have to consider that they are now possibly covering several positions on the hypercube, and they have to carry out the broadcast and search algorithm for each of the positions. When a node which covers several positions receives a broadcast message, it forwards the message on behalf of all of its positions, always applying the basic broadcast rule of forwarding only to higher dimensions. Since a source of a broadcast never is hit by a message sent out again, a node issuing a broadcast from all of its covered positions will never be hit by the message sent out again. Thus, independent from the number of nodes or the exact topology in the network (as long as it is still guided by the HyperCuP principles, which is ensured by the proper execution of the protocol), HyperCuP broadcasts are optimal in terms of messages sent. A node will receive exactly one broadcast message, even if it covers several positions.

4.3.2 Algorithm Complexity

The complexity of the protocol and the arising topology will be discussed in the following.

4.3.2.1 Message and Maintenance Complexity

Two variables determine the complexity of the algorithm: Node degree and number of messages sent during its operations. The node degree is related to the amount of work that a node has to invest into maintaining connections to other nodes. It is important that a node be informed when one of its neighbors goes down, hence logical links between nodes in the topology have to correspond to reliable transport links on the network. This requires the use of a protocol such as TCP which ensures message delivery and recognizes link disruption. However, TCP connections impose maintenance overhead on a node since keep-alive packets have to be sent regularly. Thus, the amount of work a node has to invest into keeping up the topology during a period of time without node joins and removals is correlated with its degree.

The second variable is the number of messages that are sent during node joins and removals. Sending messages imposes traffic on the network and consumes network bandwidth. Thus, the number of messages to be sent determines the communication overhead of the protocol. The number of messages to be sent during node joins and removals is not difficult to determine: It is within the order of the new node's degree after integration. If a new node joins the topology, it will be connected to a number of neighbor nodes – each of which has to be informed of the new node. The process of informing these nodes (distinguished, as mentioned in Section 4.3.1, in prospective neighbors and integration champion nodes) directly determines the number of messages to be sent. If the new node will be connected to m nodes after the

integration has finished, $O(m)$ messages will be sent (some constant c is multiplied with m since messages are acknowledged and may be sent several times among two communicating nodes). An additional number of messages is generated in the randomization process. Here, $O(\log_2 N)$ additional messages are generated at maximum as will be described in Section 4.3.3.

This section focuses on computing a node's node degree in a particular situation in the network since both communication and link maintenance overhead are determined by the node degree. To estimate the complexity order of the node degree, two cases shall be considered. In the first case, the graph is balanced; in the second case, a worst-case situation is examined.

Balanced case. In the balanced case, every node in the graph is assumed to have d neighbors, i.e., the graph has d fully loaded dimensions. Also, $m < 2^d$ nodes in the graph have an additional neighbor, i.e., they have each opened up dimension $d+1$. In this case, $O(\lceil \log_2 N \rceil)$ messages will be sent upon the integration of a new node if one of the nodes missing a neighbor on dimension $d+1$ becomes the integration champion node. The proof is simple: The new node will have $d+1$ neighbors, and it is the integration champion node's task to inform these neighbors. Hence $O(d+1) = (\lceil \log_2 N \rceil)$ messages will be sent. Note that this allows for any number of nodes in the graph. Networks that reach a large number of nodes can scale down again to a small number of nodes (as long as this takes place relatively balanced, see Section 4.3.3): Higher dimensions that are added during the construction process are removed again automatically by the protocol (by nodes covering positions along these dimensions, just as described in Section 4.3.1.5) if no peer in the network has any neighbor in a dimension any more.

Imbalanced case. In the imbalanced case, the number of messages generated upon the integration of a new node is rising as compared to the balanced case. As worst case scenario, we consider the case where a node's dimensions are either missing or fully loaded. A node's dimension d is fully loaded if the node has an immediate neighbor on dimension d, and no node in the immediate neighbor's cluster with maximum dimension d_{max} (which is the maximum dimension in the graph) and minimum dimension $d+1$ is missing. For example, if one or more nodes of nodes 4, 5, 6, 7 were missing, nodes' 0, 1, 2 and 3 0-dimension would *not* be fully loaded although it would not be missing as long as one of the nodes 4, 5, 6 or 7 was still present. Node 0 would have a missing 1-dimension if nodes 2 and 3 were missing.

The scenario described above is a worst-case scenario since a node always covers positions if it has a missing dimension, as stated in Section 4.3.1. In this case, it covers positions along the missing dimensions, and it thus connects to nodes which should have a neighbor on one of the covered positions. If all other dimensions of such a currently covering node are fully

loaded, a maximum number of nodes is present that the covering node has to connect to in its role as the covering node on the missing dimensions.

The number of links that the node has to maintain in such a case can be determined as follows: Let F be the node's set of missing dimensions, i.e., the set of dimensions along which the node covers a position. $F^+(dim)$ denotes the set of missing dimensions higher than some dimension dim, and $F^-(dim)$ the set of missing dimensions lower than dim. Conversely, $\overline{F}, \overline{F^+(dim)}, \overline{F^-(dim)}$ denote the set of non-missing dimensions, i.e., dimensions on which the node does have a neighbor and does not have to cover any position. Counting the number of links to be maintained by the node simply encompasses counting the number of positions covered and the number of links to be maintained at each of these positions.

Along dimension dim, the node covers $2^{|F^+(dim)|}$ positions: The number of positions covered along dimension dim is determined by the number of "don't cares" in the node's cover map up to this dimension, as described above. At each such covered position, only links to nodes along lower and existing dimensions will be counted as links to be maintained: Along missing dimensions, the node covers the neighbor position by itself, thus not having to maintain a link. And higher-dimensional neighbors (i.e., $d > dim$) of a position along dimension dim have already been counted before since dim is counted from d_{max} up to 0. The counting hence yields

$$\#links = \sum_{\forall d \in F} \#pos(d) \cdot \overline{F^-(d)} = \sum_{\forall d \in F} 2^{|F^+(d)|} \cdot |\overline{F^-(d)}| \tag{4.18}$$

Worst case. In the worst case, a single node in the network is missing all dimensions $1 \leq d \leq d_{max}$ while all other nodes do not miss any neighbor except for their neighbor along dimension 0. Here, a single node has taken over the coverage responsibility for an entire cluster of maximum size, thus covering $2^{d_{max}-1}$ positions. At each position, it maintains exactly one 0-dimensional link. In this case, the node has to maintain $2^{d_{max}-1}$ links.

The number of links that a node maintains determines the number of messages to be sent if the node integrates a new node: Then, the new node will be assigned a subset of the covered positions, and all prospective neighbors of the new node have to be informed. In the balanced case, this yields a message complexity of $O(\log_2 N)$, in the worst case, $O(2^{d_{max}-2})$ will be sent since half of the node's covered positions would be passed on to a new node.

4.3.2.2 Characteristic Path Length

The characteristic path length of a graph is the average over all shortest paths between any two nodes in the graph. For a complete hypercube topology of base b, it can be derived as follows: Starting from one node in the topology, sum up the number of steps of the shortest paths leading to all other nodes, then divide the number of steps by the number of nodes

P	0	1	2	3	...	$log_b N$
$1 \cdot (\sum_{i=1}^{log_b N}(b-1))$	$b-1$	$b-1$	$b-1$	$b-1$	...	$b-1$
$2 \cdot (\sum_{i=1}^{log_b N-1}(b-1)^2 \cdot \sum_{j=1}^{i} 1)$	$(b-1)^2 \cdot \sum_{i=1}^{log_b N-1} 1$	$(b-1)^2 \cdot \sum_{i=1}^{3} 1$	$(b-1)^2 \cdot \sum_{i=1}^{2} 1$	$(b-1)^2 \cdot \sum_{i=1}^{1} 1$	...	0
$3 \cdot (\sum_{i=1}^{log_b N-2}(b-1)^3 \cdot \sum_{j=1}^{i} \sum_{k=1}^{j} 1)$	$(b-1)^3 \cdot \sum_{i=1}^{log_b N-2} \sum_{j=1}^{i} 1$	$(b-1)^3 \cdot \sum_{i=1}^{2} \sum_{j=1}^{i} 1$	$(b-1)^3 \cdot \sum_{1}^{1} \sum_{j=1}^{i} 1$	0	...	0
$4 \cdot (\sum_{i=1}^{log_b N-3}(b-1)^4 \cdot \sum_{j=1}^{i} \sum_{k=1}^{j} \sum_{l=1}^{k} 1)$	$(b-1)^4 \cdot \sum_{i=1}^{log_b N-3} \sum_{j=1}^{i} \sum_{k=1}^{j} 1$	$(b-1)^4 \cdot \sum_{i=1}^{1} \sum_{j=1}^{i} \sum_{k=1}^{j} 1$	0	0	...	0
...	...	...	...	...	...	...
$log_b N \cdot (\sum_{i=1}^{1}(b-1)^{log_b N} \cdot \sum_{j=1}^{i} {}^{(log_b N-1)} 1$	$(b-1)^{log_b N} \cdot \sum_{i=1}^{1} {}^{(log_b N-1)} 1$	0	0	0	...	0

Figure 4.3: Table for characteristic path length computation

reached, $N-1$. Since the topology is symmetric, it is sufficient to analyze this algorithm for only one node in the network, and for any arbitrary node. To compute the path to every single node from a starting node, the broadcast algorithm described above is used. In Figure 4.3.2.2, the process of reaching every single node in the network starting at a fixed node is divided into two dimensions: Along the "x-axis" of the table, the starting dimension, i.e., the dimension of the link that is traversed in the first hop away from the starting node is depicted. Along the "y-axis" of the table, the steps are counted: In total, there will be at maximum $\log_b N$ hops to reach the most distant node from the starting node. Clearly, this final step is located at the position $(x, y) = (0, \log_b N)$ in the table since only if the first link traversed is of the lowest possible dimension (0) can the maximum path ($\log_b N$) be walked.

The table counts the number of nodes that can be reached when leaving the initial node via a dimension *column* link and walking for *row* hops. For example, when leaving the starting node, there are $b-1$ possible links to be used along every dimension: The starting node has $b-1$ neighbors along each dimension, and all neighbors have not been visited yet. However, when walking along dimension $\log_b N$ and reaching the nodes at the end of the links, there will be no legal out-link that the broadcast algorithm permits the walker to use from any of these nodes - since the maximum dimension has already been traversed. Thus, all table elements below element $(x, y) = (log_b N, 0)$ are 0.

Since the elements in the table count the number of nodes reachable by traversing links along a particular dimension i in the j-th step of the broadcast process, its elements m_{ij} have to be multiplied with j (the number of steps walked in the broadcast process to get to the elements in row j in the table) and summed up. For every row, this sum is computed in the leftmost column of the table. All such row sums are summed up and then finally divided by $N-1$, yielding the characteristic path length.

In these equations, the following notation denotes a recursive sum:

$$\sum_{i=1}^{N} {}^{(M)} = \sum_{x_1=1}^{N} \sum_{x_2=1}^{x_1} \sum_{x_3=1}^{x_2} \cdots \sum_{x_M=1}^{x_{M-1}} 1 \tag{4.19}$$

Consider the following example to understand the appearance of this recursive sum. Say one has arrived at a node via a dimension 1 link, and there are four dimensions in total in the graph. Thus, the node may be left via dimension 2 and 3 links. If it is left via a dimension 2

Total Path Length	**0**	**1**	**2**
$1 \cdot 3$	1	1	1
$2 \cdot 3$	2	1	0
$3 \cdot 1$	1	0	0

Table 4.1: Calculation of total path length in a three-dimensional hypercube

link (there are $b-1$ nodes connected to the current node via a dimension 2 link), one would reach nodes which are permitted to be left via either dimension 3 or 4 links. If, however, the node is left via a dimension 3 link (again, there are $b-1$ nodes connected to the current node via a dimension 3 link), one would reach nodes which are permitted to be left only via dimension 4 links. Even more restricted, if the node is left via a dimension 4 link, one would arrive at nodes which would not be left any more at all. The recursive sum counts exactly the number of options one has *in total* if one arrives at a node via a link of a certain dimension. The lower this dimension, the deeper nested the recursive sum.

Summing up the path lengths in the different steps yields

$$\sum_{i=1}^{\log_b N} p_i = \sum_{i=1}^{\log_b N} (\log_b N - i + 1) \cdot (b-1)^{\log_b N - i + 1} \cdot \sum_{j=1}^{i} \sum_{k=1}^{(\log_b N - i)} 1 \tag{4.20}$$

Or

$$\sum_{i=1}^{\log_b N} p_i = \sum_{i=1}^{\log_b N} (\log_b N - i + 1) \cdot (b-1)^{\log_b N - i + 1} \cdot \sum_{j=1}^{i} 1 \tag{4.21}$$

To simplify this, use the identity

$$\sum_{j=1}^{i} {}^{(a)} 1 = \frac{1}{a!} \cdot \prod_{j=0}^{a-1} (i+j) \tag{4.22}$$

This leads to the final result

$$\#paths(b, N) = \sum_{i=1}^{\log_b N} (b-1)^{\log_b N - i + 1} \cdot \frac{1}{(\log_b N - i)!} \cdot \prod_{j=0}^{\log_b N - i} (i+j) \tag{4.23}$$

Thus, the characteristic path length in a hypercube of base b with N nodes is

$$L = \frac{\#paths(b, N)}{N-1} = \frac{1}{N-1} \cdot \sum_{i=1}^{\log_b N} \frac{(b-1)^{\log_b N - i + 1}}{(\log_b N - i)!} \cdot \prod_{j=0}^{\log_b N - i} (i+j) \tag{4.24}$$

For large N, this converges to $0.5 \cdot \log_b N$.

As an example, consider the three-dimensional hypercube (Figure 4.1): Starting from node 0, try to reach every node with the smallest number of hops possible. This yields Table 4.1.

Walking along dimension 0 from node 0 leads to node 4, walking along dimension 1 leads to node 2 and walking along dimension 2 leads to node 1 – captured by the first row of the

table. Now, having arrived at node 4, the broadcast algorithm 4.2.2 permits to proceed to nodes 6 and 5, along dimension 1 and 2, respectively. Hence the path that had started by walking along dimension 0 from node 0 in the first step branches and yields two possibilities of proceeding. Proceeding via the dimension 2 link leads to node 5 – and the path stops there: The maximum dimension has been traversed, the algorithm does not allow for any further movement. Proceeding via the dimension 1 link leads to node 6. From node 6, one further step is allowed, finally leading to node 7. All these movements, starting via a dimension 0 link from node 0 in the first step, are captured by column 0 in Table 4.1.

Consider column 1 in the table: Having arrived at node 1 via the path that had started by walking along dimension 1 from node 0 in the first step, one may proceed to node 3 only.

Finally, column 2 represents an even shorter path: Having arrived at node 1 via a dimension 2 link in the first step, the algorithm does not allow for any further movement along this path since the maximum dimension has been traversed.

In total, the movement yields 3 paths of length 1 (from node 0 to nodes 1, 2 and 4), 3 paths of length 2 (from node 0 to nodes 3, 5 and 6) and 1 path of length 3 (from node 0 to the most distant node, node 7). Hence the sum of all path lengths from node 0 to any node in the graph is 12, and the characteristic path length for the three-dimensional hypercube is $\frac{12}{7} \approx 1.71$.

4.3.3 Randomization

In Section 4.3.2, it was shown that the complexity of the HyperCuP protocol depends on the balance of the graph topology. However, if the protocol was carried out as presented in Section 4.3.1, the graph would probably become imbalanced very quickly: Freely evolving P2P networks tend to exhibit a particular topology which is governed by several so-called power-laws [24] [16].

4.3.3.1 Power-Law Graphs

Basically, a power-law network topology consists of a few nodes with high node degree and many nodes with low node degree. More formally, four power-laws can be identified [5]:

1. The out-degree d_i of a node i is proportional to the rank of the node r_i to the power of a constant: $d_i \sim r_i^{\alpha}$. The rank of a node here is given by a node's out-degree, i.e., the node with the largest number of links has rank 1. (Note that in the networks considered here only indirected links are considered, i.e., in-degree equals out-degree equals node degree in general.)

2. The frequency f_d of an out-degree d is proportional to the out-degree to the power of a constant: $f_d \sim d^{\beta}$.

3. The total number of pairs of nodes $P(h)$ within h hops is proportional to the number of hops h to the power of a constant: $P(h) \sim h^{\gamma}$.

4. The eigenvalues λ_i of a graph are proportional to the order i of the respective eigenvalue to the power of a constant: $\lambda_i \sim i^{\delta}$.

A graph topology that obeys these power-laws can be created as described in [5] by following two simple rules:

1. Nodes join a network iteratively, i.e., an initial population of nodes is joined by more and more nodes over time.

2. When a node joins a network, it connects to one or more nodes already in the network with probability $\frac{d_i}{\sum_{j=0..N-1} d_j}$ where N is the number of nodes currently in the network and d_i is the degree of node i.

These policies construct a network that obeys the law of 'the rich getting richer': Nodes with relatively high node degree become the point of contact of many new nodes joining, thus building a topology in which only a few nodes rise to be highly connected.

Such a network will be simulated in Section 4.4. For now, these considerations point to one problem: If real-world P2P networks exhibit a power-law 'behavior' – i.e., a few nodes are preferably chosen as integration champion nodes by new nodes joining the network – the network will become imbalanced since a few nodes will keep extending the dimensionality of the hypercube to integrate more and more nodes as their immediate neighbors. Hence a balancing scheme has to be implemented.

4.3.3.2 Random Walks on Graphs

The balancing scheme used in HyperCuP tries to distribute node joins equally over the network, i.e., it aims at ensuring that the probability of any node in the network to become the initial integration champion node is virtually equal. This property is provided by a random walk on a graph G: If any graph G is a random graph, the stationary probability distribution of a random walk over all nodes quickly converges to a uniform distribution. More formally:

$$\|A\vec{\pi} - \vec{\mu}\| \leq \lambda_2(G) \cdot \|\vec{\pi} - \vec{\mu}\| \tag{4.25}$$

Here, $\vec{\pi}$ is an initial probability distribution over the vertices of a graph$G$, A is the normalized adjacency matrix of G and $\vec{\mu}$ is a uniform distribution over the vertices of G. In [47], it is proven that the above equation determines the convergence of a random walk on a graph$G$: If a random walk is started on any node of G with any stationary probability distribution $\vec{\mu}$ and takes random hops on the graph, the probability distribution $\vec{\mu}$ will converge to a

uniform distribution with a speed determined by the value of the second largest eigenvalue of the adjacency matrix A of graph G. The higher $\lambda_2(G)$, the faster the convergence. (In the equation, a hop is denoted by the multiplication of A and $\vec{\pi}$: The current probability distribution over all nodes in G, $\vec{\pi}$, is modified by jumping to a node connected to the current node with uniform probability $\frac{1}{nodedegree}$, drawn from A. The equation states that the new probability distribution, $A\vec{\pi}$, is closer to uniform than the old distribution before the hop,$\vec{\pi}$.)

Note that if $\vec{\mu}$ has converged to a uniform distribution, there is an equal probability for being at any node in the graph G at the end of a random walk.

The first eigenvalue of a complete hypercube graph is $d = \log_b N$ since the hypercube graph is a regular graph with a node degree of d. Unfortunately, the second largest eigenvalue of a hypercube graph is -1 since the hypercube graph is bipartite. Intuitively, a random walk will never converge to a uniform distribution - which is easily understandable: Since the graph is bipartite, the random walker will always end up at one half of all nodes, even if it walks on for an infinite number of steps. The graphs HyperCuP deals with are not symmetric most of the times, hence this conclusion may hold only partly in most cases – however, a better randomization scheme would be desirable. An approach will be presented in the following section.

4.3.3.3 Randomization in HyperCuP

The balancing algorithm looks as follows: If a new node attempts to join the network, it contacts any node V already integrated into the topology. This node, instead of becoming the integration champion node right away, picks a random position on the hypercube,$\vec{p}_{dest}$, as a binary number with d_{max} digits, the maximum graph dimension. The network then carries out shortest-path routing to the destination position $\vec{p}_{dest}$.

Thus, every position on the hypercube has equal probability of being chosen as destination position. The node at the final message destination $\vec{p}_{dest}$ then is selected to become the integration champion node. This procedure adds a maximum delay of $\log_2 N$ hops to the integration procedure since this is the maximum path that can be travelled in shortest-path routing. Nodes which currently cover more positions than other nodes of course have a higher probability of being chosen as final integration champion – however, this is actually an advantage since these nodes occupy a larger number of vacant positions to be passed on to newly joining nodes.

The average path travelled can be shortened by not routing the integration message to its final destination, but by picking any node touched by the message during the routing

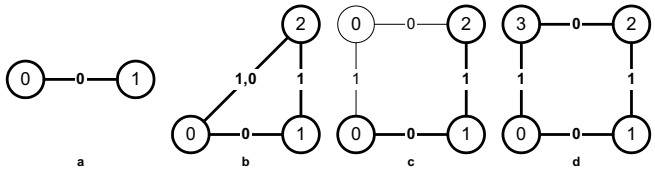

Figure 4.4: Network Topology Construction

process as integration champion with a certain probability: The probability that a node$V$ that receives an integration message becomes the integration champion is set to

$$p = 1 - \frac{\|\vec{p}_v \oplus \vec{p}_{dest}\|}{\|\vec{p}_{src} \oplus \vec{p}_{dest}\|} \tag{4.26}$$

where $\vec{p}_{src}$ corresponds to the position of the node that initially issued the integration message (i.e., the node that was contacted by a newly joining node in the first place),$\vec{p}_{dest}$ is the random hypercube position which was picked by this node and$\|\cdot\|$ again denotes the Hamming distance of binary vectors. Thus, the probability that a node which receives an integration message becomes the integration champion depends on the length of the path that the message has travelled so far. The scheme combines the randomization properties of the simple random position selection with a shorter average path, therefore reducing the communications overhead of the randomization procedure. Simulation results for this scheme are presented in Section 4.4.

4.3.4 Topology Construction Example

This section will walk through a HyperCuP topology construction example. During the example, the maximum number of dimensions is set to 3. All position vectors and cover maps of nodes thus consist of three elements. For each step, the position vectors and cover maps of each node are spelled out and depicted, too. The randomization procedure is switched off for ease of presentation – the node that is contacted first by a new node becomes its integration control champion.

Start. At the beginning, only peer 0 is active. The first peer in a network occupies the position 000, and it covers all positions in the three-dimensional hypercube, thus its cover map consists of only 1s.

$$\begin{aligned} \vec{p}_0 &= 000 \\ \vec{c}_0 &= 111 \end{aligned} \tag{4.27}$$

Step a. Peer 0 is contacted by node 1 which wants to join the P2P network. Peer 0 integrates peer 1 as 0-neighbor: The peers establish a link between each other which is tagged with the

neighbor set $\{0\}$, as depicted in Figure 4.4. Generally, a peer integrates a joining peer along its first vacant dimension, the dimensions are ordered such that lower dimensions always come first. Both peers currently cover 4 positions in the three-dimensional hypercube.

$$\begin{aligned} \vec{p}_0 &= 000 \quad \vec{p}_1 = 100 \\ \vec{c}_0 &= 011 \quad \vec{c}_1 = 011 \end{aligned} \tag{4.28}$$

Step b. Peer 2 contacts one of the two peers (here, assume that it contacts peer 1) to join the network. The first vacant dimension of peer 1 is 1 since it already maintains a 0-neighbor, peer 0. Essentially, peer 1 opens up a new dimension for the hypercube, as depicted in Figure 4.4b. Peer 1 becomes the integration champion node for the complete integration of peer 2 into the network: It is responsible for providing peer 2 with all necessary links – at the end of the integration process, peer 2 has to have neighbor links connecting it all currently existing dimensions, in order to be able to carry out complete broadcasts. Since peer 1 currently has two neighbors, a 0- and a 1-neighbor, it knows that it has to provide peer 2 with a 0- and a 1-neighbor, too. Peer 1 itself has become peer 2's 1-neighbor. Peer 1 selects peer 0 as the new 0-neighbor for peer 2. However, peer 0 can only become a temporary 0-neighbor for peer 2 because it already has another 0-neighbor, namely peer 1 – and it has been said before that a peer can only have one neighbor per dimension. Essentially, peer 0 now covers a vacant position in the hypercube, i.e., it acts as if it occupies two positions in the hypercube, as depicted by the thin copy of peer 0 in Figure 2c. To mark the link between peers 2 and 0 as temporary relationship, it is tagged with the link set $\{0, 1\}$ (instead of $\{0\}$) in the figure: This link set denotes the path from peer 2 via the position at which the link set is originally aiming to peer 0, the peer which currently covers this position. (This path is also well visible in Figure 4.4c.) Note that link sets are actually not stored by any node in the HyperCuP algorithm, they can always be derived by comparing the position vectors of two nodes.

$$\begin{aligned} \vec{p}_0 &= 000 \quad \vec{p}_1 = 100 \quad \vec{p}_2 = 110 \\ \vec{c}_0 &= 011 \quad \vec{c}_1 = 001 \quad \vec{c}_2 = 001 \end{aligned} \tag{4.29}$$

Step c. Peer 3 wants to join the network. Three cases are compared, viz. peer 3 contacting peer 0, 1 or 2 to join the network. In case peer 3 contacts peer 0 to join, peer 0 follows the general rule to integrate the peer in its first vacant dimension – which is 1, since peer 0 has a 0-neighbor, but no 1-neighbor. As its new 1-neighbor, peer 3 will now cover the temporary position that peer 0 used to maintain in the hypercube: Hence peer 0 can pass on links that are associated with this position to peer 3. In checking its cover map, peer 0 is able to determine that link $\{0, 1\}$ to peer 2 is a link that is to be passed on to peer 3. Peer 3 then establishes a link tagged by link set $\{0\}$ to peer 3, as depicted in Figure 4.4d. In case peer 3 contacts peer 2 to join, peer 2 decides to integrate peer 3 as its new (and non-temporary) 0-neighbor.

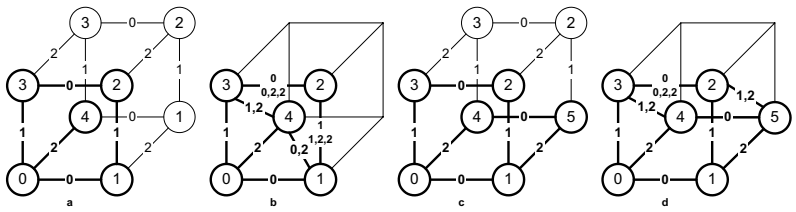

Figure 4.5: Network topology construction continued

However, it does not carry out the integration itself: Since peer 0 currently covers the position that will soon be occupied by peer 3, the integration control responsibility has to be forwarded to peer 0. Peer 2 can do so via peer 0.

Peer 0 carries out the integration just as described above, arriving at Figure 4.4d. In case peer 3 contacts peer 1, peer 1 will integrate peer 3 in dimension 2, i.e., it opens up a new dimension for the hypercube. This leads to a momentary imbalance in the hypercube with some peers maintaining more links than others. To preserve network balance, the randomization technique described in Section 4.3.3 is used in a real implementation of the HyperCuP protocol.

$$\begin{array}{llll} \vec{p}_0 = 000 & \vec{p}_1 = 100 & \vec{p}_2 = 110 & \vec{p}_3 = 010 \\ \vec{c}_0 = 001 & \vec{c}_1 = 001 & \vec{c}_2 = 001 & \vec{c}_3 = 001 \end{array} \tag{4.30}$$

Step d. Peer 4 arrives and contacts peer 0. Now, the network crosses a threshold – a hypercube with 2 dimensions cannot accommodate 5 peers, hence a third dimension is opened up. Peer 0 integrates peer 4 in its first vacant dimension as its new 2-neighbor. Peer 4 needs 3 neighbors, one in each dimension – but neither peer 0's 1-neighbor, peer 3, nor peer 0's 0-neighbor, peer 1, are linked to their own 2-neighbor which they could provide as a new neighbor to peer 4. Thus, peer 3 acts as temporary 1-neighbor for peer 4, whereas peer 1 acts as temporary 0-neighbor for peer 4, indicated once again by the link sets $\{0,2\}$ and $\{1,2\}$ among these peers (see Figure 4.5b) and peer 1's cover map. Figure 4.5a shows the existing peers in the network in bold style and the positions that are additionally covered by them in thin style. Once again, note that the positions that are additionally covered by peers determine the temporary connections the peers have to maintain. Figure 4.5a also demonstrates another basic rule: Peers that are closest to a vacant position in the hypercube structure are always chosen to cover it. Here, closest means that the peer in the highest dimension to a vacant position covers it. Among the other peers in the network, adding another dimension to the graph means the multiplication of existing links, too: For example, peers 1 and 2 could now both integrate 2-neighbors, which would then be linked in dimension 1. Thus, the already existing $\{1\}$ link is tagged additionally as $\{1,2,2\}$ link. (Of course, additional links are

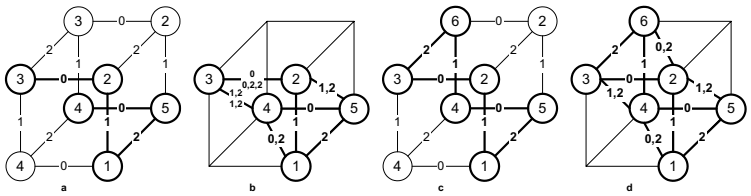

Figure 4.6: Network topology construction continued

multiplied only virtually – two already connected peers do not open up an additional link.) So is the already existing $\{0\}$ link between peers 2 and 3.

$$\begin{array}{lllll} p_0 = 000 & p_1 = 100 & p_2 = 110 & p_3 = 010 & p_4 = 001 \\ c_0 = 000 & c_1 = 001 & c_2 = 001 & c_3 = 001 & c_4 = 000 \end{array} \tag{4.31}$$

Step e. Peer 1 is contacted to integrate the newly arriving peer 5. Peer 1 is still lacking a 2-neighbor, thus peer 5 will be integrated on this position (Figure 4.5d). Now, peer 1 can get rid of its $\{1, 2, 2\}$ link to peer 2: It is moved to peer 5. However, since 2 is not peer 5's final 1-neighbor either, the link stays temporary: Peers 2 and 5 now maintain a$\{1, 2\}$ link among them. Peer 5 takes over peer 1's temporary $\{0, 2\}$ link to peer 4, which still lacks its final 0-neighbor. It has found one now, namely peer 5.

$$\begin{array}{llllll} \vec{p}_0 = 000 & \vec{p}_1 = 100 & \vec{p}_2 = 110 & \vec{p}_3 = 010 & \vec{p}_4 = 001 & \vec{p}_5 = 101 \\ \vec{c}_0 = 000 & \vec{c}_1 = 000 & \vec{c}_2 = 001 & \vec{c}_3 = 001 & \vec{c}_4 = 000 & \vec{c}_5 = 000 \end{array} \tag{4.32}$$

Step f. Assume that peer 0 suddenly leaves the network. In the maintenance protocol, it is obliged to carry out the departure protocol: It decides which existing peers that it knows will be chosen to take over responsibility for the positions it gives up. In the example, peer 0 leaves only one position vacant, its original position in the graph – however, a node which covers multiple positions will have to find successors for each of its positions in the graph. Peer 4 takes over peer 0's position due to its closeness to peer 0's position, establishing temporary links to the former neighbors of peer 0, peers 1 and 3. Figure 4.6a shows the new distribution of covering responsibilities, Figure 4.6b depicts the link structure that arises from this network state.

$$\begin{array}{llllll} \vec{p}_1 = 100 & \vec{p}_2 = 110 & \vec{p}_3 = 010 & \vec{p}_4 = 001 & \vec{p}_5 = 101 \\ \vec{c}_1 = 000 & \vec{c}_2 = 001 & \vec{c}_3 = 001 & \vec{c}_4 = 001 & \vec{c}_5 = 000 \end{array} \tag{4.33}$$

Step g. Peer 4 is contacted by peer 6 and decides to integrate it as its new 1-neighbor. This position is currently covered by peer 3, hence peer 4 forwards the integration control to peer 3, just as described in step c. In the example, all temporary links that are currently owned

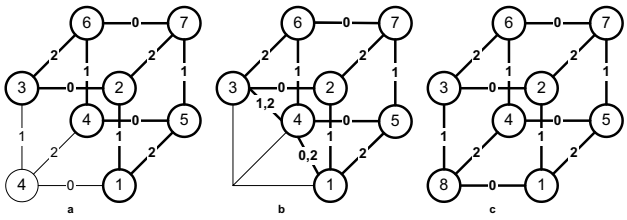

Figure 4.7: Network topology construction continued

by peer 3, but originally belong to the new position of peer 6, are restored and passed on to peer 6. Additionally, peer 3 integrates peer 6 as its new 2-neighbor, arriving at Figure 4.6d.

$$\begin{array}{llllll} \vec{p}_1 = 100 & \vec{p}_2 = 110 & \vec{p}_3 = 010 & \vec{p}_4 = 001 & \vec{p}_5 = 101 & \vec{p}_6 = 011 \\ \vec{c}_1 = 000 & \vec{c}_2 = 001 & \vec{c}_3 = 000 & \vec{c}_4 = 001 & \vec{c}_5 = 000 & \vec{c}_6 = 000 \end{array} \tag{4.34}$$

Step h. Peer 6 is contacted by peer 7, leading to peer 7's integration as peer 6's new 0-neighbor. Figure 4.7a and Figure 4.7b depict the state of the network: Almost all positions of a complete hypercube graph with 3 dimensions are held by active peers, only peer 4 still covers two positions in the hypercube.

$$\begin{array}{lllllll} \vec{p}_1 = 100 & \vec{p}_2 = 110 & \vec{p}_3 = 010 & \vec{p}_4 = 001 & \vec{p}_5 = 101 & \vec{p}_6 = 011 & \vec{p}_7 = 111 \\ \vec{c}_1 = 000 & \vec{c}_2 = 000 & \vec{c}_3 = 000 & \vec{c}_4 = 001 & \vec{c}_5 = 000 & \vec{c}_6 = 000 & \vec{c}_7 = 000 \end{array} \tag{4.35}$$

Step i. On integrating peer 8, peer 4 pushes its links $\{1, 2\}$ and $\{0, 2\}$ to its new 2-neighbor, arriving at a complete hypercube topology again.

$$\begin{array}{llllllll} \vec{p}_1 = 100 & \vec{p}_2 = 110 & \vec{p}_3 = 010 & \vec{p}_4 = 001 & \vec{p}_5 = 101 & \vec{p}_6 = 011 & \vec{p}_7 = 111 & \vec{p}_8 = 000 \\ \vec{c}_1 = 000 & \vec{c}_2 = 000 & \vec{c}_3 = 000 & \vec{c}_4 = 000 & \vec{c}_5 = 000 & \vec{c}_6 = 000 & \vec{c}_7 = 000 & \vec{c}_8 = 000 \end{array} \tag{4.36}$$

Link failures. A link failure in the network leads to a node's immediate departure from the P2P topology, not being able to send any departing messages. If that happens, the topology must be able to recover and head back to a normal state. In the hypercube graph, the topology can always recover from a sudden node loss. The node that is closest to the vanished node contacts the vanishing node's neighbors by asking its own neighbors for them. The node then carries out the node departure routine on behalf of the vanished node.

This algorithm is not able to deal with simultaneous node failures. Dealing with simultaneous node failures is a current research topic also in other topology construction protocol developments such as [39] and [43], and further research is required to make P2P topology construction protocols less vulnerable to unexpected node failures.

4.4 Simulation

To simulate the algorithm, an experiment is set up in which nodes join and depart from a network using the HyperCuP protocol.

4.4.1 Network Model

The simulation uses a simplified instance of the P2P network model described in Chapter 6: Here, peers do not issue any queries, hence only the considerations in Section 6.4 and Section 6.5 are used.

In short, the simulation proceeds as follows: In each time unit, a number of nodes joins and leaves the network. The HyperCuP protocol is carried out by all nodes to always construct a hypercube topology, as described above. Then, the characteristic path length and average node degree in the network are measured.

4.4.2 Simulation Results

Figure 4.8 depicts the average node degree in a network that is scaled up from 1000 to 10000 nodes. Obviously, the protocol is not able to build the most dense hypercube topology at all times: The actual average node degree is always slightly higher than the smallest possible average node degree. The smallest possible average node degree is $\lceil \log_2 N \rceil$ in a network of N nodes. Figure 4.8 shows that the HyperCuP protocol manages to keep the average node degree within a constant of the minimum average node degree which is an important achievement.

Figure 4.9 depicts the characteristic path length of a network that is scaled from 1000 to 4000 nodes. The characteristic path length is calculated as the expected path length between any two nodes on the network. To compute it in an efficient manner, two nodes are randomly picked and the shortest path between these nodes is searched. This process is repeated several times, and the results are averaged. The characteristic path length on the network may even be lower than the minimum path length in a hypercube of dimension $\lceil \log_2 N \rceil$: This is due to the fact that the average node degree in the network is actually always higher than the minimum node degree in a hypercube of the specified dimension. Thus, more links exist which provide shortcuts in the network, which in turn decrease the characteristic path length.

4.5 Cayley Graphs and HyperCuP

The binary hypercube is a representative of a larger class of graphs, the so-called Cayley graphs [2]. The HyperCuP algorithm exploits certain properties of the hypercube graph which turn out to be properties of the class of Cayley graphs. This section shows that the

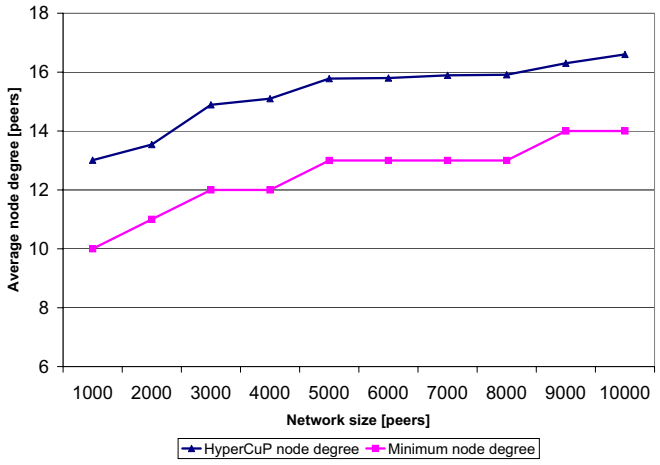

Figure 4.8: Average node degree in HyperCuP simulations

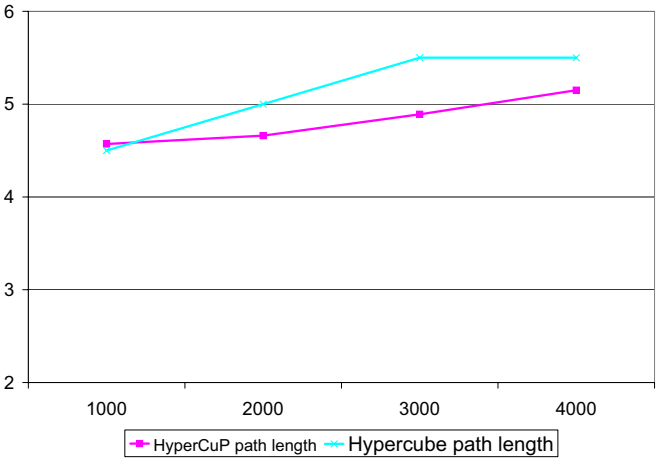

Figure 4.9: Characteristic Path Length in HyperCuP simulations

algorithm can be used to build Cayley graph topologies instead of hypercubes, and why this forms an interesting extension to the algorithm.

4.5.1 Introducing Cayley Graphs

A Cayley graph is the graphical interpretation of a group. A group is a simple mathematical entity: It contains a number of elements, and these elements can all be generated from each other by a number of generators. A group contains *all* elements that can be generated from some starting element by applying the generators in any arbitrary sequence.

The simplest way of regarding a group as a geometric object is through its Cayley Graph: Consider a finitely generated group G with generators $s_1, s_2, \ldots, s_n$. This just means that every element of G can be expressed as a product of the s_i generators with the starting

element s_1. The Cayley graph of G is a graph whose vertices are precisely the elements of G, and whose edges are described by the rule that for each pair of elements$x, y \in G$ there is an edge labeled by the generator s_i which originates at x and terminates at y provided that $x * s_i = y$. Section 4.5.2 will demonstrate an example of a Cayley graph.

A permutation group is a group whose elements are strings of symbols and whose generators permute the ordering of symbols within an element. For example, a generator 321 applied to a group element 231 yields another group element, 132. The permutation generator specifies the new positions of symbols in an input symbol string and thus produces an output symbol string. For example, the generator 321 reads: Take a three-symbol string$a_1a_2a_3$ as input and re-order the symbols to $a_3a_2a_1$.

The binary hypercube is a Cayley graph: The generators for a three-dimensional hypercube are

$$213456 - 124356 - 123465 \tag{4.37}$$

and the starting element is 123456. A simple mapping rule to binary coordinates can be given: Recall that a three-dimensional hypercube is characterized by 3-bit long position vectors for its nodes. Such a 3-bit position vector is derived from a 6-digit symbol string such as 124356 by dividing it into groups of two symbols each and interpreting each symbol group as a bit. The corresponding bit is not set if the elements of the group are ordered, and it is set if the elements of the group are not ordered. Thus, element 123456 denotes the binary address

$$123456 = 12|34|56 = 0|0|0 = 000 \tag{4.38}$$

Similarly:

$$213465 = 21|34|65 = 1|0|1 = 101 \tag{4.39}$$

Thus, each of the three operators of the binary hypercube Cayley graph corresponds to flipping a bit in a binary position vector, thus building a three-dimensional hypercube from starting element 123456 = 000.

Cayley graphs have a number of interesting properties [2]. Furthermore and most importantly, some Cayley graph are hierarchical, i.e., a Cayley graph of dimension$n$ can be decomposed into $n-1$ dimensional Cayley graphs generated by the same generators. A Cayley graph is hierarchical if its generators are all transpositions: Each generator exchanges the place of exactly two symbols in a symbol string. Note that the hypercube is a hierarchical Cayley graph since this property holds for its generators.

4.5.2 The Star Graph

The star graph was described in [2] as an alternative to the hypercube. Figure 4.10 depicts a four-dimensional star graph. Its generators are simple: Each generator swaps the first element

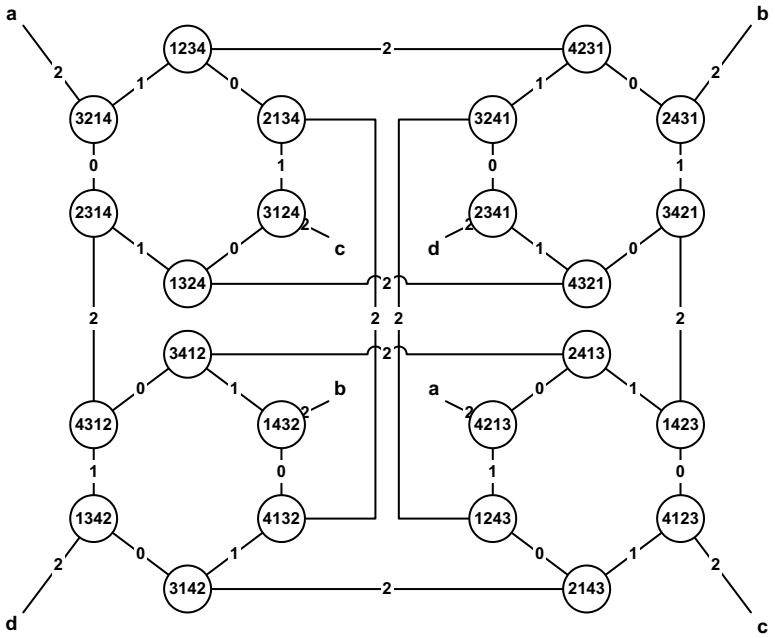

Figure 4.10: The Star Graph

in a symbol string with one of the other elements in the string. Thus, the generators building a four-dimensional star graph are

$$2134 - 3214 - 4231 \tag{4.40}$$

These generators are transposition generators, hence the star graph is a hierarchical Cayley graph. The star graph features virtually identical properties as the hypercube. An optimal broadcast algorithm can be defined [2], shortest-path routing is possible, the topology is fault-tolerant and symmetric. Yet, while the hypercube has logarithmic node degree and network diameter, the star graph features sub-logarithmic node degree and network diameter. In a d-dimensional star graph, there are $d!$ nodes, the node degree is $d-1$ and the network diameter is $\Delta = \lceil \frac{3}{2} \cdot (d-1) \rceil$.

4.5.3 HyperCuP Extension

The HyperCuP protocol is capable of building transposition-based Cayley graph topologies with small modifications to the protocol.

First, there has to be an ordering on the generators to introduce a notion of distance into the graph topology. In the binary hypercube, an ordering on the dimensions was assumed, and neighbors along a higher dimension were closer to a node than neighbors along a lower dimension. The same has to hold for building other Cayley graph topologies. For example,

node 4231 in the star graph is farther away from node 1234 (since the dimension 2 generator 4231 was applied) than node 2134 (since the dimension 0 generator 2134 was applied). (The ordering on the dimensions is reversed as compared to the hypercube.)

Second, cover maps of nodes are no binary vectors any more. In the binary hypercube, there used to be exactly two large clusters on each dimension (the cluster consisting of nodes where the corresponding dimensional address bit is set to 0, and the cluster of those nodes where this bit is set to 1). In the general case, there may be more such clusters per dimension. For example, in the star graph, there are four 2-dimensional clusters, each consisting of nodes where the last address symbol is fixed to either 1, 2, 3 or 4 (the corresponding nodes' address is $***i$). In each 2-dimensional cluster, there are three 1-dimensional clusters, each of which has the penultimate address symbol fixed. For example, cluster ***1 contains clusters **21, **31 and **41.

Yet, since cover maps in the HyperCuP protocol are actually only a means of compressing the information on which positions are covered by a node, cover maps are abolished in order to build, e.g., the star graph. The node simply stores the positions it covers (and may apply any other data compression technique in order to reduce the memory size required for storing the positions).

As an example, if all nodes in cluster ***1 disappear, the covering for the cluster is assigned to the nodes in cluster ***2 first; and so on.

4.6 Implementation on JXTA

JXTA [45] is a set of open source protocols aiming at providing a network protocol- and language-independent P2P infrastructure. JXTA specifies protocols which provide basic P2P primitives, e.g., identifying peers and resources by unique identifiers, sending messages between peers, searching for resources etc. Bindings to programming languages such as Java and C/C++ exist.

The great advantage of JXTA's network protocol independence is that a JXTA-based P2P network can span several distinct transport networks: A JXTA peer located on a 3G wireless network can communicate with another JXTA peer on a TCP/IP based network such as the Internet. Using the P2P primitives defined by the JXTA protocols, developers of P2P applications can entirely abstract from the specifics of message transport on and among the underlying transport networks.

HyperCuP has been implemented as a Java-based extension to JXTA. In the following, the architecture of HyperCuP on JXTA shall be described, starting with a description of the JXTA architecture.

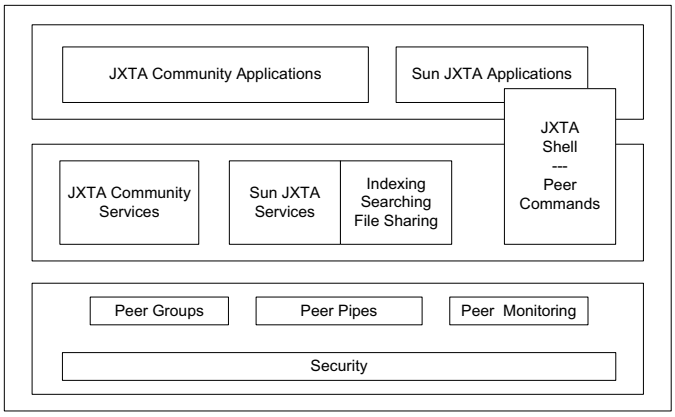

Figure 4.11: JXTA Layer Architecture

4.6.1 Introducing JXTA

The JXTA architecture is divided into three layers, depicted in Figure 4.11. At its core, JXTA allows for communication among peers on a transport network, without a central naming instance (such as the well-known Domain Name Service system, DNS), by associating peers with global unique identifiers and by exchanging information in the form of XML documents.

Platform Layer The platform layer, or JXTA core, contains primitives essential for P2P networking such as the definition of peer groups, routines for their creation and management, the definition of pipes, i.e., messaging channels between peers, and primitives for security of data and operations on a JXTA network. These primitives serve as core building blocks for services and applications defined on higher layers. They are provided by the basic set of JXTA protocols.

Services Layer This layer contains services that are common and desirable in P2P networks, such as searching, indexing and file sharing services.

Applications Layer P2P applications such as distributed knowledge management, P2P file systems etc. are associated with the applications layer.

Key JXTA concepts include peers, peer groups, advertisements and services. Any instance on a JXTA network which implements the basic JXTA protocols is a JXTA peer: It is able to communicate with other JXTA peers on the network. Also, JXTA peers can be organized into peer groups: Peer groups are logical groups that offer group-specific services, may have admission control and their own security procedures. Any peer on a JXTA network can create and join peer groups. JXTA peers are able to publish and read advertisements: Any resource on a JXTA network is described by an XML document, called an advertisement.

An advertisement contains a globally unique identifier for a resource and meta-data on its content. For example, a peer on the network is described by a peer advertisement, containing its name, global unique identifier plus additional meta-data. Advertisements are published by peers to describe the content they are storing, communication services they are offering etc. JXTA services are basically message handlers, installed at peers which support a particular service: For example, a peer group on a JXTA network may define a service by specifying the format of messages with which the service is invoked and used, and by installing a message handler at each peer which listens for incoming messages and processes all messages that are related to the service.

As an example, consider the JXTA discovery service: This service provides the functionality to discover resources on a JXTA network. It is one of the standard services which is implemented by most peers on a JXTA network. If a peer wants to search for a specific resource on the network, it may invoke the discovery service and specify meta-data that is used to discover the resource. The discovery service at the peer then issues a search request which is relayed to other peers in the network. Much as the broadcast search in the Gnutella network [21], the search message is thus propagated throughout the network. At each peer, the search message is compared with advertisements owned or cached locally by the peer - if an advertisement matches the search request, it is returned to the searching peer as a response to the request.

For a detailed introduction to JXTA, refer to [44].

4.6.2 HyperCuP on JXTA Architecture

The HyperCuP protocol has been implemented as a service on JXTA. More specifically, the HyperCuP implementation contains the definition of a HyperCuP peer group which supports the HyperCuP service. Instead of searching using the JXTA discovery service, a peer supporting the HyperCuP service is able to search by using the much more efficient HyperCuP search, as described in Section 4.2.2.

Any peer on a JXTA network is able to join the HyperCuP peer group by executing a join protocol which can be broken down to the following steps.

Discovering the HyperCuP peer group First, a peer that wants to use the HyperCuP service has to join the HyperCuP peer group. In order to do so, it requires an advertisement for the group. Hence as a first step, a peer uses the JXTA discovery service to discover an advertisement for the HyperCuP peer group.

Joining the HyperCuP peer group Having discovered the advertisement, the peer joins the HyperCuP peer group. Most notably, the advertisement contains group-specific classes which the peer can invoke. Thus, it is able to instantiate the HyperCuP service and execute its procedures.

Joining the HyperCuP topology In joining the HyperCuP peer group, a peer is assigned 'pending' status in the peer group: It is a member of the peer group in the JXTA sense, but it has not yet been assigned a position on the hypercube, hence it has not been integrated into the current topology. Now, the peer invokes the JXTA discovery service again in order to locate another peer in the HyperCuP peer group which has already been integrated into the topology. This peer then becomes the integration champion for the peer wishing to join the topology and carries out the HyperCuP integration protocol. Once completed, the peer's status is changed to 'integrated' - it is now a member of the HyperCuP peer group and fully integrated into the HyperCuP topology.

Using the HyperCuP service Now, the peer is able to issue search requests which are propagated on the HyperCuP topology using the efficient HyperCuP search mechanism. The peer is also able to handle the integration requests of newly arriving HyperCuP peers.

To always provide an entry point into the HyperCuP peer group, a HyperCuP rendezvous peer is always up and running on the JXTA network. It also periodically publishes an advertisement for the HyperCuP peer group and is integrated into the HyperCuP topology, hence is able to integrate newly arriving peers.

The code for the HyperCuP rendezvous peer and an example peer capable of joining the HyperCuP peer group and using the HyperCuP service is available in the Edutella CVS repository.

4.7 Finding Semantic Web Services

So far, this chapter has addressed a means of searching a P2P infrastructure in a scalable way. Yet, the term "searching" the network was used rather in the sense of "traversing" a network:

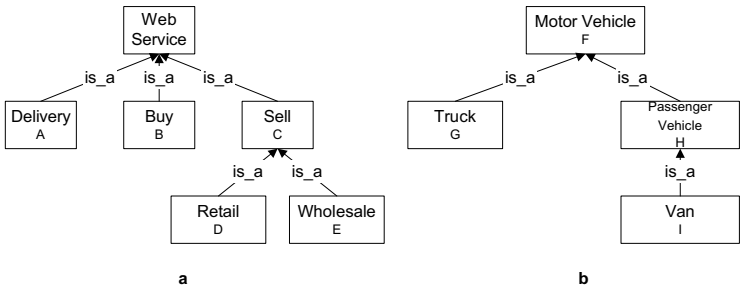

Figure 4.12: Service ontology and domain ontology

The propagation of messages through the network was made very efficient – yet once a query message is received by a peer, what does the peer actually do with it? In the domain of file-sharing [21], the peer checks if it is able to provide a file asked for by the query. In the world of Web Services, however, queries will be more complicated: They are trying to locate a Web Service that is capable of performing a specified task. Hence service queries have to be matched to Web Service descriptions.

4.7.1 Describing Web Services

Web Service descriptions on a P2P infrastructure face a major challenge: They have to be understood in a distributed and diverse world. Thus, in order to describe a Web Service, a common understanding of what a Web Service description may encompass is required. As introduced in Chapter 3, the Web Services Description Language WSDL provides a means of describing a service based on its operations and the input/output data types accepted. However, such descriptions will not be expressive enough to describe the real-world behavior of a Web Service.

4.7.1.1 Ontologies

Semantic Web research [11] has used ontologies to mark up and describe information on the Internet.

An ontology is a formal, explicit specification of a shared conceptualization [46]. A "conceptualization" refers to an abstract model of some phenomenon in the real world by identifying the relevant concepts of that phenomenon. "Explicit" denotes that the type of concepts used and the constraints on their use are explicitly defined and spelled out. "Formal" refers to the fact that the ontology should be machine understandable and processable. Note that this excludes natural language: Machines can only have a very limited understanding of natural language texts. Finally, "shared" states that an ontology captures consensual knowledge, i.e., it is not private to some individual, but accepted by a group. Basically, the role of ontologies in the knowledge engineering process is to facilitate the construction of a domain model. An ontology provides a vocabulary of terms and relations with which a domain can be modeled.

Technically, ontologies define classes (or concepts) of a domain, properties of each concept describing various features and attributes of the concept (called slots, or roles or properties), and restrictions on slots (dubbed facets, or role restrictions).

The vocabulary specified by an ontology can be used in a formalized fashion to describe the content of a Web site, for example. In the world of Semantic Web Services, the same approach may be taken: Ontologies specify vocabulary to describe the capabilities of a Web Service, and the vocabulary is used by individual Web Services to mark up and advertise their services.

The crucial advantage of ontologies is that the semantics of the vocabulary contained in an ontology are clear. They have been agreed upon during the construction of the ontology by all stakeholders in the domain. Thus, these semantics can be hardwired into machine algorithms. As soon as an algorithm, e.g., an algorithm to match a service query to a service description, encounters vocabulary drawn from an ontology, it is able to understand the vocabulary since it has a predefined meaning.

4.7.1.2 Using Ontologies to Describe Web Services

Semantic Web Services can be described by using various ontologies in parallel, augmenting a functional service ontology by domain ontologies. Figure 4.12 depicts two very simple ontologies, a service ontology (a) and a domain ontology (b). The ontologies depicted only feature concepts and "is-a" (or inheritance) relationships among concepts. Ontologies may become way more complex than in this example:

- Ontologies may have additional types of links among concepts with entirely different semantics, such as "part-of" links (denoting that a concept is, e.g., physically speaking part of another concept), or "belongs-to" links. These links may be used to build more complex concept graphs.
- Concepts in the example may have properties. For example, the concept *Car* may have the properties *Color* and *Price*, all of which are inherited by *Car*'s subconcepts.
- Ontologies may be enriched with logical statements to explicitly encode semantics in the ontology. For example, inheritance (is-a) relationships may be considered transitive: If concept C inherits from concept B and concept B is a subconcept of concept A, then concept C also inherits from concept A. Such a rule may be written in, e.g., first order logic and attached to the ontology.

Most notably, ontologies are a vehicle to encode semantics in a machine-readable form. Thus, they also provide a means of describing Web Services and their capabilities. DAML-S [31] is a research effort which aims at providing primitives to describe the semantics of a Web Service. DAML-S itself is an ontology which contains vocabulary to describe, for example, the in- and output data of a Web Service, or preconditions and effects of the service. Additionally, the DAML-S ontology can be enhanced by combining its vocabulary with other ontologies, especially so-called domain ontologies. Domain ontologies compile important vocabulary to describe entities from certain domains. A large repository of domain ontologies from domains so diverse as weather, travel booking and finance can be found at [9].

In the DAML-S sense, a Web Service is described by a ServiceProfile, a ServiceModel and a ServiceGrounding. A service's ServiceProfile specifies a high-level description of the ser-

vice, it is used to register the service with service repositories such as UDDI. It contains a description of the service (in natural language, primarily intended to be read by humans), of its functionality (a formal specification of its in- and outputs, preconditions and effects) and its functional attributes (such as quality of service ratings). A service's ServiceModel contains a workflow-like description of the service's dynamics. Analogous to WS-BPEL, DAML-S specifies primitives with which a service workflow model can be built. A service's ServiceGrounding provides details on how a service can be accessed, most notably a specification of a communications protocol such as SOAP, RPC or CORBA-IDL.

DAML-S is a representative of the Semantic Web, whereas WSDL and WS-BPEL represent traditional Internet standards: As an ontology, DAML-S specifies vocabulary and the semantics of it an a machine-understandable way, while WSDL and WS-BPEL merely provide syntax standardizations.

In [37], the authors describe an algorithm which is capable of matching service requests with service descriptions based on the use of ontologies. A match between a service advertisement and a service request consists of the match of all the outputs of the request against the outputs of the advertisement, and all the inputs of the advertisement against the inputs of the request. (It may be extended to also include preconditions and effects of a service in the matching process.) Inputs and outputs of request and advertisements are lists of ontology concepts. The matching process computes a degree of distance between the lists of request and advertisement by comparing the ontology concepts used in both lists with each other: For example, if a concept is a subconcept of another concept, the concepts are semantically very close, and a low distance or a high degree of match is computed.

The previous sections have shown how to organize a P2P network in a way that allows for efficient search – however, the actual search was semantically undirected, i.e., it did not attempt to restrict the broadcast of the search message to potentially interesting peers. Having introduced ontologies as a possible means of marking up Semantic Web Services, an approach to discovering services based on ontology-based service descriptions will be described in the following.

4.7.2 Ontology-based Network Organization

Section 4.2 has used P2P networks to organize Web Services in a dynamic domain such that they can be searched efficiently. More efficiency will be achieved if a query for a particular service is not broadcasted (albeit efficiently broadcasted in HyperCuP) throughout the network, but the search is semantically directed from the very beginning: Queries are to be sent only to

services whose service descriptions somewhat match the service expected by the query. This section will present an algorithm which uses simple service descriptions to locate a service in a P2P network. Services with similar capabilities will thus be close to each other (in terms of graph distance, or hop counts on the P2P network) in the network.

The approach uses ontologies to describe Web Services in their very simplest form. Hence the difficult question of matching complex service queries to complex service descriptions is not addressed and not solved – rather, simple service descriptions will be used as a first-stage filter to reduce the size of the set of potentially interesting services for a particular query. Also, it may be part of future research efforts to incorporate more complex ontologies into the scheme.

4.7.2.1 Web Service Semantics

Service descriptions in the scheme presented here use ontologies as depicted in 4.12: An ontology is a tree of concepts, connected by is-a links denoting inheritance. A service uses available ontologies to choose which concepts it supports and which concepts it does not support. A service supports an ontology concept if the concept can be used to describe its capabilities.

A car retailer Web Service would describe itself by combining concepts from the service ontology with concepts from the car domain ontology, for example $C \wedge G \wedge I$. Semantic Web research has spawned many results on the design of and distributed negotiation on such ontologies which can well be reused to create service and domain ontologies, also in the field of Semantic Web Services [32].

Ideally, the P2P service network should allow for issuing a query to be sent to exactly those peers that can potentially answer the query. For example, a query $B \wedge I \wedge \neg G$ is to be broadcasted among those peers that buy vans, but are not interested in trucks. To allow for such broadcast containment, concept clusters are introduced into the hypercube network topology as described in Section 4.2: Peers with identical or similar interests or services are grouped in concept clusters which are in turn assigned to a specific logical combination of ontology concepts that describes best the peers belonging to the cluster.

The scheme that will be presented in the following is capable of handling ontologies that are encoded in the form of a tree. Links between concepts make up an 'is-a' or inheritance hierarchy. Thus, the ontologies used here resemble concept hierarchies.

In extension to the addressing scheme used in the original HyperCuP algorithm (Section 4.2), a peer's address (or rather, position) in an ontology-enriched HyperCuP network is divided into two classes of coordinates, concept and storage coordinates. This yields a topology in which concept clusters group peers with common interests or capabilities. Concept clusters

are internally organized using the original HyperCuP algorithm (the storage coordinates in a peer's position vector refer to the location of a peer within its concept cluster). These concept clusters are linked among each other in form of another (an 'outer' or concept) hypercube, again using a version of the HyperCuP algorithm (the concept coordinates in a peer's position vector refer to its location on the outer hypercube).

4.7.2.2 Concept Coordinates

Concept coordinates address a concept cluster on the concept hypercube. A set of structuring concepts is chosen to build this hypercube (see Section 4.7.4). A structuring concept is contained in one of the ontologies that are available to describe Web services participating in the network, i.e., in the service or domain ontologies. Each selected structuring concept is represented by a single ontology coordinate whose binary value in a concept coordinate vector reflects the support of peers in the addressed concept cluster for the respective structuring concept. Hence a peer describes itself by a concept vector $\vec{C} = (c_0, c_1, \ldots, c_{C-1})$ where each c_i represents a concept drawn from an ontology (not to be confused with a peer's cover map $\vec{c}$). Several ontologies can be combined, for example the two ontologies from Figure 4.13. In that case, the concept vector used to describe a peer simply contains the concatenation of the concepts in all ontologies combined. Peers only need to know the ontology graphs (and, if not all ontology concepts are used as structuring concepts, which concepts are *not* used) in order to build their personal concept vector (and thus determine their position in the network): Traversing each ontology graph in prefix order and simply listing the concepts visited yields an identical linearization of the ontology trees at every peer and thus a uniform, network-wide addressing scheme. Peers then set those digits to 1 that represent ontology concepts supported by them and thus create their concept vector.

The semantics of a concept vector is as follows: Each concept marked by a 1 in the binary concept vector $\vec{C}$ indicates that the peer supports the concept, i.e., its capabilities can be described by using the concept. Each concept marked by a 0 indicates that the peer does not support the concept. A concept vector is translated into a position vector on the concept hypercube by a simple rule:

$$\vec{p} = \vec{C} \tag{4.41}$$

A peer which describes its capabilities with the concept vector $\vec{C}$ will be located at position $\vec{p}$ on the concept hypercube.

Structuring concepts may be all concepts contained in an ontology. Sometimes, it may be decided to omit certain concepts in an ontology from becoming structuring concepts, for example since they are abstract concepts which will never be used by a peer to describe itself.

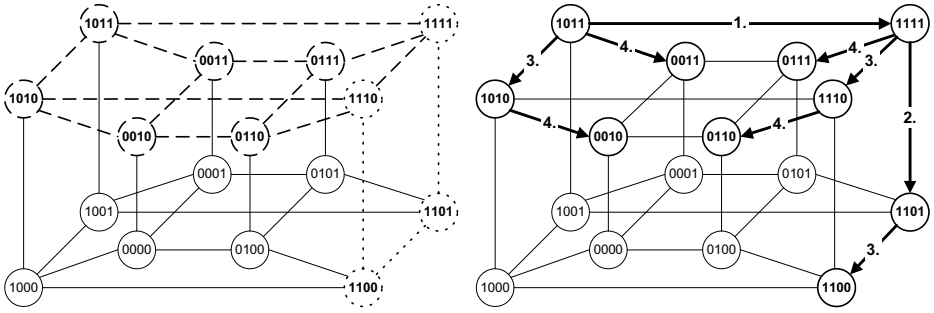

Figure 4.13: a. Concept hypercube topology b. Routing example

(The root concept of an ontology, e.g., "Web Service", will probably never be used in a peer description itself since it is just too general. Hence it may be omitted as structuring concept.) In the examples used here, four structuring concepts out of Figure 4.13 will be chosen, simply to be able to graphically depict the resulting 4-dimensional hypercube topology. In reality, all concepts from Figure 4.13 may be used.

4.7.2.3 Storage Coordinates

A concept cluster will contain more than one peer. Hence an additional address space is needed to accommodate multiple peers within a concept cluster. Storage coordinates denote the location of a peer within a specific concept cluster on the selected storage topology. Concept clusters form sub-graphs of the outer ontology-based concept hypercube – however, their internal topology can be based on hypercubes, star graphs or any other Cayley graph topology.

4.7.3 Ontology-based Network Querying

Querying the network works in two routing steps: First, the query is propagated to those concept clusters that contain peers which the query is aiming at. Second, a broadcast is carried out within each of these concept clusters, optimally forwarding the query to all peers within the clusters. This involves shortest-path routing in the concept coordinate system as well as restricted broadcast in the concept and storage coordinate systems. The algorithms described in Section 4.2.2 are applied.

4.7.3.1 Queries and Query Minimization

Analogous to the way peers describe their capabilities, queries ask for peers that support certain capabilities. Hence a query is made up by a concept query vector $\vec{q} = (c_0, c_1, \ldots, c_{C-1})$. In this vector, again all concepts in an ontology (or a combination of ontologies) are either

switched on or off. However, the semantics are slightly different as compared to peer descriptions: Concepts that are switched off are regarded as "dont't care" rather than being regarded as concepts that are explicitly *not* supported by a peer. If a peer receives the query and realizes that in its description $\vec{p}$ all concepts are switched on that are switched on in the query $\vec{q}$, plus some additional concepts are switched on in its peer description, this means that the peer is more specialized than the query actually requires it to be – it is able to carry out a more specialized service. However, due to the fact that the activated query concepts are a subset of its description concepts, it is also still able to provide the more general service requested by the query. As an example, a peer might be able to accept payments via credit card and cash, whereas the query is just looking for a peer that accepts payments via credit card. Thus, the peer's capabilities satisfy the query.

Query Minimization. A query that is issued by a peer undergoes logic minimization (e.g. Karnaugh minimization) to identify its logical minterms (conjunctions of structuring concepts). Minterms denote a group of concept clusters. All concept clusters pointed to by a single minterm are direct neighbors of each other in the network topology. Figure 4.13a depicts the 'outer' concept hypercube of a network that is organized by structuring concepts A, D, E and F from the ontologies in Figure 4.12. The query $E \vee A \wedge D$ consists of minterms E as well as $A \wedge D$ and asks for some peer that is a wholesale service or a combined retail and delivery service.

4.7.3.2 Query Routing

Distinct minterms in the query resemble distinct groups of concept clusters in the network. To each of these groups, one copy of the query message has to be delivered to enable them to carry out broadcasts within the group. These groups may overlap or are adjacent. Figure 4.13a depicts the 4-dimensional concept hypercube that is created by using concepts A, D, E and F as structuring concepts. Each node in the network represents a concept cluster (for example, node 0101 represents the concept cluster containing peers which are motor vehicle retailers). The two minterms in the query are associated with two (overlapping) groups of concept clusters, highlighted in Figure 4.13a.

Routing to concept clusters. A copy of the broadcast message is delivered to each concept cluster addressed in the query. If queries span groups of concept clusters, this can be accomplished by carrying out restricted broadcasts in the concept coordinate system. Figure 4.13b depicts the broadcast steps that are executed in order to inform all concept clusters addressed by the query $E \vee A \wedge D$. The broadcast algorithm modifies the algorithm described in Section 4.2.2: A concept cluster group associated with a minterm is described by the set of dimensions in which it exists (directly associated with the concepts it supports). Broadcast

is carried out only in those dimensions and branches out into additional dimensions at peers which belong to more than one minterm or are adjacent to peers belonging to another minterm. In order to start the broadcast, the broadcast message has to reach any peer within the concept cluster group – this is achieved by shortest-path routing from the querying peer to the closest peer in the group (by correcting one address bit in each routing step).

Broadcast in concept clusters. All informed concept clusters broadcast the query message among all their member peers. Broadcasting is carried out in the storage coordinate system, restricting it to the peers that belong to the broadcasting concept cluster.

Peer feedback. Once the query message has arrived at a peer, the peer is able to react to the message – for example, by contacting the issuer of the query in order to establish a business relationship.

4.7.4 Topology Construction

Peers can join the ontology-based topology by contacting any peer already in the network. A new peer V reveals its capabilities in terms of concepts contained in any of the available global ontologies in its concept vector $\vec{C}_v$, and it is to be integrated in the concept cluster matching its description, at HyperCuP address $\vec{p_v} = \vec{C}_v$. If the peer describes itself with concepts that are not used as structuring concepts in the network, it is integrated in the most specific existing concept cluster. A join message is then routed to any peer in this cluster, using shortest-path routing. The contacted peer then integrates the new peer into the concept cluster using precisely the algorithm described in Section 4.3.1. Note that not all clusters will actually be built: Only those clusters which represent a concept combination that actually is used by a peer in the network will be built. All other clusters are not populated by any nodes and therefore not built. The clusters will be interconnected in the usual HyperCuP way as described in Section 4.3.1 – thus, concept clusters which do not contain any peers can simply be missing in the topology, just as positions can be vacant in the original protocol.

4.7.5 Extensions

The scheme can be made more efficient by considering is-a relationships in the ontologies used and by building hierarchical networks.

4.7.5.1 Exploiting Is-A Relationships

So far, the scheme presented above does not exploit the fact that the ontology used may contain is-a relationships (as does the ontology in Figure 4.13). In fact, considering is-a relationships in building the network will reduce the number of clusters to be built. Currently, a maximum number of 2^C clusters may be built if the ontology used contains C concepts. This

number is reduced to $2^{C_{leaf}}$ clusters if the following rule is used: Only concepts which are leaf concepts of the ontology tree are used as structuring concepts. The width of the position and query vectors in the network is thus reduced from C to C_{leaf} bits. In routing queries, an additional interpretation is required:

If m c_i are all immediate subconcepts of concept $c*$, i.e., there is an is-a relationship between each concept c_i and $c*$ in the ontology and $c*$ does not have any further immediate subconcepts, then

$$\bigwedge_{i=0}^{m-1} c_i \Leftrightarrow c* \tag{4.42}$$

$$\bigwedge_{i=0}^{m-1} \neg c_i \Leftrightarrow \neg c* \tag{4.43}$$

Logically speaking, a closed-world assumption is made: A super-concept $c*$ is always supported if all of its subconcepts c_i are supported. Conversely, a concept $c*$ cannot be supported if neither of its subconcepts c_i is supported.

Using this rule, routing can proceed virtually without any modifications: If a query is issued which requires a non-leaf concept $c*$ from the ontology to be set (for which there is no concept coordinate in the query vector), instead simply all subconcepts of $c*$ are set in the query vector. This continues recursively until all affected leaf concepts are set in the query vector. For example, the concept vector using all concepts from Figure 4.12 would be

$$\vec{C} = (ABDCEGFIH)^T \tag{4.44}$$

A peer searching for a service supporting concepts *Sell* and *Passenger Vehicle* would issue the query $\vec{q} = (000101000)^T$. Using the is-a based compression, the concept vector of the network becomes

$$\vec{C} = (ABDEGI) \tag{4.45}$$

The query searching for peers supporting the concepts *Sell* and *Passenger Vehicle* becomes $\vec{q} = (001101)^T$. Instead of activating the super-concepts C and H – which are not used as structuring concepts any more, hence are not part of the concept and query vectors any more – their respective sub-concepts are activated.

4.8 Related Work

Making P2P networks scalable has recently received much attention. Distributed hash table approaches [41] such as CAN [39] and Chord [43] aim at enforcing a deterministic content distribution instead which can be used for routing point queries. While similar in terms of

message complexity for joining and departing nodes, the approach presented here specifically performs well at optimizing the network load in broadcast and multipoint search, without requiring hash functions, and allows for more detailed queries, viz. logical combinations of ontology concepts.

[36] constructs an efficient P2P topology, yet does not provide means of clustering peers with similar capabilities. Also, the topology construction algorithm relies on a centralized server – despite the fact that the server does not have to handle too much traffic in the approach, it opposes the fundamental idea of a symmetric and self-organizing P2P network.

[4] describes a distributed algorithm which is capable of building random expander graphs in a distributed fashion. Despite not being deterministic topologies, random expander graphs exhibit properties that allow for efficient search and broadcast on these graphs. The algorithm presented in [4] also is of logarithmic complexity (logarithmic to the number of nodes in the network – i.e., nodes send a logarithmic amount of messages when joining or leaving the network, the network diameter is logarithmic etc.). However, simulations have to show if its advertised properties hold.

Building a Semantic Service Web on a P2P infrastructure is opposed to centralized approaches such as UDDI [50]. Automated composition and verification of Semantic Web Services is addressed in [32], building on the service description framework DAML-S [31]. The approach presented here facilitates service discovery as a major building block of automated service composition.

4.9 Conclusion

In this chapter, a topology to efficiently cluster peers in a P2P network was presented which features efficient broadcast and search algorithms without any message overhead during broadcast, logarithmic network diameter, and resiliency towards node failure. Super peers or central servers are not required. A set of globally known ontologies is used to categorize peers as providers of particular services to efficiently route and broadcast queries. Organizing peers in this manner allows for enhancing Semantic Web Services technology with the flexibility and dynamics of P2P networks while ensuring scalability to a large number of nodes.

Chapter 5

Distributed Trust Management

Peer-to-peer networks were introduced as a practical infrastructure for publishing and discovering Web Services, able to cope with a large and highly dynamic body of service providers offering a diverse range of distinct services. However, the greatest advantage of the infrastructure, its distributed principle of operation, may also become its greatest vulnerability: Peers on such a network do not know which service providers can be expected to deliver good quality services and which service providers may even try to cause damage when invoked.

In fact, recent experience on the file-sharing Gnutella network shows that the anonymous, open nature of file-sharing networks offers an almost ideal environment for the spread of self-replicating inauthentic files, a strategy to attack a P2P network that shall be described in more detail below. In very much the same way, peers on a P2P Web of Services may offer services which in some way cause damage to individual peers and the network.

This chapter will present an algorithm to decrease the number of downloads of inauthentic files in a peer-to-peer file-sharing network. Its application in the domain of Semantic Web Services will be described. The algorithm assigns each peer a unique global reputation value, based on the peer's history of uploads. A distributed and secure method is presented to compute global reputation values, based on Power iteration. By having peers use these global reputation values to choose the peers from whom they download, the network effectively identifies malicious peers and isolates them from the network.

Simulations will show that the reputation system presented here significantly decreases the number of inauthentic files on the network, even under a variety of conditions where malicious peers cooperate in an attempt to deliberately subvert the system.

5.1 Why Trusting Everybody Is Just As Bad An Idea As Trusting Nobody

Peer-to-peer file-sharing networks have many benefits over standard client-server approaches to data distribution, including increased robustness, scalability, and diversity of available data. However, the open and anonymous nature of these networks leads to a complete lack of accountability for the content a peer puts on the network, opening the door to abuses of these networks by malicious peers.

Attacks by anonymous malicious peers have been observed on today's popular peer-to-peer networks. For example, malicious users have used these networks to introduce viruses such as the *VBS.Gnutella* worm, which spreads by making a copy of itself in a peer's Gnutella program directory and then modifying the Gnutella.ini file to allow sharing of .vbs files [48]. Far more common have been inauthentic file attacks, wherein malicious peers respond to virtually any query providing "decoy files" that are tampered-with or do not work.

It has been suggested that the future development of P2P systems will depend largely on the availability of novel methods for ensuring that peers obtain reliable information on the quality of resources they are receiving [6]. In this context, identifying malicious peers as sources of inauthentic or bad quality files is a method superior to attempting to track down inauthentic files themselves (a method implemented on the KaZaA P2P network, called Integrity Rating [25]) - since malicious peers can easily generate a virtually unlimited number of inauthentic files if they are not banned from participating in the network. This chapter presents such a method wherein each peer i is assigned a unique *global reputation value* that reflects the experiences of all peers in the network with peer$i$. In this approach, all peers in the network participate in computing these values in a distributed and node-symmetric manner with minimal overhead on the network. Furthermore, it is described how to ensure the security of the computations, minimizing the probability that malicious peers in the system can lie to their own benefit. And finally, it is shown how to use these values to identify peers that provide material deemed inappropriate by the users of a peer-to-peer network, and effectively isolate them from the network.

5.2 Reputation Systems

An important example of successful reputation management is the online auction system eBay [13]. In eBay's reputation system, buyers and sellers can rate each other after each transaction, and the overall reputation of a participant is the sum of these ratings over the last 6 months. This system relies on a centralized system to store and manage trust ratings.

In a distributed environment, peers may still rate each other after each transaction, as in the eBay system. For example, each time peer i downloads a file from peer j, it may rate the transaction as positive (+1) or negative (-1). Some reasons why peer i may rate a download as negative if the file downloaded is inauthentic or tampered with, or if the download is interrupted. In the domain of Semantic Web Services, transactions of files equals transactions of services: If service A discovers and invokes another service B, service A has the right to evaluate service B's performance after the transaction has been closed. Like in the eBay model, a *local reputation value* s_{ij} can be defined as the sum of the ratings of the individual transactions that peer i has downloaded from peer j.

Equivalently, each peer i can store the number satisfactory transactions it has had with peer j, $sat(i,j)$ and the number of unsatisfactory transactions it has had with peer$j$, $unsat(i,j)$. Then, s_{ij} is defined:

$$s_{ij} = sat(i,j) - unsat(i,j) \tag{5.1}$$

Note that this is a fairly simple way of assessing the quality of a file or service transaction. However, it is not to be used to evaluate the precise quality of service offered: The method aims at preventing *malicious behavior* in the first place. Hence an unsatisfactory service exchange has taken place if a service provider has clearly not offered the service that had been advertised before. As a drastic example, a service provider may offer a service which defragments a user's hard-drive – in fact, the service then uses the right to access the user's hard-drive, granted by the user himself, to format the drive. In contrast, a service transaction should not be rated as unsatisfactory if the response or processing time of the service provider was slow, e.g., due to bandwidth problems. In such a case, the service provider obviously has not attempted to deliberately cause damage to the network or the user, and only those cases should be captured by the above definition.

This chapter presents a reputation system that aggregates the local reputation ratings of all of the users in a natural manner, with minimal overhead in terms of message complexity. The approach is based on the notion of transitive trust: A peer i will have a high opinion of those peers who have provided it authentic files. Moreover, peer i is likely to trust the opinions of those peers, since peers who are honest about the files they provide are also likely to be honest in reporting their local trust assessments.

As remains to be shown in the course of this chapter, the idea of transitive trust leads to a system where global reputation values correspond to the left principal eigenvector of a matrix of normalized local reputation values. This eigenvector computation can be performed in a distributed manner with just a few lines of code, where the message complexity is provably bounded and empirically low. Most importantly, it will be shown that this system is highly

effective in decreasing the number of unsatisfactory downloads, even when up to 70% of the peers in the network form a malicious collective in an attempt to subvert the system.

5.3 Design Considerations

There are six issues that are important to address in any P2P reputation system.

1. The system should be *self-policing*. That is, the shared ethics of the user population are defined and enforced by the peers themselves and not by some central authority.
2. The system should maintain *anonymity*. That is, a peer's reputation should be associated with an opaque identifier (such as the peer's Gnutella username) rather than with an externally associated identity (such as a peer's IP address).
3. The system should not assign any *profit to newcomers*. That is, reputation should be obtained by consistent good behavior through several transactions, and it should not be advantageous for malicious peers with poor reputations to continuously change their opaque identifiers to obtain newcomers status.
4. The system should have *minimal overhead* in terms of computation, infrastructure, storage, and message complexity.
5. The system should be *robust to malicious collectives* of peers who know one another and attempt to collectively subvert the system.

5.4 A Power-Iteration Based Approach

In this section, a system is described where the global reputation of each peer i is given by the local reputation values assigned to peer i by other peers, weighted by the global reputations of the assigning peers. Section 5.4.1 shows how to normalize the local reputation values in a manner that leads to an elegant probabilistic interpretation and an efficient algorithm for aggregating these values. Section 5.4.2 discusses how to aggregate the normalized reputation values in a sensible manner. Section 5.4.3 introduces discuss the probabilistic interpretation of the local and global reputation values. In Section 5.4.4 through Section 5.4.6, an algorithm for computing the global reputation values is presented. Section 5.6 finally discusses how to use the global reputation values.

5.4.1 Normalizing Local Reputation Values

In order to aggregate local reputation values, it is necessary to normalize them in some manner. Otherwise, malicious peers can assign arbitrarily high local reputation values to other malicious

peers, and arbitrarily low local reputation values to good peers, easily subverting the system. A *normalized local reputation value*, c_{ij}, is defined as follows:

$$c_{ij} = \frac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij}, 0)} \tag{5.2}$$

This ensures that all values will be between 0 and 1. (Notice that if $\sum_j \max(s_{ij}) = 0$, then c_{ij} is undefined. This case will be addressed in Section 5.4.4.) There are some drawbacks to normalizing in this manner. For one, the normalized reputation values do not distinguish between a peer with whom peer i did not interact and a peer with whom peer i has had poor experience. Also, these c_{ij} values are relative, and there is no absolute interpretation. That is, if $c_{ij} = c_{ik}$, it is known that peer j has the same reputation as peer k in the eyes of peer i, but it is unclear if both of them are very reputable, or if both of them are mediocre. However, here the local reputation values shall be normalized in this manner because it leads to an elegant probabilistic model, and allows for performing the computation that will be described below without renormalizing the global reputation values at each iteration (which is prohibitively costly in a large distributed environment). Furthermore, substantially good results can still be achieved despite the drawbacks mentioned above.

5.4.2 Aggregating Local Reputation Values

The normalized local trust values have to be aggregated. A natural way to do this in a distributed environment is for peer i to ask its acquaintances about their opinions about other peers. It would make sense to weight their opinions by the trust peer i places in them:

$$t_{ik} = \sum_j c_{ij} c_{jk} \tag{5.3}$$

where t_{ik} represents the trust that peer i places in peer k based on asking his friends.

This can be written in matrix notation: If C is defined to be the matrix $[c_{ij}]$ and $\vec{t_i}$ to be vector containing the values t_{ik}, then $\vec{t_i} = C^T \vec{c_i}$. (Note that $\sum_j t_{ij} = 1$ as desired.)

This is a useful way to have each peer gain a view of the network that is wider than his own experience. However, the reputation values stored by peer i still reflect only the experience of peer i and his acquantainces. In order to get a wider view, peer i may wish to ask his friends' friends ($t = (C^T)^2 c_i$). If he continues in this manner, ($t = (C^T)^n c_i$), he will have a complete view of the network after $n =$ large iterations (under the assumptions that C is irreducible and aperiodic, which is guaranteed in practice and addressed in Section 5.4.5).

Fortunately, if n is large, the trust vector $\vec{t_i}$ will converge to the same vector *for every peer i*. Namely, it will converge to the left principal eigenvector of C. In other words, $\vec{t}$ is a global reputation vector in this model. Its elements, t_j, quantify how much trust the system as a whole places peer j.

5.4.3 Probabilistic Interpretation

It is useful to note that there exists a straightforward probabilistic interpretation of this method, similar to the Random Surfer model of [35]. If an agent were searching for reputable peers, it can crawl the network using the following rule: at each peer i, it will crawl to peer j with probability c_{ij}. After crawling for a while in this manner, the agent is more likely to be at reputable peers than unreputable peers. The stationary distribution of the Markov chain defined by the normalized local reputation matrix C is the global reputation vector $\vec{t}$.

5.4.4 Non-distributed Algorithm

In this section, the basic trust algorithm is presented, ignoring for now the distributed nature of the peer-to-peer network. That is, it is assumed that some central server knows all the c_{ij} values and performs the computation. Section 5.4.6 describes how the computation may be performed in a distributed environment.

The computation has to yield $\vec{t} = (C^T)^n \vec{e}$, for n =large, where $\vec{e}$ is defined to be the m-vector representing a uniform probability distribution over all m peers, $e_i = 1/m$. (In Section 5.4.2, it was stated that $\vec{t} = (C^T)^n \vec{c_i}$ is to be computed, where $\vec{c_i}$ is the normalized local reputation vector of some peer i. However, since they both converge to the principal left eigenvector of C, $\vec{e}$ can be used instead.)

At the most basic level, the algorithm would proceed as in Algorithm 1:

$\vec{t}^{(0)} = \vec{e}$
While $\delta < \epsilon$ repeat {
$\vec{t}^{(k+1)} = C^T \vec{t}^{(k)}$
$\delta = ||t^{(k+1)} - t^k||$
}

5.4.5 Practical Issues

There are three practical issues that are not addressed by this simple algorithm: a priori notions of trust, inactive peers, and malicious collectives.

A priori notions of trust. Often, there are some peers in the network that are known to be trustworthy. For example, the first few peers to join a network are often known to be trustworthy, since malicious peers generally don't enter a peer-to-peer network until later in the network's development. It would be useful to incorporate such notions of trust in a natural and seamless manner. This is done by defining some distribution $\vec{p}$ over pre-trusted peers. (For example, if some set of peers P are known to be trusted, it can be defined that $p_i = 1/|P|$ if $i \in P$, and $p_i = 0$ otherwise.) This distribution $\vec{p}$ is used in three ways. First of all, in the

presence of malicious peers, $\vec{t} = (C^T)^n \vec{p}$ will generally converge faster than $\vec{t} = (C^T)^n \vec{e}$, so $\vec{p}$ is used as the start vector. The other two ways to use this distribution $\vec{p}$ shall be described below.

Inactive Peers. If peer i doesn't download from anybody else, or if it assigns a zero score to all other peers, c_{ij} from Equation 5.1 will be undefined. In this case, it is set $c_{ij} = p_j$. So c_{ij} is redefined as:

$$c_{ij} = \begin{cases} \frac{max(s_{ij},0)}{\sum_j max(s_{ij})} & \text{if } \sum_j max(s_{ij},0) \neq 0; \\ p_j & \text{otherwise} \end{cases} \tag{5.4}$$

That is, if peer i doesn't know anybody, or doesn't trust anybody, he will choose to trust the pre-trusted peers.

Malicious Collectives. In peer-to-peer networks, there is potential for malicious collectives to form. A malicious collective is a group of malicious peers who know each other, who give each other high local reputation values and give all other peers low local trust values in an attempt to subvert the system and gain high global reputation values. This issue can be addressed by taking

$$\vec{t}^{(k+1)} = (1-a)C^T \vec{t}^{(k)} + a\vec{p} \tag{5.5}$$

where a is some constant less than 1. This is equivalent to setting the opinion vector for all peers to be $\vec{c_i} = (1-a)\vec{c_i} + a\vec{p}$, breaking collectives by having each peer place at least some trust in the peers P that are not part of a collective. Probabilistically, this is equivalent to saying that the agent that is crawling the network by the probabilistic model given in Section 5.4 is less likely to get stuck crawling a malicious collective, because at each step, he has a certain probability of crawling to a pre-trusted peer. Notice that this also makes the matrix C is irreducible and aperiodic, guaranteeing that the computation will converge.

The modified algorithm is given in Algorithm 2.

$\vec{t}^{(0)} = \vec{p}$
While $\delta < \epsilon$ repeat {
$\vec{t}^{(k+1)} = C^T \vec{t}^{(k)}$
$\vec{t}^{(k+1)} = (1-a)\vec{t}^{(k+1)} + a\vec{p}$
$\delta = ||t^{(k+1)} - t^{(k)}||$
}

5.4.6 Distributed Algorithm

Here, an algorithm is presented where all peers in the network cooperate to compute and store the global trust vector, and the computation, storage, and message overhead for each peer are minimal.

In a distributed environment, the first challenge that arises is how to store C and $\vec{t}$. In previous sections, it was suggested that each peer could store its local trust vector $\vec{c_i}$. Here, it is also suggested that each peer store its own global trust value t_i. (For presentation purposes, issues of security will be ignored for the moment and peers will be allowed to store their own trust values. Security issues will be handled by Section 5.5.)

In fact, each peer can compute it's own global trust value:

$$t_i^{(k+1)} = (1-a)(c_{1i}t_1^{(k)} + \ldots + c_{ni}t_n^{(k)}) + ap_i \tag{5.6}$$

Inspection will show that this is the component-wise version of $\vec{t}^{(k+1)} = (1-a)C^T\vec{t}^{(k)} + a\vec{p}$. Notice that, since peer i has had limited interaction with other peers, many of the components in equation 5.6 will be zero. This lends itself to the simple distributed algorithm shown in Algorithm 3.

Definitions:

- A_i: set of peers which have downloaded files from
- B_i: set of peers from which peer i has downloaded files

Algorithm: Each peer i do {
Query all peers $j \in A_i$ for $t_j^{(0)} = p_j$
While $\delta < \epsilon$ repeat {
Compute $t_i^{(k+1)} = (1-a)(c_{1i}t_1^{(k)} + c_{2i}t_2^{(k)} + \ldots + c_{ni}t_n^{(k)}) + ap_i$
Send $c_{ij}t_i^{(k+1)}$ to all peers $j \in B_i$
Compute $\delta = |t_i^{(k+1)} - t_i^{(k)}|$
Wait for all peers $j \in A_i$ to return $c_{ji}t_j^{(k+1)}$
}
}

It is interesting to note two things here. First of all, only the pre-trusted peers need to know their p_i. This means that pre-trusted peers may remain anonymous; nobody else needs to know that they are pre-trusted[1]. Therefore, the pre-trusted peers maintain anonymity as pre-trusted peers.

Secondly, in most P2P networks, each peer has limited interaction with other peers. There are two benefits to this. First, the computation $t_i^{(k+1)} = (1-a)(c_{1i}t_1^{(k)} + c_{2i}t_2^{(k)} + \ldots + c_{ni}t_n^{(k)}) + ap_i$ is not intensive, since most c_{ji} are zero. Second, the number of messages passed is small,

[1] Recall that, for the moment, it is assumed that peers are honest and may report their own trust values, including whether or not they are a pre-trusted peer. The secure version is presented in Section 5.5

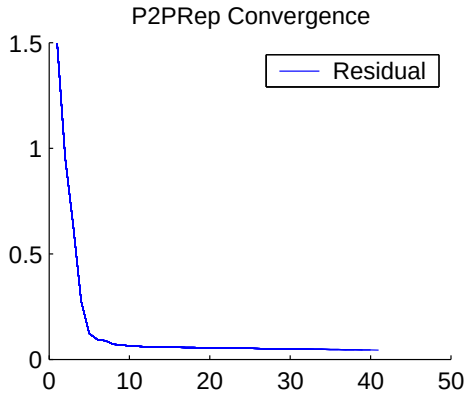

Figure 5.1: Algorithm convergence

since A_i and B_i are small. In the case where a network is full of heavily active peers, these benefits can be enforced by limiting the number of local trust values c_{ij} that each peer can report.

5.4.7 Algorithm Complexity

The complexity of the algorithm is bounded in two ways. First, the algorithm converges fast: For a network of 1000 peers after 100 query cycles (refer to Section 5.7.1 for a description of how to simulate the system), Figure 5.1 depicts the residual $\|t^{(k+1)} - t^{(k)}\|_1$. Clearly, the algorithm has converged after less than 10 iterations, i.e., the computed global reputation values do not change significantly any more after a low number of iterations. In the distributed version of the algorithms, this corresponds to less than 10 exchanges of updated reputation values among peers.

Second, the number of local reputation values that a peer reports can be explicitly limited. In the modified version of the algorithm, each peer reports a subset of its total set of local reputation values. Preliminary simulations have shown this scheme to perform comparably well as the algorithm presented here, where peers report all of their local reputation values.

5.5 Secure Algorithm

In the algorithm presented in the previous section, each peer i computes and reports its own trust value t_i. Malicious peers can easily report false trust values, subverting the system.

This is battled by implementing two basic ideas. First, the current trust value of a peer must not be computed by and reside at the peer itself, where it can easily become subject to manipulation. Thus, a different peer in the network computes the trust value of a peer. Second, it will be in the interest of malicious peers to return wrong results when they are

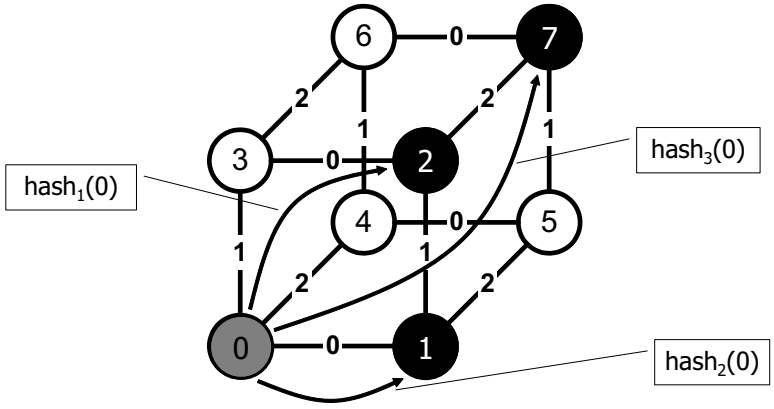

Figure 5.2: Example HyperCuP topology

supposed to compute any peer's trust value. Therefore, the trust value of one peer in the network will be computed by more than one other peer.

In the secure version of the distributed trust computation algorithm, M peers (dubbed 'mother peers' of a peer i) compute the trust value of a peer i. If a peer needs the trust value of peer i, it can query all M mother peers for it. A majority vote on the trust value then settles conflicts arising from a number of malicious peers being among the mother peers and presenting faulty trust values as opposed to the correct one presented by the non-malicious mother peers.

To assign mother peers, HyperCuP (see Section 4.2) can be used in an elegant way. As described in Section 4.2.2 and in Section 4.7, the coordinates of a node in the network can be assigned a meaning that can be exploited by search algorithms.

In the context of the P2P reputation algorithm described here, a hash function is used to deterministically map keys into points in a logical coordinate space. In the HyperCuP algorithm, the logical coordinate space resembles a hypercube grid. At any time, the coordinate space is partitioned dynamically among the peers in the system by the HyperCuP algorithm such that every peer covers a region in the coordinate space. Peers are responsible for storing (key, value) pairs the keys of which are hashed into a point that is located within their region.

In the approach presented here, a peer's mother is located by hashing a unique ID of the peer, such as its IP address and TCP port, into a point in the HyperCuP hash space, i.e., on the hypercube grid. The peer which currently covers this point as part of its covering responsibility is appointed as the mother of that peer. All peers in the system which know the unique ID of a peer can thus locate its mother peer. The initial algorithm can be modified such that it can be executed by mother peers.

As an example, consider the HyperCuP topology in Figure 5.2. Peer 0's unique IDID_0 is mapped into points covered by peers 1, 2 and 7, respectively, by hash functions$hash_1$, $hash_2$ and $hash_3$. Thus, these peers become peer 0's mother peers.

It is also important to cope with the inherent dynamics of a P2P system. When a mother peer leaves the system, it passes on its state (i.e., trust values or ongoing trust computations) to its covering neighbor. To increase the reliability of the process, data replication may be introduced as an extension to the system.

5.5.1 Algorithm Description

In the following, the secure algorithm to compute a global trust vector will be described. These definitions shall be used: Each peer has a number M of mother peers, whose HyperCuP coordinates are determined by applying a set of one-way secure hash functions$h_0, h_1, \ldots, h_{M-1}$ to the peer's unique identifier. pos_i are the coordinates of peer i in the hash space. Since each peer also acts as a mother peer, it is assigned a set of daughters D_i - the set contains the indexes of peers whose trust value computation is covered by the peer. As a mother peer, peer i also maintains the opinion vector c_d^i of its daughter peer d (where $d \in D_i$) at some point in the algorithm. Also, peer i will learn A_d^i which is the set of peers which downloaded files from its daughter peer d: It will receive trust assessments from these peers referring to its daughter peer d. Finally, peer i will get to know the set B_d^i which denotes the set of peers which its daughter peer d downloaded files from: Upon kicking off a global trust value computation, its daughter peer d is supposed to submit its trust assessments on other peers to its mother peer, providing the mother peer with B_d^i.

For each peer i\{
Submit local trust ratings $\vec{c_i}$ to all mother peers at positions $h_m(pos_i)$, $m = 1 \ldots M-1$
Collect local trust ratings $\vec{c_d}$ and sets of acquaintances B_d^i of daughter peers $d \in D_i$
Submit daughter d's local trust ratings c_{dj} to mother peers $h_m(pos_d)$, $m = 1 \ldots M-1, \forall j \in B_d^i$
Collect acquaintances A_d^i of daughter peers
For each daughter peer $d \in D_i$\{
Query all peers $j \in A_d^i$ for $c_{jd} t_{\text{trusted}j}$
While $|t_d^{(k+1)} - t_d^{(k)}| < \epsilon$ repeat \{
Compute $t_d^{(k+1)} = (1-a)(c_{1d}t_1^{(k)} + c_{2d}t_2^{(k)} + \ldots + c_{nd}t_n^{(k)}) + at_{\text{trusted}d}$
Send $c_{dj}t_d^{(k+1)}$ to all peers $j \in B_d^i$
Wait for all peers $j \in A_i^d$ to return $c_{jd}t_j^{(k+1)}$
\}
\}

}

Upsides of the secure algorithm in terms of increased security and reliability include:

Anonymity. It is not possible for a peer at a specific coordinate to find out which peer ID exactly it computes the trust for - hence malicious peers cannot increase the trust of other malicious peers.

Randomization. Peers that enter the system cannot select at which coordinates in the hash space they want to be located (due to the randomization methodology described in Section 4.3.3) - hence it is not possible for a peer to, for example, compute the hash value of its own ID and locate itself at precisely this position in the hash space to be able to compute its own trust value.

Redundancy. Several mothers compute the trust value for one peer. To assign several mothers to a peer, several multi-dimensional hash functions are used. Peers in the system still take over a particular region in the coordinate space, yet now there are several coordinate spaces, each of which is created by one multi-dimensional hash function. A peer's unique ID is thus mapped into a different point in every multi-dimensional hash space.

5.5.2 Security Improvements

Effect of multiple mothers on trust value authenticity. The system assigns M mother peers to compute the trust value of a single peer. If a majority vote among mother peers is used upon querying for the peer's trust value, a faulty trust value would be presented to the system if there were more than $\frac{M}{2}$ malicious peers among the peer's set of M mother peers. Assuming the network is populated by a fraction ofp_{mal} malicious peers, the probability that there are more than $\frac{M}{2}$ malicious peers among a peer's M mother peers is

$$P = \sum_{i=\frac{M}{2}}^{M} \binom{M}{i} \cdot p_{mal}^{i} \cdot (1 - p_{mal})^{M-i} \tag{5.7}$$

This equation resembles a Bernoulli process in which the probability of a peer being a malicious peer is p_{mal}, and M peers are examined. Increasing the number of mothers per peer decreases the probability of malicious peers being able to attack the system by computing faulty trust values. However, it does increase the control traffic imposed on the system, too - since several mothers now have to be kept posted on a peer's interactions.

Effect of multiple mothers on trust computation authenticity. Peers in the system do not submit their opinion assessments directly, but they pass their opinion assessments on to their mothers. Their mother peers then normalize and submit the opinion assessments. That way, an opinion assessment received by a mother peer has to be received in$M$ identical copies – since each of a peer's M mothers submits the same opinion assessment. If one or

more mother peers are malicious, fewer than M identical copies will arrive. A mother peer accepts a submitted opinion assessment if at least $\lceil \frac{M}{2} \rceil$ identical copies are received.

Majority of malicious mothers. What if a peer is assigned a majority of malicious mothers? The probability can be designed to be arbitrarily small by increasing the number of mother peers, yet this comes at the expense of increased message complexity. To prevent a peer from being permanently excluded from network operations, the set of mother peers of a peer is changed after every trust rank computation. However, this does not decrease the probability of a peer having a majority of malicious mother peers in one trust rank pass. In that case, a peer would lose the trust value which it has built up during potentially many former trust rank passes in an instant – the malicious mother peers might report its trust value as 0 to the system, throwing it back into the state of a peer which has just entered the system. This problem is overcome by combining the global trust value of a peer with that peer's trust value cached locally at another peer. Essentially, if a peer has made good experiences with downloading files from a peer, there is no reason why it should be discouraged from doing so by a sudden and sharp drop in the peer's global trust value. Thus, peers will not be discouraged from downloading from the fooled peer and will report a solid opinion value to the peer's next set of mothers, which is highly unlikely to consist of yet another majority of malicious mother peers.

5.6 Using Global Reputation Values

There are two clear ways to use these global reputation values in a peer-to-peer system. The first is to isolate malicious peers from the network by biasing users to download from reputable peers. The second is to incent peers to share files by rewarding reputable peers.

Isolating Malicious Peers. When peer i issues a query, the system may use the reputation values t_j to bias the user towards downloading from more reputable peers. One way to do this would be to have each peer download from the most highly trusted peer who responds to its query. However, such a policy leads to the most highly trusted peers being overloaded, as shown in Section 5.7. Furthermore, since trust is built upon sharing authentic files, this policy does not enable new peers to build up trust in the system.

A different strategy is to select the peers from whom to download probabilistically based on their trust values. In particular, the probability that a peer will download a file from responding peer j can be set to be directly proportional to the reputation valuet_j of peer j.

Such a policy limits the number of unsatisfactory downloads on the network, while balancing the load in the network and allowing newcomers to build trust. The experiments in Section 5.7 validate this.

Incenting Freeriders to Share. Secondly, the system may reward peers with high reputation values. For example, reputable peers may be rewarded with increased connectivity to other reputable peers, or greater bandwidth. Rewarding reputable peers has a twofold effect. It gives users an incentive to share files, since a good trust rating may only be achieved by sharing authentic files. In the current Gnutella network, less than 7% of the peers are responsible for over 50% of the files, and as many as 25% of peers on the network share no files at all [42]. Incentives based on trust ratings should reduce the number of free riders on peer-to-peer networks. Also, rewarding highly trusted peers gives non-malicious peers an incentive to delete inauthentic files that they may have accidentally downloaded from malicious peers, actively keeping the network tidy. This makes it more difficult for inauthentic files to replicate in the system.

5.7 Experiments

In this section, the performance of the scheme as compared to a P2P network where no trust model is implemented will be assessed. The scheme's performance shall be examined under a variety of threat models.

5.7.1 Simulation

The experimental findings are based on simulations of a P2P network model the details of which are discussed in Section 5.7.1.

Node model. The network consists of good nodes (normal nodes, participating in the network to download and upload files) and malicious nodes (adversarial nodes, participating in the network to undermine its performance). In the experiments, different threat models will be considered, where a threat model describes the behavior of a malicious peer in the network. Threat models will be described in more detail later on. Note also that, based on the considerations in Section 5.4.5, some good nodes in the network are appointed as highly trusted nodes.

Simulation execution. The simulation of a network proceeds in simulation cycles: Each simulation cycle is subdivided into a number of query cycles. In each query cycle, a peer$i$ in the network may be actively issuing a query, inactive, or even down and not responding to queries passing by. Upon issuing a query, a peer waits for incoming responses, selects a download source among those nodes that responded and starts downloading the file. The latter two steps are repeated until a peer has properly received a good copy of the file that it has been looking for[2]. Upon the conclusion of each simulation cycle, the global trust value computation is kicked off. Statistics are collected at each node, in particular, the number of

[2]

Network	# of good peers	60
	# of malicious peers	42
	# of pre-trusted peers	3
	# of initial neighbors of good peers	2
	# of initial neighbors of malicious peers	10
	# of initial neighbors of good peers	10
	# Time-to-live for query messages	7
Content Distribution	# of distinct files at good peer i	file distribution in [42]
	set of content categories supported by good peer i	Zipf distribution over 20 content categories
	# of distinct files at good peer i in category j	uniform random distribution over peer i's total number of distinct files
	top % of queries for most popular categories and files malicious peers respond to	20%
	top % of queries for most popular categories and files pre-trusted peers respond to	5%
	% of time peer i is up and processing queries	uniform random distribution over [0%, 100%]
	% of time pre-trusted peer i is up and processing queries	1
	% of up-time good peer i issues queries	uniform random distribution over [0%, 50%]
	% of up-time pre-trusted peer i issues queries	1
Peer Behavior	% of download requests in which good peer i returns inauthentic file	5%
	% of download requests in which malicious peer i returns inauthentic file	0% (varied in Section 5.7.3)
	download source selection algorithm	probabilistic algorithm (varied in Section 5.7.2)
	probability that peer with global trust value 0 is selected as download source	10%
Simulation	# of simulation cycles in one experiment	30
	# of query cycles in one simulation cycle	50
	# of experiments over which results are averaged	5

Table 5.1: Simulation settings

authentic and inauthentic up- and downloads of each node. Each experiment is run several times and the results of all runs are averaged. An experiment is run until convergence to a steady state is observed (to be defined in the descriptions of the experiments), initial transient states are excluded from the data.

The base settings that apply for most of the experiments are summarized in Table 5.1. The settings represent a fairly small network to make the simulations tractable. However, experiments with larger networks in some instances show that the conclusions continue to hold. That is, schemes that do well in a small setting, do proportionately as well as the network is scaled up. Also note that these settings describe a pessimistic scenario with a powerful adversary: Malicious peers connect to the most highly connected peers when joining the network (see Section 5.7.3), they respond to the top 20% of queries received and thus have a large bandwidth, they are able to communicate among themselves in most of the threat models, and they make up a significant fraction of the network in most of the experiments. Yet, the experiments indicate that the scheme works well in this hostile a scenario, and thus will also work in less hostile environments.

In Section 5.7.2, two different ways of choosing download sources from those nodes that respond to a query will be considered and their performance in one of the experiments will be examined.

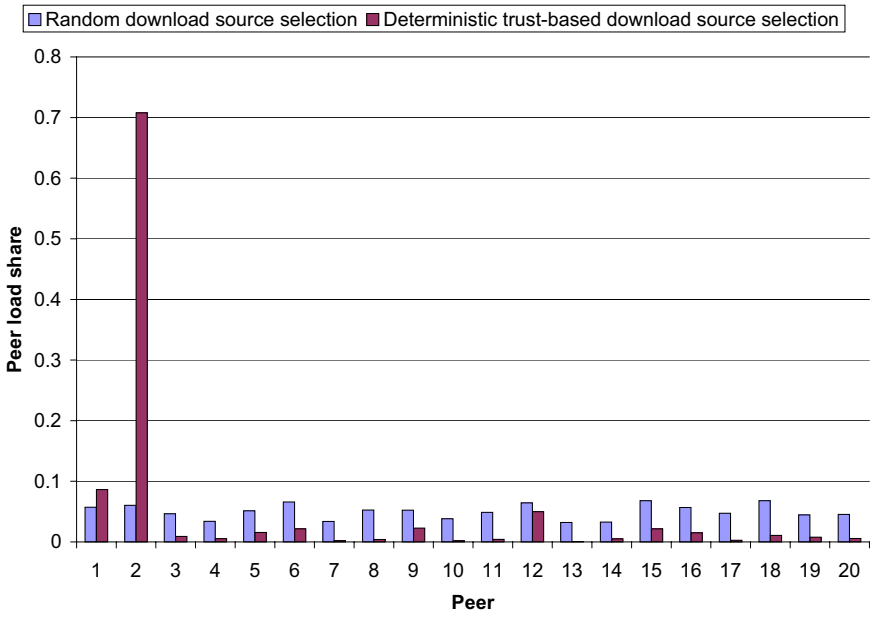

Figure 5.3: Load distribution in a network using deterministic download source selection versus a non-trust based network. The load distribution is heavily skewed, peer 2 will eventually accumulate all trust in the network.

As metrics, there is particular interest in the number of inauthentic file downloads versus the number of authentic file downloads: If the computed global trust values accurately reflect each peer's actual behavior, the number of inauthentic file downloads should be minimized.

Before the strengths of the scheme in suppressing inauthentic downloads in a P2P network are analyzed, it was examined if it leads to unwanted load imbalance in the network. In the following section, a precise definition on how we use global trust values in downloading files is given, too.

5.7.2 Load Distribution in a Trust-based Network

In P2P networks, a natural load distribution is established by peers with more content and higher bandwidth being able to respond to more queries and thus having a higher likelihood of being chosen as download source for a file transfer. In the scheme, a high global trust ranking of a peer additionally contributes to a peer's likelihood of being chosen as download source. Possibly, this might lead a peer into a vicious circle of accumulating trust by responding to many queries, thus being chosen even more frequently as download source in the future, thus accumulating even more trust. In a non-trust based system, this situation does not occur: From responding peers, a peer usually is randomly picked and selected as download source,

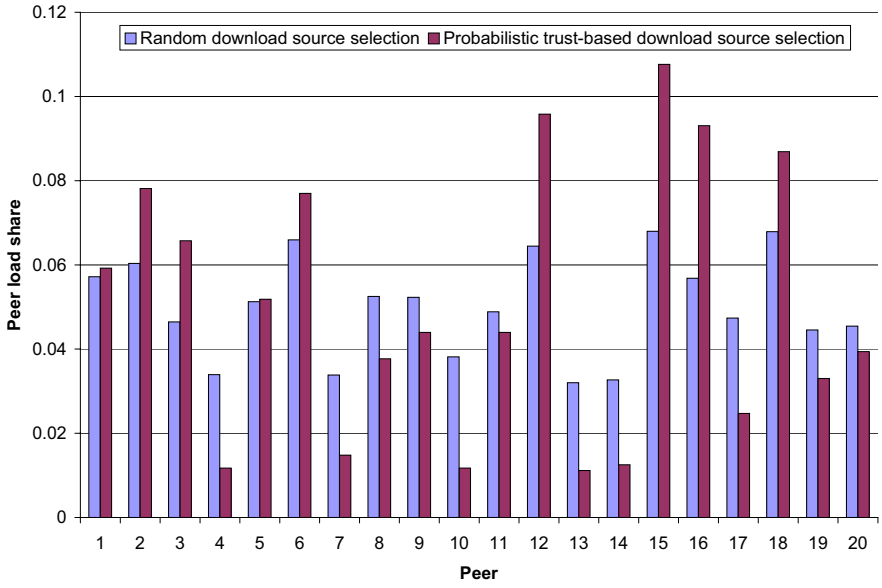

Figure 5.4: Load distribution in a network using probabilistic download source selection versus a non-trust based network. The load distribution does not deviate too much from the load distribution in a network based on random, non-trust based download source selection and is thus close to the natural load distribution in a normal Gnutella network.

somewhat balancing the load in the network. In the following, load-distributing randomization will be integrated into the scheme. In the experiment in Figures 5.3 and 5.4, the load distribution performance of a network in which the scheme is activated is studied. Two different trust-based algorithms for selecting download sources among peers responding to a query are considered, a deterministic algorithm and a probabilistic algorithm.

When $\{t_0, t_1, \ldots, t_{R-1}\}$ are the trust values of peers responding to a query:

Deterministic algorithm Choose the peer with the highest trust value t_{max} among the peers responding to a query as download source.

Probabilistic algorithm Choose peer i as download source with probability $\frac{t_i}{\sum_{j=0}^{R} t_j}$. With a probability of 10%, select a peer j which has a trust value of $t_j = 0$.

If a download returns an inauthentic file, delete the peer from the list of responding peers and repeat the algorithm.

To give new peers in the network – which start with a global trust value of 0 – the chance of building up trust, the probabilistic algorithm assigns a fixed 10% chance to download from the group of responding peers with trust value 0. Otherwise, new peers would maybe never

be chosen as download source, depriving them of the chance to become a trusted member of the network. Based on experience, a probability of 10% strikes a balance between granting malicious peers (which might also have a trust value of 0) too high a chance of uploading inauthentic files and allowing new peers to prove themselves as download sources of authentic files.

These download source selection algorithms are compared to a network where no trust system is deployed, i.e., among peers responding to a query a peer is picked as download source entirely at random. The load distribution in these networks is examined. There are no malicious peers in this experiment.[3]

Setup. A network consisting of 20 good peers is simulated, no pre-trusted peers and no malicious peers. Other than that, the standard settings in Table 5.1 apply. After running queries on the system for 20 query cycles, the load distribution is measured in Figures 5.3 and 5.4: For each peer 1 – 20 in the network, its load share is depicted, i.e., the fraction of its uploads after a full run of the experiment divided by the total number of uploads in the entire network. The load distribution in a network using the deterministic download source selection algorithm is compared to the load distribution in a network using no trust system at all in Figure 5.3, whereas a system employing the probabilistic download source selection algorithm is compared to the non-trust based network in Figure 5.4.

Discussion. Always choosing the responding peer with the highest global trust value as download source leads to a vast load imbalance in the network: Popular peers do not stop accumulating trust value and gain further popularity. In Figure 5.3, peer 2 will eventually become the download source for virtually all queries that it is able to answer. Also note that in each experiment that was run another peer turned out to be the most trusted peer. Choosing download sources probabilistically yields only a slight deviation in terms of individual load share of each peer from the case where no trust is used to select download sources among responding peers, therefore leading to a much better natural load distribution in the network. In Figure 5.4, peer 2 becomes the download source for 8% of all queries in the system, and many other peers participate in sharing the load, mainly determined by the number of and popularity of files the peers share. The measurements also show that the efficiency in suppressing inauthentic downloads does not vary between the two approaches. Thus, for the remaining experiments the probabilistic peer selection algorithm will be used.

[3] Malicious peers would not impact the load distribution among good peers since downloading peers keep trying until they have found an authentic copy of a file – hence malicious peers would add inauthentic uploads to the network, but not change anything about the number of authentic uploads from good peers.

Threat Model	File Upload Behavior	Local Trust Behavior	Figure
A	Always upload inauthentic files.	Assign trust to peers which upload inauthentic files.	5.5
B	Always upload inauthentic files.	Assign trust to previously known malicious peer to form malicious collective.	5.6
C	Upload inauthentic files in f% of all cases.	Assign trust to previously known malicious peer to form malicious collective.	5.7, 5.8
D	Upload authentic files.	Assign equal trust share to all type B nodes in the network.	5.9

Table 5.2: Threat models and associated experiments

5.7.3 Strategies for Malicious Peers

Now, the performance of the system in suppressing inauthentic downloads will be evaluated. Several strategies of malicious peers to cause inauthentic uploads even when the scheme is activated will be considered. In short, malicious peers operating under threat model A simply try to upload inauthentic files and assign positive trust ratings to any other malicious peer they get to interact with while participating in the network. In threat model B, malicious peers know each other upfront and deterministically distribute positive trust ratings among each other. In threat model C, malicious peers try to get some positive trust ratings from good peers by providing authentic files in some cases when selected as download sources. Under threat model D, one group of malicious peers in the network provides only authentic files and uses the trust they gain to boost the trust values of another group of malicious peers that only provides inauthentic files.

The experiments are started by considering the simplest threat model, where malicious peers are not initially aware of other malicious peers and simply upload inauthentic files.

Threat model A. Malicious peers always provide an inauthentic file when selected as download source. Malicious peers set their local trust values to be $s_{ij} = inauth(j) - auth(j)$, i.e., malicious peers value *inauthentic* file downloads instead of authentic file downloads.

Setup. A network consisting of 63 good nodes is simulated, 3 of which are highly trusted nodes, applying the standard settings from Table 5.1. In each experiment, a number of malicious peers is added to the network such that malicious nodes make up between 0% and 70% of all nodes in the network. For each fraction in steps of 10% the experiments are run and the results are depicted in Figure 5.5. Upon joining the network, malicious peers connect to the 10 most highly connected peers already in the network in order to receive as many queries travelling through the network as possible. In practice, P2P protocols such as the Gnutella protocol enable nodes to crawl the network in search of highly connected nodes. The experiments are run on a system where download sources are selected probabilistically based on the global trust values and on a system where download sources are chosen randomly from the set of peers responding to a query. Bars depict the fraction of inauthentic files downloaded in one simulation cycle versus the total number of files downloaded in the same period of time. The results are averaged over the last 10 query cycles in each experiment.

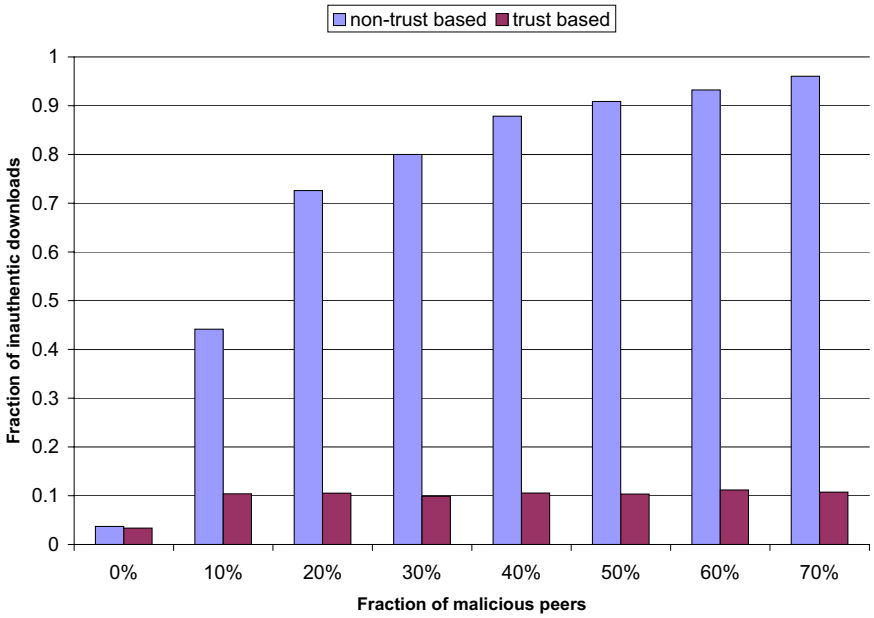

Figure 5.5: Reduction of inauthentic downloads by basing download source selection on global trust values in a network where independent malicious peers are present. Upon activation of the trust scheme, the number of inauthentic downloads in the network is significantly decreased to around 10% of all downloads in the system, malicious peers in the network are virtually banned from uploading inauthentic files.

Discussion. If no trust scheme is present, malicious peers succeed in inflicting many inauthentic downloads on the network. Yet, if the scheme is activated, malicious peers receive positive trust ratings only from other malicious peers, and even that only occasionally – since malicious peers have to happen to get acquainted with each other through a file exchange. Because of their low trust values, malicious peers are rarely chosen as download source which minimizes the number of inauthentic file downloads in the network. Mostly since good nodes make mistakes once in a while and upload inauthentic files (e.g., by not deleting a downloaded inauthentic file from their shared folders), a 10% fraction of inauthentic downloads can be observed. Even if no malicious peers are present in the network, downloads are evaluated as inauthentic in 5% of all cases – this accounts for mistakes users make when creating and sharing a file, e.g., by providing the wrong meta-data or creating and sharing an unreadable file.

However, note that due to the fact that the current secure algorithm uses majority vote to keep malicious peers from manipulating the global trust value computation, a *cooperating* fraction of more than 40% of malicious peers in the network will be able to influence the

assignment of trust values in the network during their computation. This is not represented in Figure 5.5 which assumes that the trust values are computed correctly. If malicious peers were able to manipulate the global trust value computation on a large scale in a coordinated effort, the number of inauthentic downloads would be higher when there are more than 40% peers in the network. Yet, in reality usually there will be less than 40% malicious peers in the network. Also, more efficient secure versions of the algorithm may be given (e.g., by having only peers with fair trust values participating in the global trust value computation).

Thus, in knowing that the scheme is present in a system, malicious peers know that they have to gain a somewhat positive trust rating in order to be considered as download sources. Therefore, strategies will be examined on how malicious peers can increase their global trust rating *despite* uploading inauthentic files.

Since malicious peers cannot expect to receive any positive trust ratings from non-malicious peers, they can try to gain some system-wide trust by teaming up as a so-called malicious collective. In the experiment depicted in Figure 5.6, the number of malicious peers in the network is varied to assess their impact on the network's performance when they are aware of each other and form a malicious collective.

Threat model B. Malicious peers always provide an inauthentic file when selected as download source. Malicious peers form a malicious collective by assigning a single trust value of 1 to another malicious peer in the network. Precisely, if M denotes the set of malicious peers in the network, each $peer_i \in M$ sets

$$s_{peer_i peer_j} = \begin{cases} 1 & \text{if } j = i+1 \\ 1 & \text{if } i = |M| \text{ and } j = 0 \\ 0 & \text{else} \end{cases}$$

which resembles a malicious 'chain' of mutual positive trust assessments. In terms of the probabilistic interpretation of the scheme, malicious peers form a collective out of which a random surfer or agent, once it has entered the collective, will not be able to escape, thus boosting the trust values of all peers in the collective.

Setup. The setup is equal to the setup in the previously described experiment, albeit with malicious nodes operating under threat model B. As shown in Figure 5.6, the experiments are run on a system where download sources are selected based on the global trust values and on a system where download sources are chosen randomly from the set of peers responding to a query.

Discussion. The system performs well even if a majority of malicious peers is present in the network at a prominent place. The experiment clearly shows that forming a malicious collective does not decisively boost the global trust values of malicious peers: These peers are

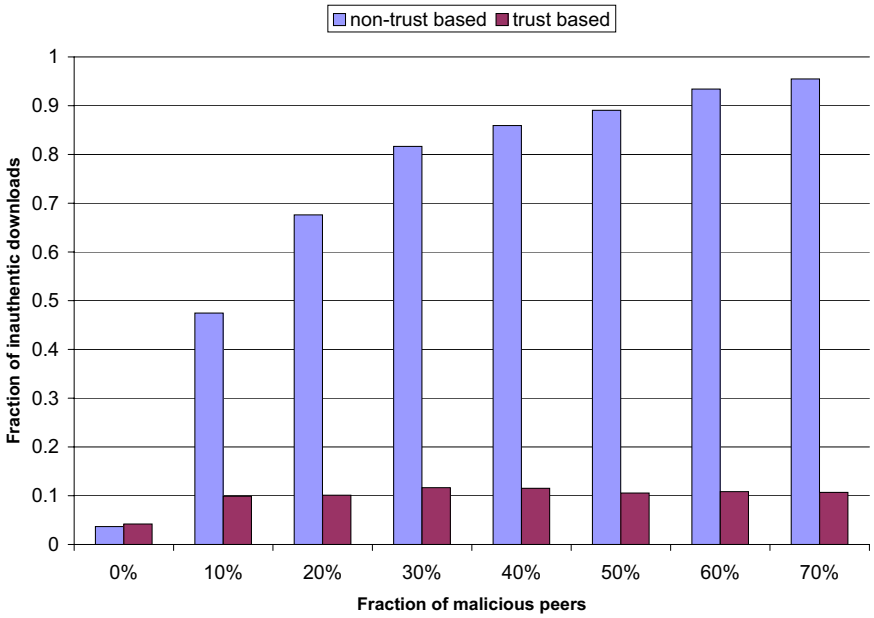

Figure 5.6: Trust-based reduction of inauthentic downloads in a network where a fraction of peers forms a malicious collective and always uploads authentic files. Forming a malicious collective does not boost the trust values of malicious peers significantly, they are still virtually banned from uploading inauthentic files, similar to Figure 5.5.

tagged with a low trust value and thus rarely chosen as download source. The system manages to 'break up' malicious collectives through the presence of pre-trusted peers (see Section 5.4.4): If pre-trusted peers were not present in the network, forming a malicious collective in fact heavily boosts the trust values of malicious nodes. Under the presence of pre-trusted peers, the trust ratings of malicious peers are significantly lower than those of good peers already after one simulation cycle. This minimizes the number of inauthentic downloads, and the numbers are virtually equal to the numbers in Figure 5.5 when peers do not form a malicious collective. For example, with 40% of all peers in a network being malicious, around 87% of all file downloads will end up in downloading an inauthentic version of the file in a normal, non-trusted network. Upon activation of the trust scheme, around 10% of all file downloads return an inauthentic file.

Forming a malicious collective obviously does not increase the trust ranking of malicious peers sufficiently in order for them to have impact on the network. This leaves malicious peers with one choice: They have to increase their trust ratings by receiving positive trust ratings from at least some good and trusted peers in the network. In the experiment in Figure 5.7, a

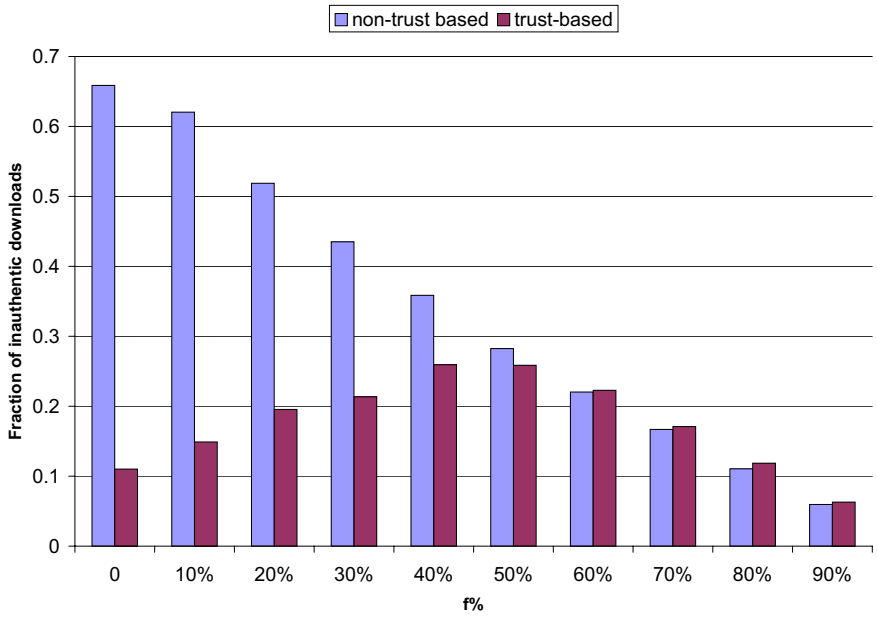

Figure 5.7: Trust-based reduction of inauthentic downloads in a network where a fraction of peers forms a malicious collective and returns authentic files with certain probabilities. When malicious peers partly provide authentic uploads, they receive more positive trust ratings and will be selected as download sources more often, also increasing their chances to upload inauthentic files. Yet, uploading authentic files may be associated with a cost for malicious peers.

strategy for malicious peers is considered that is built on the idea that malicious peers try to get some positive trust ratings from good peers.

Threat model C. Malicious peers provide an inauthentic file in $f\%$ of all cases when selected as download source. Malicious peers form a malicious collective as described above.

Setup. A network consisting of 53 good peers is simulated, 3 of which are pre-trusted peers, and 20 type C malicious peers applying the standard settings in Table 5.1. In each experiment, a different setting of parameter f in threat model B is applied such that the probability that malicious peers return an authentic file when selected as download source varies from 0% to 90%. Experiments are run for each setting of parameter f in steps of 10%. Running the experiments on both a non-trust based system and on the trusted system yields Figure 5.7. Bars depict the fraction of inauthentic files downloaded in one simulation cycle divided by the total number of files downloaded in the same period of time.

Discussion. Malicious peers that operate under threat model C attempt to gain positive trust ratings from some peers in the network by sometimes providing authentic files. Thus, they will not be assigned zero trust values by all peers in the network since some peers will

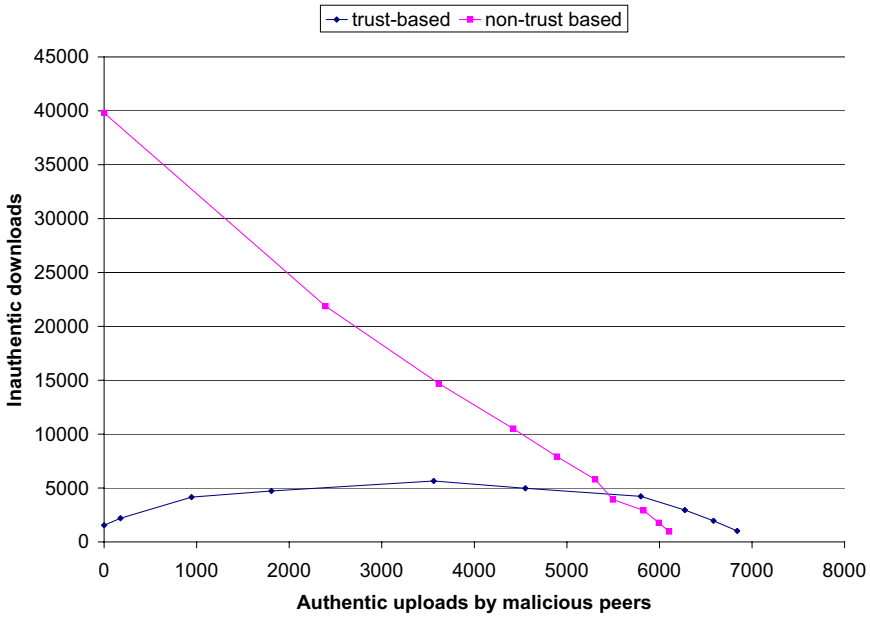

Figure 5.8: Inauthentic downloads versus authentic uploads provided by malicious peers with trust-based and non-trust based download source selection. When malicious peers provide authentic files in more than 20% of the cases when selected as download source, the increase in authentic files uploaded by malicious peers exceeds the increase in inauthentic downloads in the network, hence possibly coming at a higher cost than benefit for malicious peers.

receive an authentic file from them. This in turn provides them with higher global trust rankings and more uploads – a fraction of which will be inauthentic. Figure 5.7 shows that malicious peers have maximum impact on the network when providing 50% authentic files: 28% of all download requests return inauthentic files then. However, this strategy comes at a cost for malicious peers: They have to provide some share of authentic files, which is undesirable for them. First of all, they try to prevent the exchange of authentic files on the network, and in this strategy they have to participate in it; second, maintaining a repository of authentic files requires a certain maintenance overhead.

Figure 5.8 depicts the trade-off between authentic (horizontal axis) and inauthentic (vertical axis) downloads. Each scenario from Figure 5.7 is represented by one data point in Figure 5.8. For example, consider the fourth dark bar in Figure 5.7, corresponding to $f = 30\%$ and the trust scheme in place. In this scenario, malicious peers provide 1850 authentic downloads and 5000 inauthentic ones in a particular run.[4] The value $(1850, 5000)$ is plotted in Figure 5.8 as

[4] More precisely, 30 query cycles are run, the first 15 query cycles are excluded, and the number of inauthentic and authentic downloads is counted. After executing a second run, the numbers from both runs are added.

the fourth data point (left to right) on the lower curve, representing the case when the trust scheme is used. The points on each curve represent increasing f values, from left to right.

In Figure 5.8, malicious nodes would like to operate in the upper left quadrant, with as high as possible number of inauthentic downloads, and as low as possible number of authentic downloads. However, the file sharing mechanism in place constrains malicious nodes to operate along one of the curves shown. Without the trust scheme (top curve), malicious nodes can set f to a small value and move to the upper left quadrant. On the other hand, with the trust scheme, malicious peers have no good (for them) choices. In particular, increasing f beyond 20% does not make much sense to malicious peers since the incremental authentic uploads they have to host outnumber the increase in inauthentic downloads. Moreover, for all settings of parameter f below 50%, malicious peers will lose all positive trust assignments by other peers in the long run – since on average they do provide more inauthentic than authentic files.

The previous experiment has shown that malicious peers can increase their impact by partly concealing their malicious identity. Yet over time, their malicious identity will be uncovered and they lose their impact on the network. In the experiment in Figure 5.9, a team effort strategy is considered that malicious peers can use to work around this drawback. Two different types of malicious peers are present in the network: Malicious nodes of type B and of type D.

Threat model D. Malicious peers answer 0.05% of the most popular queries and provide a good file when selected as download source. Malicious peers of type D assign trust values of 1 to all malicious nodes of type B in the network. Precisely, if M_B and M_D denote the set of malicious type B peers resp. type D peers in the network, each $peer_i \in M_D$ sets

$$s_{peer_i peer_j} = \begin{cases} \frac{1}{\|M_B\|} & \text{if } peer_j \in M_B \\ 0 & \text{else} \end{cases}$$

Setup. A network is considered consisting of 63 good peers, 3 of which are pre-trusted peers, and 40 (39%) malicious peers, divided into two groups of malicious type B and type D peers. Otherwise, the standard settings from Table 5.1 apply. In each experiment, a different number of type B and type D peers is considered. Configurations considered are: I. 40 type B, 0 type D peers II. 39 type B, 1 type D peer III. 36 type B, 4 type D peers IV. 35 type B, 5 type D peers V. 30 type B, 10 type D peers VI. 25 type B, 15 type D peers VII. 20 type B, 20 type D peers VIII. 15 type B, 25 type D peers IX. 10 type B, 30 type D peers X. 5 type B, 35 type D peers. From left to right, these data points are plotted in a graph that depicts the number of inauthentic file downloads versus the number of authentic file uploads provided by malicious peers, as in the previous experiment.

Discussion. Malicious peers establish an efficient division of labor in this scheme: Type D peers act as normal peers in the network and try to collect global trust, which they will

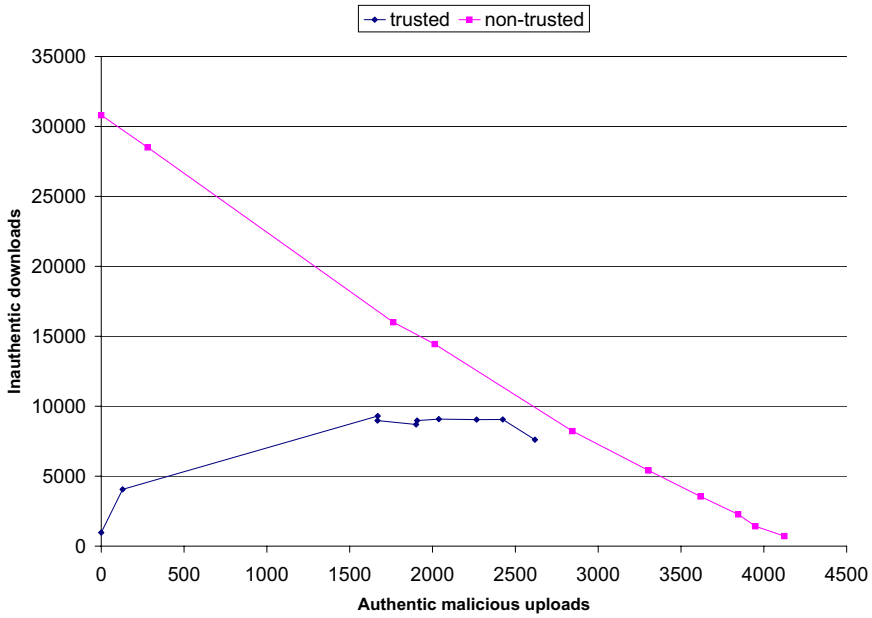

Figure 5.9: Inauthentic downloads versus authentic uploads provided by malicious peers with trust-based and non-trust based download source selection in a network populated by type D and type B peers. As with threat model C, malicious peers have to provide a number of authentic uploads in order to increase their global trust values. Yet, as compared to Figure 5.8, less authentic uploads by malicious peers are necessary to achieve equal numbers of inauthentic downloads in the network: 5000 inauthentic downloads cost 400 authentic uploads with this strategy as compared to more than 1000 authentic uploads with threat model C.

in turn assign to malicious nodes of type B providing inauthentic files. The malicious nature of type D peers will not be uncovered over time since these peers do not provide inauthentic files – hence they can continue to increase the global trust ratings of type B peers in the network. An interesting configuration for malicious peers would be configuration I: Malicious peers provide a fairly low number of authentic downloads (around 100), yet achieve almost the same number of inauthentic downloads in the network as in other configurations with a higher share of authentic downloads by malicious peers. In any configuration though, the trust scheme performs better than a system without trust-based download source selection. Also, this strategy would probably be the strategy of choice for malicious peers in order to attack a trust-based network: For example, by hosting 500 authentic file uploads in this strategy malicious peers achieve around 5000 inauthentic file downloads – as opposed to about 2500 inauthentic file downloads in the previous strategy, given the same effort on providing authentic uploads.

5.8 Related Work

Previous work in P2P reputation systems [6] [1] has all been based on similar notions of local reputation values. Reference [15] considers more complex pieces of information, yet does not specify any algorithms on how to actually compute reputation values of peers. [1] aggregates trust ratings from all peers in the network, but does not provide a solid theoretical model as the probabilistic model presented in this chapter. [40] aggregates trust ratings over several peers, yet no structured attempt to explicitly collect all ratings in the network is made.

The challenge for reputation systems in a distributed environment is how to aggregate the local reputation ratings s_{ij} without a centralized storage and management facility. While each of the previous systems cited above addresses this issue, each of the previous systems proposed suffers from one of two drawbacks. Either it aggregates the ratings of only a few peers and doesn't get a wide view about a peer's reputation, or it aggregates the ratings of all the peers and congests the network with system messages asking for each peer's local reputation values at every query. The system described here is able to consider all local experiences in a network and does so in a scalable manner.

5.9 Conclusion

A method has been presented to minimize the impact of malicious peers on the performance of a P2P system. The system computes a global reputation value for a peer by calculating the left principal eigenvector of a matrix of normalized local reputation values, thus taking into consideration the entire system's history with each single peer. It is also shown how to carry out the computations in a scalable and distributed manner. In P2P simulations, using these trust values to bias downloads has shown to reduce the number of inauthentic files on the network under a variety of threat scenarios. Furthermore, rewarding highly reputable peers with better quality of service incents non-malicious peers to share more files and to self-police their own file repository for inauthentic files. The algorithm is suitable for policing service provider behavior in a P2P Web of Services, too: Service providers which fake their service advertisements and cause damage to service consumers upon invocation of their service can be efficiently banned from the network. As in a file-sharing system, peers provide mutual ratings of their services and cooperate to compute a global reputation value for each peer, reflecting the trustworthiness of a service provider.

Chapter 6

Modeling P2P Networks

Assessing the performance of peer-to-peer algorithms such as topology construction protocols, distributed trust or search algorithms is impossible without simulations since testing new algorithms by deploying them in an existing P2P network is prohibitively expensive. However, some P2P algorithms are sensitive to the network and traffic models that are used in the simulations. In order to produce realistic results, P2P research therefore requires models that resemble real-world P2P networks as closely as possible. In this section, a model for P2P file-sharing networks shall be described and linked to measurements on existing P2P networks. The model has been used to simulate the reputation management algorithm described in Chapter 5.

6.1 Why P2P Research Needs Accurate P2P Network Simulations

Peer-to-peer research has encompassed promising work on algorithms in a variety of directions, including distributed protocols to construct efficient P2P network topologies, search algorithms for unstructured P2P networks, and algorithms to determine reputation of peers in a network, among others. Due to the decentralized nature and fast growth of today's P2P networks, actually testing such algorithms in a real-world environment by simply deploying them on an existing P2P network and collecting data on their performance is a daunting task. In some cases, measurements are easier to carry out due to some easily accessible central control entity in the network which, for example, manages node joins and departures [51]. Also, some algorithms may be tested by deploying them on one or a few controlled nodes in the network (as in [53]). However, for a wide range of P2P-related algorithms and protocols, simply deploying and testing them on existing P2P networks is not possible. For example, most algorithms require each peer in the network to implement the algorithm. Today's popular

peer-to-peer networks [21] have over 20,000 nodes, and performing a software update for each of these nodes in order to test each novel search algorithm is impractical. As another example, security protocols require testing under different threat scenarios such as an attack on the network by a coordinated group of malicious peers. Testing such protocols would require introducing malicious peers into the network. Thus, P2P algorithms and protocols are tested by simulation, under network models that attempt to mimic typical node interconnections, traffic patterns etc. Since algorithms and protocols are often sensitive to the traffic and network behavior, there is a clear need for accurate P2P network models.

Work in this area was mainly done on the fly during P2P research on a novel algorithm and mostly with simple models. For example, [1] assumes entirely random interactions among peers in a P2P network to test a P2P trust management protocol. In simulating a distributed search algorithm, [20] simply use a uniformly random location of files at and generation of queries by peers.

In the following, an approach to accurately model P2P networks shall be described. The focus will be on modeling a file-sharing network such as Gnutella [21]. Section 6.2 introduces the basic simulation concepts underlying the P2P network model presented in this work. Section 6.3 explains how content distribution in a P2P network is modeled. Section 6.4 describes properties of peer behavior in a network and in which ways they are part of the model. Section 6.5 discusses a model of the transport network on which a P2P network is built.

6.2 Basic Concepts

A typical P2P network will be considered: Interconnected, file-sharing peers are able to issue queries for files, peers can respond to queries, and files can be transferred between two peers to conclude a search process. When a query is issued by a peer, it is propagated by broadcast with hop-count horizon throughout the network (in the usual Gnutella way), peers which receive the query forward it and check if they are able to respond to it.

6.2.1 The Query-Cycle Model

Here, a simulation process that proceeds in query cycles is suggested. In each query cycle, a peer i in the network may be actively issuing a query, inactive, or even down and not responding to queries passing by. Upon issuing a query, a peer waits for incoming responses, selects a download source among those nodes that responded and starts downloading the file. The query cycle finishes when all peers who have issued queries download a satisfactory response. Statistics may be collected at each peer, such as the number of downloads and uploads of the peer.

6.2.2 Peer-Level Properties

The system-level dynamics of a P2P network are highly dependent on local, peer-dependent properties, such as the activity level or file-sharing behavior of each peer. In [29], different convergence behavior and different characteristic path lengths are observed in simulating a novel P2P network topology construction algorithm under two different models, one assigning bandwidth capacities to nodes based on a Zipf distribution, the other one based on a real-world distribution measured in [42].

Since the system-level dynamics of a P2P network – and hence the system-level impact of a P2P algorithm – is dependent on local, peer-level parameters, it is essential to accurately model these parameters. These parameters may be classified into two types: content distribution parameters, and peer behavior parameters.

Content Distribution. The volume and type of content each peer carries has to be modeled accurately. P2P networks are far from homogeneous in terms of type and volume of data shared, hence a model reflecting real-world P2P networks is required.

Peer Behavior. Also, peer behavior has to modeled accurately, including how a peer submits and responds to queries, how it chooses which query response to download, and its uptime and session duration.

In the next two sections, it shall be discussed how to accurately model the content distribution and peer behavior parameters.

6.3 Content Distribution Model

The dynamics of a P2P networks are highly dependent on the volume and variety of files each peer chooses to share. If few peers choose to share files, then queries are likely to be routed via many peers, and the load on the network referring to file uploads is likely to be highly imbalanced. If many peers choose to each share a wide variety of files, the network of peers who interact with one another is likely to be dense and unclustered, and query response times are likely to be quick.

Accurate assessment of the impact of intelligent query routing algorithms and content-based topologies depends on the accurate modeling of the volume and variety each peer shares. Furthermore, accurate modeling of the content shared by peers in the network provides greater insight into the file-sharing and communication patterns in the network, which is useful in many areas of P2P research.

6.3.1 Data Volume

In the model, each peer in the network shares a certain number of files.

Real-world observations. [42] has measured the probability distribution over the number of files shared by peers in Gnutella.

Model. This distribution is used to assign a number of shared files, F^i, to each peer i in the network. The absolute values from [42] are used.

6.3.2 Content Type

In this section, it will be described how to model the individual files each peer chooses to share. It is important to accurately model this because it will determine patterns of peers who interact with each other. A model in which the files peers share are chosen randomly is insufficient, as it will fail to produce clusters of peers that interact with on another, as has been observed [8]. Such properties affect the performance of many algorithms, including search algorithms [7] and reputation algorithms.

Real-world observations. In [8], it is observed that peers in a P2P network are in general interested in a subset of the total available content on the network. Furthermore, it is also observed in [8] that peers are often interested only in files from a few content categories. For example, in the domain of educational resources [34], users have a certain affinity towards learning materials related to the course of study they undertake.

It also has been observed in [26] that many document storage systems, including the WWW, exhibit Zipf distributions on the popularity of documents. This reflects the fact that some popular documents are very widely copied and held, while most documents are held by far fewer peers. The same can be said of content categories: there are some content categories (such as "Top 40 Hits" in the music domain) which are very popular and widely held, while most other categories (such as "Acid Jazz") are less widely held.

Model. The properties described above are modeled as follows. Briefly, peers are assumed to be interested in a subset of the total available content in the network, i.e., each peer initially picks a number of content categories and shares files only in these categories. Furthermore, it is assumed that files with different popularities exist within each content category, governed by a Zipf distribution. Files are assigned to peers at initialization in the following manner. According to the probabilistic model described below, each peer i is assigned some content categories C^i. Then, peer i is given an interest level for each content category $c \in C^i$. Finally, peer i is assigned files F according to its content categories and interest levels in those categories. In this model, each distinct file $f_{c,r}$ may be uniquely identified by the content category c to which it belongs and its popularity ranking r within that category. The probabilistic model is based on empirical observations of file distributions in [42] and [26].

Assigning content categories. The existence of n content categories $C = \{c_1, \ldots, c_n\}$ is assumed. Some content categories are more popular than others. That is, the files in some

content categories are more widely held than the files in other categories. This popularity is modeled by a Zipf distribution: when a peer is initialized, it is set to be interested in content category $c \in C$ with probability $p(c)$ given by

$$p(c) = \frac{\frac{1}{c}}{\sum_{i=0}^{n} \frac{1}{i}} \tag{6.1}$$

A peer is required to be interested in at least C_{min} content categories, repeating the peer's interest test until C_{min} categories have been chosen. The set C^i is the set of content categories that interest peer i.

Modeling interest level. A peer i interested in content categories C^i is probably not equally interested in all categories $c \in C_i$. Rather, peer i is more likely to be more interested in some categories than others. This is modeled by assigning an interest value w_c^i to each content category $c \in C_i$ of interest to peer i. This interest value is determined uniformly at random for each content category for each peer i. The fraction of files shared by peer i that are in category c is given by

$$p(c|i) = \frac{w_c^i}{\sum_{c' \in C^i} w_{c'}^i} \tag{6.2}$$

The number of files shared by peer i that are in category c is given by $F_c^i = p(c|i)F^i$.

Note that the interest value is not correlated with the general popularity of content category c. This reflects the fact that, while a certain category may be of interest to many peers (i.e., Top 40 hits), that category is not necessarily the main interest of those peers. Also note that since a steady-state network is assumed, the interests of peers do not change over time.

Modeling files. The individual files held by each peer have to be modeled. Each distinct file may be uniquely identified by the tuple $\{c, r\}$, where c represents the content category to which the file belongs, and r represents its popularity rank within content category c. This file is denoted $f_{c,r}$. Within each content category there are some files that are very popular, and some that are held by few people. This is modeled by a Zipf distribution as well. The fraction of files in content category c that are copies of file $f_{c,r}$ is given by:

$$p(f_{c,r}|c) = \frac{\frac{1}{r}}{\sum_{i=1}^{F_c} \frac{1}{i}} \tag{6.3}$$

where F_c is the number of distinct files in category c. Notice that in order to evaluate $p(f_{c,r}|c)$, the number of distinct files in each content category (see below) is required. The probability that a file f shared by peer i is a copy of file $f_{c,r}$ is given by the level of interest $p(c|i)$ that peer i has in category c times the popularity $p(f_{c,r}|c)$ of file $f_{c,r}$ within category c:

$$p(f_{c,r}|i) = p(c|i)p(f_{c,r}|c) \tag{6.4}$$

At initialization, files are assigned to each peer i based on this distribution and the number of files F_c^i shared by peer i in each category. Each peer stores the $\{c, r\}$ values for the files that it shares.

Modeling the number of distinct files per category. If there is maximum replication going on in the network, there are at maximum F_c^a files of content category c in the network, where F_c^a represents the number of files in category c shared by peer a, the peer who shares the most files in category c. On the other hand, if every single file on the network is distinct, then there are $p(c)F$ distinct files, where F is the total number of files on the network, and $p(c)$ is the fraction of files that are in category c. The truth probably lies somewhere in between, and it is set

$$F_c = dF_c^a + (1-d)p(c)F \tag{6.5}$$

where d is some number between 0 and 1. In the implementation used here,$d = .25$. Empirical evidence would be useful to determine an accurate choice of$d$.

6.4 Peer Behavior Model

In addition to content distribution, another primary factor in the system-wide dynamics of a P2P network is peer behavior, including peer uptime and session duration, peer activity levels, and how peers issue and respond to queries. These parameters affect the network in many ways. For example, frequently changing network participation (i.e., very short session durations, yet high uptimes of nodes) increases the administration overhead of topology construction protocols, which generally require communication among a number of peers to repair the network topology once a peer has left or joined, an important cost factor to consider in the design of such protocols. Second, node uptime represents the availability of storage space and computational power in the network. The pattern of node uptime in P2P networks is of interest for applications that wish to take advantage of these networks for large-scale computations, as in Section 5. If the network consists of a large pool of nodes that participate only infrequently and are down most of the time, those nodes that remain in the network have to sustain a higher work and storage load.

Query activity level is another peer behavior that is of particular interest to P2P research, as the query behavior of peers (in conjunction with the network's content distribution patterns) determines which peers interact with each other. These interaction patterns are of importance to the effective design of P2P algorithms ranging from search algorithms to file indexing protocols.

6.4.1 Uptime and Session Duration

Participating nodes frequently leave and re-join a P2P network, and a peer's uptime is defined to be the fraction of an observation period that a peer is participating in the P2P network, i.e., issuing, responding to and forwarding queries.

Real-world observations. Observations on the MojoNation P2P network [51] have revealed that up to 84% enter the network one time, and for less than one hour. At the moment, these peers are not considered in the simulations. They probably do not contribute to the shared data much, and they probably do not issue too many queries.

Model. A pool of N peers is assumed. Peers join the network in the simulations in a Poisson process [36]: At every time unit, the number of nodes joining the network is$x$ with probability

$$p(N_{join} == x) = \frac{\exp^{-\lambda} \cdot \lambda^x}{x!} \tag{6.6}$$

Each node is assigned an uptime, which is the number of time units the node will spend in the network: Here, an exponential distribution is used.

$$p(t_{up}(node_i) == t) = \mu \cdot \exp^{-\mu t} \tag{6.7}$$

These distributions are also used in [10] and [29] to describe the dynamics of a P2P network. Parameters λ and μ determine to which number of nodes N the network will converge when it reaches its steady state.

6.4.2 Query Activity

Peers in a P2P network issue queries to search for downloadable files that match their interests. A peer's query activity determines the rate at which it issues queries when it is up.

Real-world observations. [30] has examined overall query rates in a P2P network, yet the distribution of query rates over peers has not yet been studied in more detail. An empirical study on the distribution of query rates of real-world P2P networks would be straightforward and very useful to the accurate modeling of the network.

Model. In the model, nodes generate queries based on a Poisson process. The query rate of each node is set upon initialization and is picked uniformly at random from an interval $\{rate_{min}, rate_{max}\}$. In each query cycle, equation $p(\#queries == x) = \frac{\exp^{-\lambda} \cdot \lambda^x}{x!}$ gives the probability that a node issues x queries, where λ is the node's query rate.

6.4.3 Queries

In the query cycle model, each active peer issues a query at each query cycle. The specific query that peer i issues is given by the model described below.

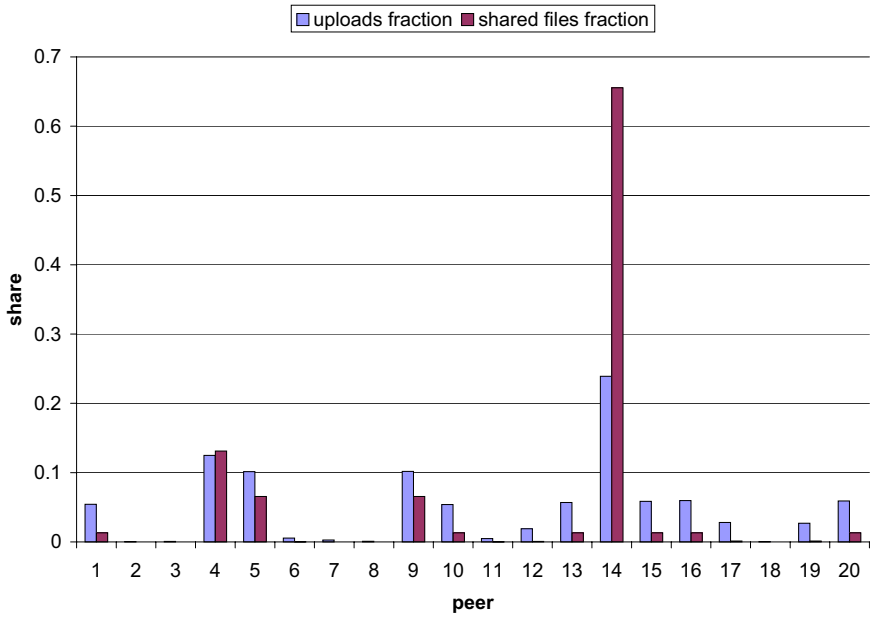

Figure 6.1: Share of uploads and files in a P2P network.

Real-World observations. Peers in general query for files that exist on the network and are in the content category of their interest. The first is true in large and diverse P2P networks, the latter can be claimed to be true for the majority of queries a peer issues, albeit it is yet to be shown by empirical studies on the query behavior in P2P networks.

Model. In the model, a query $q_{c,r}$ represents a query for the file $f_{c,r}$. A peer only issues queries in in the content categories in which it is interested. The probability that a peer$i$ generates a query $q_{c,r}$ is given by it's interest level in category c times the popularity of file r in c:

$$p(q_{c,r}|i) = p(c|i)p(f_{c,r}|c) \tag{6.8}$$

(The popularity $p(q_{c,r}|c)$ of a query $q_{c,r}$ is equal to the popularity $p(f_{c,r}|c)$ of its corresponding file $f_{c,r}$.) Also, a peer will not issue a query for a file that it already owns.

6.4.4 Query Responses

In this framework, modeling query responses is straightforward: If peer$i$ receives a query $q_{c,r}$, and it owns a copy of the corresponding file $f_{c,r}$, it responds to the peer that has issued the query and offers to upload the file.

6.4.5 Downloads

In the query cycle model, a cycle consists of each active peer issuing a query, waiting for the list of incoming responses, and downloading one of the responses. In the model, a peer randomly chooses a response to download. This may not accurately reflect reality. In fact, for a file-sharing system such as Gnutella, users tend to select peers with high bandwidth, hoping to being able to download a file fast. However, the model presented here can easily be extended to bias source selection based on peer bandwidth.

6.5 Network Parameters

Network parameters characterize the underlying transport network of a P2P network and the transport network related properties of peers.

6.5.1 Topology

Peers form an overlay network on top of a transport network. Upon joining the network, peers establish links to a number of peers in the network. When leaving, peers disband these links. Query and control messages are passed along the interconnection links between peers. As described in Section 4.3.3, it is assumed that peers organize into a power-law topology. In real-world P2P networks, it has been observed that the network topology is governed by a power-law [24].

To maintain a power-law based topology, a node join attractiveness is assigned to each node in the network which is based on a power-law:

$$p(p_{attractivity}(node_i) == k) = k^C \tag{6.9}$$

where C is close to unity, i.e., $C = -1$ [24]. When a node joins the network, it picks a node i currently in the network with probability

$$p = \frac{p_{attractivity}(node_i)}{\sum_{\forall j} p_{attractivity}(node_j)} \tag{6.10}$$

where the sum goes over all nodes currently in the network. Since the node join attractiveness values are assigned based on a power-law, the attractiveness of all nodes currently in the network is governed by a power-law, too. Thus, some nodes are contacted by nodes joining the network very frequently, while many other nodes seldomly are asked to integrate a new node. Such a network closely resembles real-world P2P networks. In Section 4.3.3, the origin of a power-law in P2P networks was claimed to be the fact that nodes tend to connect to highly connected nodes in the first place. However, it can be claimed that this is only the consequence of a deeper rooted power-law [16] which is associated with a general attractiveness

for nodes to become the target of joins, which is actually independent from the current node degree of a node: For example, the Gnutella P2P network has been shown to exhibit a power-law topology [24]. Highly attractive Gnutella nodes may be machines running in an IP subnet with many other P2P users. Since some Gnutella clients use broadcast on the local IP subnet to discover other Gnutella nodes, such nodes will become the target of joins frequently. This is precisely the reason why a non-changing node join attractiveness is assigned to nodes. Also, simply the fact that nodes have a long uptime increases the probability that they are connected to a high number of nodes, something that is also modelled by the simulation approach used here.

6.6 Discussion and Conclusion

The methodology described here has been successfully used to simulate the P2P reputation management algorithm presented in Section 5.

As a short example for how the scheme described here models peer interactions, Figure 6.1 depicts the load share in a sample network of 20 peers that was simulated based on the considerations above. One graph shows the number of uploads at a particular peer versus the total number of uploads in the system after 300 query cycles, the other graph shows the number of files shared by a peer versus the total number of files shared by all peers. Although the distribution of files is highly imbalanced – a property observed real-world P2P networks [42] – all peers participate in responding to queries, since even peers with only a few files have a fair likelihood of responding to queries for very popular files. This is a property that can also be observed on real-world P2P networks and provides an indication that the model presented here is somewhat accurate. More empirical studies are required to develop a network model for P2P networks that accurately reflects a network such as the Gnutella network. The efficiency of algorithms can only be compared if they can be run versus each other on commonly accepted problem sets or simulated on widely accepted models, an insight accepted in many other research domains such as Internet research [18].

Chapter 7

Conclusion

Algorithms for distributed discovery of Semantic Web Services and trust management in a peer-to-peer file-sharing network and a Web of Services have been described in this work. A general model for P2P file-sharing networks which uses experimental data to simulate real-world P2P networks has been built to test the performance of the algorithms.

The HyperCuP algorithm, a distributed topology construction algorithm for P2P networks, is capable of building binary hypercube topologies in a distributed fashion. Peers running the protocol automatically organize into a topology that features logarithmic diameter and node degree in a balanced state. The topology allows for efficient searching of peers. When using ontologies to specify the semantics of information stored or services offered by a peer, the addressing scheme of the topology can be used to enable even more efficient ontology-driven search.

The algorithm for distributed trust management presented in this work aggregates an entire P2P network's experience with any peer on the network into a trust rating for the peer. Building on the HyperCuP topology, trust values can be computed in an efficient and distributed manner, not requiring any centralized certification or trust authority. In experiments, the algorithm has been shown to greatly reduce the number of downloads of inauthentic files from malicious peers trying to subvert the network.

The algorithmic building blocks presented in this work can be used to build a network of Semantic Web Services, serving as an infrastructure for sophisticated future algorithms aiming at providing and composing complex Web Services.

Bibliography

[1] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the 10th International Conference on Information and Knowledge Management (ACM CIKM)*, New York, USA, 2001.

[2] S. B. Akers and B. Krishnamurty. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, April 1989.

[3] B. Biebow and S. Szulman. TERMINAE: A Linguistics-based Tool for the Building of a Domain Ontology. In *Proceedings of EKAW-99*, pages 49–66. LNAI, 1999.

[4] C. Law and Kai-Yeung Siu. Distributed Construction of Random Expander Graphs. 2002. draft available at http://perth.mit.edu/ ching.

[5] Christopher R. Palmer and J. Gregory Steffan. Generating Network Topologies That Obey Power-laws. In *Proceedings of GLOBECOM '2000*, November 2000.

[6] F. Cornelli, E. Damiani, S. D. C. D. Vimercati, S. Paraboschi, and S. Samarati. Choosing Reputable Servents in a P2P Network. In *Proceedings of the 11th World Wide Web Conference*, Hawaii, USA, May 2002.

[7] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In*Proceedings of the 28th Conference on Distributed Computing Systems*, July 2002.

[8] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks. Submitted for publication 2002.

[9] DARPA Agent Markup Language and its repository of domain ontologies. www.daml.com. 2002.

[10] David Liben-Nowell and Hari Balakrishnan and David Karger. Observations on the Dynamic Evolution of P2P Networks. In *Proceedings of IPTPS 02*, February 2002.

[11] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In *R. Meersman et al. (eds.), Semantic Issues in Multimedia Systems*, Boston, 1999. Kluwer Academic Publisher.

[12] Don Box and David Ehnebuske. Simple Object Access Protocol 1.1 W3C Note. 2000. available at http://www.w3.org/TR/soap.

[13] eBay website. www.ebay.com.

[14] Erik Christensen and Francisco Curbera and Greg Meredith and Sanjiva Weerawarana. Web Services Description Language 1.1 W3C Note. 2001. available at http://www.w3.org/TR/wsdl.

[15] L. Eschenauer, V. D. Gligor, and J. Baras. On Trust Establishment in Mobile Ad-Hoc Networks. Submitted for publication 2002.

[16] A. Fabrikant, E. Koutsoupias, and C. H. Papadimitriou. Heuristically Optimized Trade-Offs: A New Paradigm for Power Laws in the Internet. In *Proceedings of STOC 02*, 2002.

[17] R. E. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[18] S. Floyd and E. Kohler. Internet Research Needs Better Models. In *Proceedings of HotNets-I*, October 2002.

[19] Frank Leymann and et al. Business Process Execution Language for Web Services 1.0. 2002. available at http://www.ibm.com/developerworks/library/ws-bpel/.

[20] M. J. Freedman and R. Vingralek. Efficient p2p lookup based on a distributed trie. In *1st International Workshop on P2P Systems*, 2002.

[21] Gnutella website. www.gnutella.com. 2002.

[22] Google, a commercial search engine. www.google.com. 2002.

[23] S. L. Johnsson and C.-T. Ho. Optimum Broadcasting and Personalized Communication in Hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, September 1989.

[24] M. A. Jovanovic, F. S. Annexstein, and M. A. Berman. Modeling P2P Networks Through Small-World Properties and Power-laws. In *Proceedings of the IX Telecommunications Form Telcor*, Belgrade, Yugoslavia, November 2001.

[25] KaZaA website. www.kazaa.com.

[26] R. Korfhage. Information Storage and Retrieval. John Wiley, 1997.

[27] Larry Page. The Google Founder Talking at BASES, Stanford University. April 2002.

[28] H. Levesque, R. Reiter, Y. Lesperance, and R. Scherl. Golog: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, (31):59–84, 1994.

[29] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In*1st International Workshop on P2P Systems*, 2002.

[30] E. P. Markatos. Tracing a large-scale P2P system: An hour in the life of Gnutella. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[31] D. Martin and et al. DAML-S: Semantic Markup for Web Services. 2001. White paper, available at www.daml.org/services/daml-s.

[32] S. McIlraith, T. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems. Special Issue on the Semantic Web*, 16(2):46–53, March/April 2001.

[33] Napster website. www.napster.com. 2002.

[34] W. Nejdl and et al. EDUTELLA: A P2P Networking Infrastructure based on RDF. In *Proceedings of the 11th World Wide Web Conference*, May 2002.

[35] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[36] G. Pandurangan, P. Raghavan, and E. Upfal. Building low diameter P2P networks. In *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science*, 2001.

[37] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *Proceedings of 1st International Semantic Web Conference*, Sardinia, Italy, June 2002.

[38] P. Ramanathan and K. Shin. Reliable Broadcast in Hypercube Multicomputers. *IEEE Transactions on Computers*, 37:1654–1657, 1988.

[39] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, August 2001.

[40] Rita Chen and William Yeager. Poblano - A Distributed Trust System for P2P Networks. 2002. available at www.jxta.org.

[41] S. S. S. Ratnasamy and I. Stoica. Routing Algorithms for DHTs: Some Open Questions. In *Proceedings of 1st International Workshop on P2P Systems*, March 2002.

[42] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[43] I. Stoica and et al. Chord: A scalable P2P lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, August 2001.

[44] Sun Microsystems. JXTA Programmer's Guide. 2002. available at www.jxta.org.

[45] Sun Microsystems. Project JXTA: An open, innovative collaboration. 2002. available at www.jxta.org.

[46] M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications.*Knowledge Engineering Review*, 11(2), 1996.

[47] S. Vadhan. Lecture 4 - Randomness in Graph Algorithms. In *M.I.T. Script CS225 - Pseudorandomness*, Cambridge, MA, USA, February 2002.

[48] VBS.Gnutella Worm. securityresponse.synmantec.com/avcenter/venc/data/vbs.gnutella.html.

[49] W3C. XML. available at http://www.w3.org/XML.

[50] White paper. UDDI. 2002. available at www.uddi.org.

[51] B. Wilcox-O'Hearn. Observations on the Dynamic Evolution of P2P Networks. In*Proceedings of IPTPS 02*, February 2002.

[52] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. Submitted for publication, available at http://dbpubs.stanford.edu:8090/pub/2002-13. 2002.

[53] B. Yang and H. Garcia-Molina. Efficient Search on P2P Networks. In *Proceedings of International Conference on Distributed Computing Systems*, July 2002.