Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Kommunikationsnetze
Prof. Dr. P. Tran-Gia

Performance Challenges and Optimization Potential of Peer-to-Peer Overlay Technologies

Simon Oechsner

Würzburger Beiträge zur
Leistungsbewertung Verteilter Systeme

Bericht 02/10

Würzburger Beiträge zur Leistungsbewertung Verteilter Systeme

Herausgeber

Prof. Dr. P. Tran-Gia
Universität Würzburg
Institut für Informatik
Lehrstuhl für Kommunikationsnetze
Am Hubland
D-97074 Würzburg

Tel.: +49-931-31-86630
Fax.: +49-931-31-86632
email: trangia@informatik.uni-wuerzburg.de

Satz

Reproduktionsfähige Vorlage vom Autor.
Gesetzt in LaTeX Computer Modern 9pt.

ISSN 1432-8801

Performance Challenges and Optimization Potential of Peer-to-Peer Overlay Technologies

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Julius–Maximilians–Universität Würzburg

vorgelegt von

Simon Oechsner

aus

Würzburg

Würzburg 2010

Eingereicht am: 31.05.2010
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr.-Ing. P. Tran-Gia
2. Gutachter: Prof. Dr. B. Stiller
Tag der mündlichen Prüfung: 28.07.2010

Acknowledgements

This monograph is the distilled essence of over five years of research. It cannot fully convey the history of the numerous small and large setbacks as well as successes during that time, nor does it capture the circumstances under which it was written. Therefore, I would like to acknowledge here the people that have made it possible with their support and help.

First of all, I owe many thanks to my advisor Prof. Phuoc Tran-Gia, who gave me the opportunity to enter the world of science and has guided me there. His untiring support and personal commitment made this thesis possible. Due to his efforts, I was allowed to gather experience in industry and EU projects, and was able to visit international conferences in faraway countries.

I would also like to thank Prof. Burkhard Stiller, who not only acted as a reviewer of this thesis, but also as the leader of a project in which much of the scientific work described here was done. Furthermore, I am indebted to Prof. Reiner Kolla and Prof. Alexander Wolff in their role as examiners at my defense, not the least for their patience in arranging a date for it.

The fact that I could concentrate on the work for this thesis is in no small part a feat of our chair secretary, Mrs. Förster, who valiantly shields us from most of the bureaucratic tasks. My thanks also go to the many project partners at Siemens, Nokia Siemens Networks, DATEV, ATG and BSI I had the pleasure of working with, and especially the partners from the SmoothIT project. Apart from them, I would also like to thank Melanie Brotzeller, Jochen Prokopetz, Florian Metzger and Michael Lang, who worked on their diploma theses under my supervision and thereby supported me in my project tasks.

One of the important benefits of being a part of our chair is the opportunity to

work with great colleagues who are always there for discussions, solving seemingly unsolvable problems or simply giving support. This is especially true for Tobias Hoßfeld, who not only was a constant source of ideas for me due to his insatiable scientific curiosity, but who also provided invaluable professional as well as personal advice in countless talks and trained me to run a marathon. In the same vein, I would like to thank Andreas Binzenhöfer, who was (and still is) a role model for me both as a scientist as well as a person. In the last few years, I worked especially close with Frank Lehrieder and Dirk Staehle, and benefited very much from their experience, knowledge and readiness for discussion. Rastin Pries gave me valuable feedback on this work and significantly improved my skiing style. Thanks also go to Thomas Zinner, without whom our videos would still be low-quality, and Robert Henjes for being an ever-helpful office mate. Barbara Staehle not only is a valued colleague, but also our environmental conscience. Apart from them, I thank Michael Duelli, Matthias Hartmann, Klaus Heck, Matthias Hirth, David Hock, Michael Jarschel, Alexander Klein, Dominik Klein, Andreas Mäder, Rüdiger Martin, Michael Menth, Jens Milbrandt, Oliver Rose, Daniel Schlosser, Kurt Tutschku and Florian Wamser for the typical discussions in the coffee room and generally for creating the great atmosphere at our chair.

Finally, I am especially grateful for the never-ending support and confidence of my parents Traudl and Richard Oechsner. They always encouraged me in my plans and backed me up whenever necessary.

Contents

1 Introduction

And now for something completely different.
Monty Python (1969 - 1983)

Even after more than 40 years, many Internet applications are still relying on the traditional, centralized client-server structure. Here, the server offers functionality used by the clients. This approach entails many benefits, such as a tight control over the service by its provider and good possibilities for traffic management through an intelligent placement of servers. Its disadvantages include high costs, problems with scalability and vulnerability to failures [96].

These disadvantages were one of the factors that played a role in the rise of Peer-to-Peer (P2P) architectures, where each participant can act as both client and server at the same time. Such systems build a logical network on top of the Internet by interconnecting hosts on application level. The term overlay is used for these networks to discern them from the underlay, i.e., the physical network. This approach enables application developers to quickly deploy new services such as Skype [88] without having to develop and invest into a complex infrastructure before.

There are currently three main overlay types that are widely used in the Internet. These are search overlays, file-sharing overlays and video-streaming overlays. The aforementioned Skype implements a Voice-over-IP (VoIP) service that enables millions of users to hold conversations over the Internet without needing the legacy telephone system. It utilizes a search overlay to keep track of a users location in the network and to establish contact between end hosts behind firewalls [45].

File-sharing is by the time of this publication the most significant application in the Internet, generating the largest share of consumer traffic [75, 77]. P2P file-sharing clients enable a user to download content while utilizing the upload bandwidth of all participating users. This makes the content distribution much more scalable than a client-server approach, since every new user in the system adds his own upload capacity. The most important examples are the BitTorrent [66] and eDonkey [35] protocols.

Video streaming is an application that gained enormous popularity recently and produces an increasing share of today's Internet traffic [75]. P2P overlays are used for video streaming for the same reasons as for file-sharing, but face higher demands. Therefore, they are not yet as wide-spread as file-sharing applications. However, protocols that evolved from file-sharing as well as new technologies have emerged and become popular. Services like SopCast [89] or PPLive [87] gain importance and utilize overlays to reduce load on streaming servers [49].

These three most popular and relevant applications of overlays serve different functions that complement each other. A search overlay can be used for the location of content which is then downloaded via a file-sharing overlay or watched using a video streaming overlay. This is reflected by the combination of all three mechanisms in clients like Vuze [91]. This software uses the BitTorrent protocol as a basis for both file download and video streaming, and builds a distributed search overlay using Kademlia [31].

Since all of these functionalities are provided by the set of all end users, they share some common features [41]. Overlays are distributed systems which typically continue to function even if part of the nodes comprising the overlay fail or leave the network. Each node in the overlay provides resources which are consumed by the other nodes. No node should be overloaded and provide an unfair amount of resources, so that the load is distributed. In general, the utilization of resources needs to be managed in some way, keeping the efficiency of the single client and the efficiency of the complete overlay in balance.

Another common feature of the described overlay types is their implementation in real applications and their usage under realistic conditions. As a result,

they face challenges that are not only of academic interest, but that stem from problems encountered during productive operation. Thus, the identification of these challenges and according optimization of the overlays leads to results that have practical relevance.

1.1 Contribution

In this monograph, we improve the understanding of the overlay types discussed above by conducting a performance evaluation of previously unconsidered, but practically relevant scenarios. To this end, we discuss for each of the predominant overlays the problems they are currently facing in their typical field of application.

We present novel approaches for their optimization with respect to these challenges, and evaluate how these changes affect the performance of the systems. This allows us to draw conclusions for the improvement of current overlay technologies. These lead to the design of systems which realize the shown optimization potential and lets us derive recommendations for their configuration. Thus, we cover the current state of the art with respect to the applied overlay architectures.

For search overlays, we extend existing approaches by evaluating the influence of system parameters on the peer load, and consequently on the search performance. We base this evaluation on an architecture built for efficient search while offering load distribution properties.

This enables the deployment of such a system for time-critical applications, e.g., in enterprise environments. We develop analytical tools to describe the query load on the participating nodes and show that a trade-off between fast searches and resource efficiency in terms of memory consumption exists. This trade-off can be tuned using the parameters evaluated in this work. As a result, we provide dimensioning guidelines for time-critical applications.

The relevant resource in file-sharing overlays is bandwidth. We conduct a performance evaluation to show that current approaches to reduce inter-domain traf-

fic are not incentive-compatible, since end users may suffer from a reduced application performance.

However, we show that efficient traffic management algorithms conserving inter-domain bandwidth exist and add an own mechanism that is complementary to existing methods. We describe the effects introduced by these algorithms into the overlay under realistic conditions. Again, the relevant traffic management mechanisms can be influenced by their parameters, allowing to choose how much fairness is sacrificed for bandwidth savings. This balances the effects for the providers and the end users.

Finally, we provide a combination of existing video streaming overlays and a new video codec to support peer heterogeneity. We investigate whether such a combination is feasible without a large additional management overhead, relying on local information and only adapting the peer behavior.

Our new overlay architecture is a promising solution for a streaming application usable by a large set of clients with different access to the Internet. We evaluate implementation alternatives for this new mechanism and judge its performance under varying conditions for the overlay. Our changes can be implemented without adding additional signaling overhead and negligible additional complexity.

1.2 Outline

This work is structured as follows. In Chapter 2, we consider search overlays and the specific issues of these systems as they are deployed in critical applications. After outlining these, we provide background information about the mechanisms of these overlays and review work in the same area as ours. Next, we describe our implementation of such a system, which serves as the basis for our performance evaluation. We develop an analytical model of this architecture based on queueing theory, which we use to derive results about the search duration in this search overlay and its dependency on parameters like the system size.

Chapter 3 covers file-sharing overlays and their traffic characteristics. For the

most common protocol, BitTorrent, we provide a description of its key mechanisms and give an overview on approaches for managing its traffic. We develop an algorithm complementary to the approaches from related work that is compared to the existing mechanisms. Next, we discuss the results of a simulative performance study for a range of different scenarios. Firstly, we observe the general behavior of the algorithms before evaluating their performance under conditions met in real BitTorrent overlays. Then, we conclude this evaluation by performing a study on the parameters of the mechanisms.

Video-on-demand streaming overlays, a special form of video streaming overlays, are the topic of Chapter 4. Here, we investigate the optimization of these systems for heterogeneous peer environments. To this end, we review certain video streaming architectures and a scalable video codec offering new possibilities in supporting different video qualities. Our approach of including this codec into existing overlay mechanisms is then presented. The evaluation of this adaptation is again conducted by means of simulation. After discussing the model and tools used for this study, results considering different load situations and comparing implementation alternatives are presented for heterogeneous overlays.

Chapter 5 summarizes our contributions and findings about the optimization potential of existing and widely-used overlay technologies. We conclude by discussing open questions and future avenues of research.

2 Search and Lookup Overlays

Looking? Found someone you have I would say, mm?
Yoda

Locating information is today a task as important as processing it. Especially since the Internet has become a resource not only for universities and big companies, but also for the end user, the amount of data freely accessible has seen an explosive growth [60]. The sheer volume of information published via websites, of content available for download in server-based or P2P Content Distribution Systems, or of user data in social networks, necessitates a way to search for a specific homepage, file, or entry. Without being able to selectively access this store of data, it becomes useless, as extensive as it may be.

This problem is exacerbated by the fact that even the data of a single service has to be stored in a distributed fashion. Service providers like Google [47], Yahoo [67], or Amazon [57] collect and manage enough user information to make it difficult to store it centrally on one physical machine. Thus, the complete data set is segmented and distributed among a number of nodes, with the aim to improve scalability and to be more cost efficient. As a consequence, an internal lookup system needs to be implemented within a single logical data structure or database to find the storage location of specific entries. This lookup system may form an additional physical layer, or it may just implement a logical interface on the machines comprising the distributed data storage system itself.

There are many possible implementations of such a search or lookup service. Architecturally, a centralized server is the simplest way to store indexes which map queries to data locations. However, such a system has to handle a potentially

high query load and has to store a lot of lookup data itself. Thus, in large-scale systems, measures are taken to distribute both the load and the set of lookup entries to several servers. This can be done in a hierarchical fashion, such as in the Domain Name System (DNS) [20], or in a flat but scalable structure, such as used by the popular BitTorrent file-sharing protocol, which uses an overlay as a distributed tracker alternative [91].

In this chapter, we discuss a group of overlays implementing such a flat structure that offers one single functionality, namely to query for a data entry. These so-called Distributed Hash Tables (DHTs) combine scalability with a guaranteed search success if the queried entry is stored in the overlay [41]. This is a significant improvement over earlier search overlays, which were based on random graphs and used query flooding mechanisms to locate entries [32]. In contrast, DHTs use a defined overlay structure to enable a deterministic routing of a query towards the storage location of the queried data, and are therefore the premier example for the group of so-called structured overlays.

Since the publication of the first DHT implementations in 2001, they have not only received considerable attention from academia, but are also implemented in widely used applications, like the aforementioned distributed tracker. This example also illustrates how search overlays and content distribution overlays, which we cover in other chapters, complement each other. With their content location functionality, DHTs can enable peers to join specific file-sharing or video-streaming networks to download content.

In the following, we first present some of the challenges that appear with the usage of DHT-based search overlays in Section 2.1. After introducing DHTs and illustrating the concept with selected popular examples, DHTs optimized for search efficiency and related work on this topic is covered in Section 2.2. Then, our first contribution is an own specific DHT architecture which is described in detail in Section 2.3. Next, we develop in Section 2.4 an analytical model describing the characteristics of this architecture with a focus on the main performance indicator, the search time. Specifically, we provide in Section 2.5 results for the first moments of the search time in dependency of important parameters such as

the system size. Thus, guidelines for dimensioning are derived as well. Another important aspect is the effort to conserve load distribution in the considered overlay. Here, we use a simulative approach to evaluate the system since analytical methods are not feasible. Finally, we conclude the chapter in Section 2.6 with a summary of all discussed issues.

2.1 Challenges in Structured Search Overlays

The search process in DHTs scales with the number of nodes due to the overlay structure and the deterministic placement of entries on the peers participating in the overlay. Most of the well-known architectures offer routing in $O(log(N))$ hops or a comparable length, where N is the number of peers [24–26, 31]. However, these are overlay hops, i.e., the transmission of a message from one peer to another. In overlays deployed in the Internet and comprising nodes from all around the world, one such hop may incur a long delay, since peers typically do not distinguish between an overlay neighbor in the same city or a neighbor on a different continent. Thus, even if the number of these hops is kept reasonably small for larger systems, a search process may take a long time [33, 41].

For some applications, this poses no problem. A search for tracker data in the DHT implementation of BitTorrent may take several seconds without having a significant effect on the download time of a file, which is in the range of minutes or even hours [36]. On the other hand, there are services which rely on fast search procedures to function properly. These services are mainly found in enterprise environments, such as a mobile service provider system. Here, a distributed mobile subscriber database is queried every time a subscriber changes its state or wants to access a specific service, sometimes more than once in the same operation [85]. A search taking more than a few milliseconds already violates the tight timing bounds common in mobile core systems. Here, it does not matter whether a DHT is used to implement the database itself or just a lookup layer of it. The same applies to user databases behind webshops, e.g., Amazon [57], where the customer satisfaction strongly depends on the correct and timely reaction to his

actions. Another example is the DNS service, where a replacement of the traditional server hierarchy by a typical DHT would offer several advantages, but is ultimately not viable because it is much slower in terms of response times [27].

For these time-critical applications, a special subgroup of DHTs provide a solution. These overlays forgo the complex routing of typical DHTs such as Chord [26] or Kademlia [31], and instead store the addresses of all peers in the overlay on every peer. This basically implements a full mesh, with the result that only one overlay hop is needed for each peer to reach the storage location of an entry. Still, the distribution of the data relies on a hashing function, so that these structures are called one-hop DHTs.

In this chapter, we focus on small to medium scale one-hop DHTs that are used for high-performance applications. Although, as explained above, a short search time is already part of their architecture, there are still some challenges that must be considered when implementing such a system. Studies on the viability of one-hop DHTs and their bandwidth overhead due to the higher effect of churn, i.e., the constant process of nodes joining and leaving the overlay, exist and are reviewed in the next section. With our work, we provide more insights into two additional topics of importance. The first is the internal node load increase in such an overlay, which results from the fact that queries have to be forwarded to and processed on more than one node. We analyze how this affects the performance of the system and which system parameters have an impact on this effect. Additionally, we show what additional effort needs to be taken in order to ensure load distribution even in small-scale systems.

2.2 Background and Related Work

In this section, we first explain the basic concepts of a DHT needed to understand the specific architecture we evaluate. To provide illustrating examples and to emphasize the similarities between different implementations, we present a selection of instances found in literature. An overview on the existing work on one-hop DHTs then provides the background for our work.

2.2.1 Distributed Hash Tables

A DHT is a hash-table data structure that is stored not on one physical machine, but on several. The links between the different parts of the data structure are formed by the overlay, i.e., logical, connections between the nodes or peers. We will now explain how a DHT overlay is formed and how it is able to route queries efficiently.

The basis of a DHT is a hash function that places both peers and items to be stored in the overlay within an identifier space, which is typically a range $[0; 2^m - 1]$ for $m = 128$ or $m = 160$ in literature. A commonly used collision-free hash function is SHA-1 [23]. Input to this function is a node identifier such as the IP address in case of a peer, and a unique document or file name in case of an item. Thus, each peer and each item is assigned a unique m-bit identifier or ID, cf. Fig. 2.1.

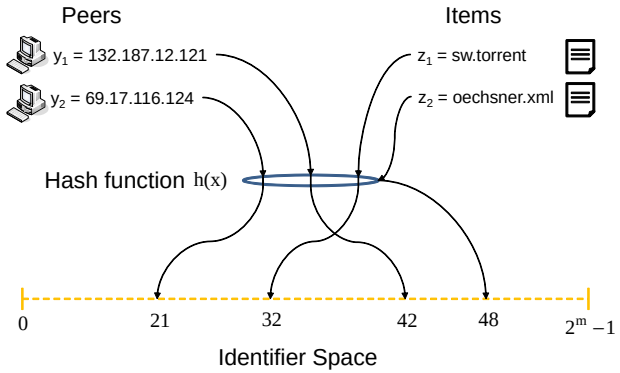


Figure 2.1: *Hashing peers and items into the identifier space*

The placement of the nodes introduces a logical separation of the identifier space and therefore can be used to implement consistent hashing [21]. The identifier space is separated into ranges or slots, with each node being responsible

for one such slot. This reduces the overhead of re-distributing items when nodes join or leave the system. Furthermore, it allows to map items to their responsible storage nodes.

A metric on the identifier space defines the peer closest to any given item identifier and thus the node where the item is to be stored. This may be the node with, e.g., the numerically closest identifier, or the node with the lowest identifier higher than the identifier of the item, also called successor node. Thus, searches can be routed deterministically to the node that is responsible for the requested entry. If it is stored in the overlay, it is stored on this node and can be returned. As a consequence, a search may return a false result only with a very low probability, which cannot be eliminated entirely because of the presence of node churn. Thus, a query may always get lost at a node going offline or may be resolved by a newly joined node that has not yet taken over all of the entries it is responsible for.

The node identifier also governs to which other nodes a node A maintains overlay connections. These connections are stored in an overlay routing or forwarding table and map the identifiers of A's neighbors to their addresses in the physical network, i.e., their IP addresses. The neighbors in a DHT can typically be separated into two groups, peers that are close in the identifier space, and peers a long distance away according to the used metric. The first group is the neighborhood of A in the identifier space, and is needed to forward queries along the last hops to their destination. In contrast, contacts in a remote part of the overlay serve as shortcuts through the structure and shorten the first phase of the search process, when a query only needs to be forwarded to the right area of the identifier space. The structure of a DHT is defined by the overlay metric and the specific rules which neighbors a peer should have. They differ for the example DHTs presented in the following, while the basic principles explained above remain the same.

The actual search procedure in a DHT follows a common pattern. If the requested entry is not already given in form of an overlay identifier by the searching application, it is hashed by the first node receiving the query to construct a valid identifier. This identifier or search key defines the destination of the query

routing process. As a consequence, DHTs only support exact search queries by default, i.e., searches where the full key of the requested item is known.

After constructing an identifier for the search, the initial node starts a lookup by forwarding the query according to the routing algorithm of the DHT. In general and for the examples described below, this means a node receiving a query forwards it to the neighbor whose identifier is closest to the requested key in terms of the overlay metric. When finally the node that is responsible for the requested key is reached, it either returns the stored value or answers negatively if no stored item exists for the key.

Chord

Chord was presented in [26] in 2001, and is one of the most referenced and theoretically evaluated DHTs. It is based on a ring structure that is formed by interpreting the identifier space of the underlying hash function as a circle $\bmod 2^m$, i.e., $2^m \equiv 0$. Items are stored on the node with the lowest ID succeeding the item's ID, i.e., its successor, cf. Fig. 2.2.

In Chord, each node A maintains connections to its immediate r successors, which form A's replication group. All items stored on A are copied on every node in this replication group, to prevent data loss should A and a number of its successors go offline in a very short time interval. A periodic stabilization routine ensures that each node knows its direct successor and resolves inconsistencies when new nodes join or old nodes leave the overlay.

A node could now already search for an item by simply sending a query along the chain of successors until it reaches the destination. This search, however, would not scale, since it takes on average $\frac{N}{2}$ hops in the default unidirectional design, where N is the number of nodes in the overlay. Therefore, each node additionally maintains a table of so-called fingers, which are peers at defined places in the overlay in relation to the local peer. The i^{th} entry in this finger table is the successor of the identifier $id_A + 2^{i-1}$, where id_A is the identifier of node A. This enables a node to forward a query at least half way towards its

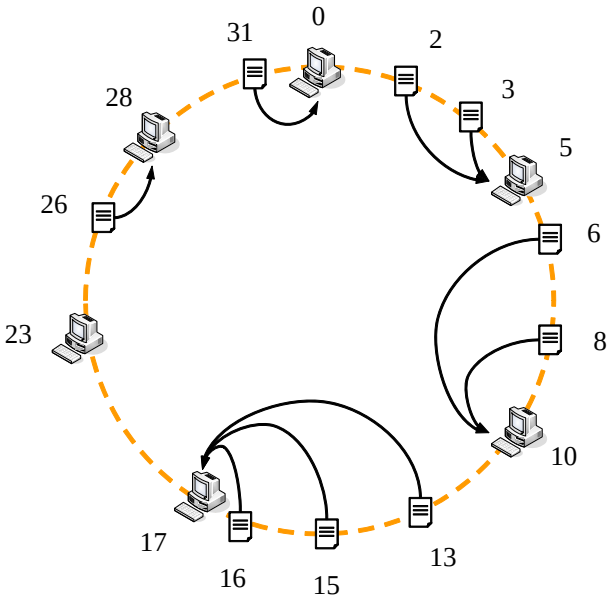


Figure 2.2: *Placement of items in a Chord ring*

destination with a single overlay hop, considerably shortening the search process and enabling a lookup in $O(log(N))$. With the successors and the fingers, each node has to store a routing table with a size also in the range of $O(log(N))$.

Pastry

Similar to Chord, Pastry [25] is based on a ring structure, with the identifier space forming the ring. The identifier space again is a range $[0; 2^m - 1]$, with $m = 128$. However, Pastry uses a different algorithm than Chord to select the overlay neighbors of a peer. Identifiers in Pastry are interpreted as encoded in a base 2^b, so that b consecutive bits of a peer identifier represent one digit. Each Pastry peer has

three different sets of neighbors: the routing table, the neighborhood set, and a leaf set.

The routing table is used in the first phase of the search process. Entries in the routing table of a peer A are selected according to the length of the common prefix their identifier shares with id_A. In row i of the routing table, contacts are stored that share a prefix of length $i-1$ digits with id_A, but differ in the i^{th} digit. In this row, column j holds entries where the i^{th} digit of the entry has the value j. Thus, there are $2^b - 1$ entries per row and $\lfloor \frac{m}{b} \rfloor$ rows in the routing table. Since the number of candidates per field in this table is limited for higher row numbers, not all fields do necessarily hold an entry.

The leaf set of A contains the 2^b peers that have the numerically closest identifiers to id_A, centered around A in the identifier space. Thus, it contains 2^{b-1} peers with an ID higher than id_A, and 2^{b-1} peers with a lower ID. It is used for the final routing step of a query.

Finally, the neighborhood set holds 2^{b+1} peers that are close to A with respect to the underlying topology, e.g., peers that have a short round-trip time to A. It is used in the initial configuration steps of a joining node and tries to ensure that short underlay connections are chosen for overlay links. It has no direct effect on the overlay routing procedure of Pastry.

A node forwards a query to a neighbor that shares a longer common prefix with the target ID. Thus, ideally the matched prefix grows at least by one digit with each step. If no node can be found that shares a longer prefix with the target ID, a neighbor is chosen that is numerically closer to the target. This can always be achieved via the leaf set entries. Finally, when a query reaches a peer which covers the target ID with its leaf set, this peer forwards the query directly to its destination, the node with the ID numerically closest to the target ID. With this routing procedure, messages in Pastry reach their target in $O(log_{2^b} N)$.

Kademlia

Kademlia [31] is a DHT that has a less rigid structure than the other DHTs described here. It utilizes the search requests themselves to stabilize the overlay and therefore saves signaling traffic. Additionally, query routing is more flexible and can utilize shorter paths with respect to the underlying network topology. Furthermore, queries can be issued in parallel along separate paths, so that both the search speed and reliability are improved. Although much less work was published on Kademlia than on Chord, it is much more widely used. One example is the distributed tracker feature of the BitTorrent Mainline [84] and Vuze [91] clients, which are both based on the Kademlia DHT.

The exact metric used in Kademlia to define the distance between two 160-bit identifiers x and y is the bitwise exclusive or (XOR) interpreted as an integer, i.e., $d(x, y) = x \oplus y$. Since the XOR metric is symmetric, neighbor relationships in a Kademlia overlay are symmetric as well. This is a difference to Chord, where a finger of a peer A does not usually have A in its own routing table. The symmetry property allows peers to use the origins of incoming queries as updates for their routing table. In addition, for any given identifier id_A and any given distance Δ, there is only one identifier id_B so that $d(id_A, id_B) = \Delta$ (unidirectionality). As a result, searches for the same identifier converge along the same path, with their origin being irrelevant. This allows for additional optimization via the caching of items.

The routing table of a Kademlia peer consists of so-called k-buckets. Each of these k-buckets holds up to k addresses of remote peers from a certain range of the identifier space. The i^{th} k-bucket of a peer A holds only entries B for which $2^{160-i} \leq d(id_A, id_B) < 2^{160-i+1}$. This means that the first k-bucket covers the distant half of the identifier space, the second a quarter, and so on. The last k-bucket contains the peer itself. Thus, the entries in the first k-buckets serve as shortcuts to quickly reach remote parts of the identifier space, while the neighborhood of the local peer is known in much more detail, since it is covered by more k-buckets, cf. Fig. 2.3. This allows for a routing in $O(log(N))$, similar

to Chord. However, in contrast to Chord or Pastry, a single routing algorithm suffices to reach the target.

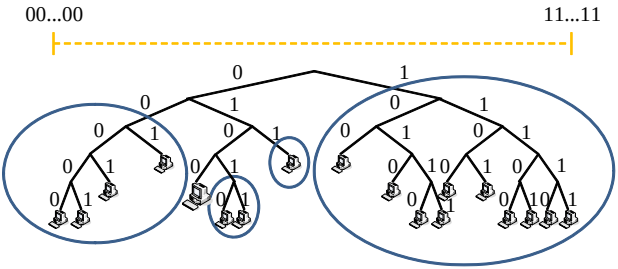


Figure 2.3: *Tree structure of Kademlia, ovals show subtrees where peer 0100 must have a contact*

If a new contact is discovered, the least recently seen entry in the according k-bucket is pinged. If it answers, the new contact is discarded. Otherwise, the old contact is assumed to be offline and the new peer is inserted. This algorithm is based on the experience that a node which was online for a long time is more probable to stay online as well.

CAN

A Content Addressable Network, or CAN [24] is the name commonly used for a specific implementation of a DHT that forms a d-dimensional Cartesian coordinate space on a d-torus as its structure. Peers are assigned a specific part of the coordinate space as their responsibility, their so-called zone, cf. Fig. 2.4. A peer stores all items that have coordinates within its zone.

The routing table of a peer A consists of all peers that have a zone bordering A's zone along $d-1$ dimensions. Since CAN forms a Cartesian coordinate space, a query can be forwarded in the direction of its destination based on the zones of the neighboring peers. The number of hops taken in a CAN overlay is in $O(dN^{1/d})$.

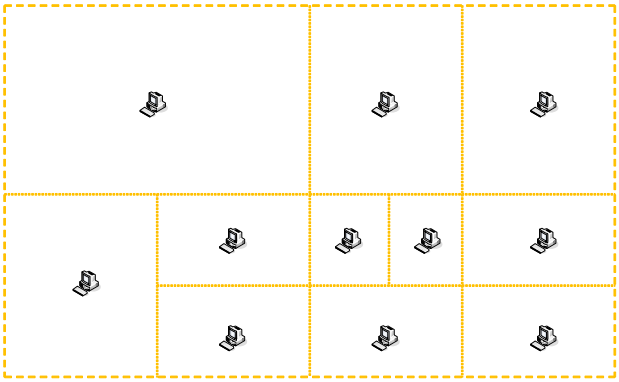


Figure 2.4: *Example of a 2-dimensional CAN network*

At the core of CAN is the algorithm which manages the zones. When a peer A joins the overlay, it selects a random point P and sends a query with that destination. This query is routed to the peer X that holds the zone in which P lies. Then, this zone is divided equally between X and A, along a dimension that is determined by earlier splits that led to the creation of X's zone. In case of a 2-dimensional CAN, zones are split alternating along the x-axis and the y-axis. All items that have to be stored on the new node, as well as the neighbors only belonging to the new zone are handed over by X.

When a node leaves the overlay, its zone has to be taken over by a remaining node. If a well-formed zone can be created by one of the neighbors, it takes over the orphaned zone. Otherwise, the neighbor with the smallest zone takes over. To increase the robustness and content availability, the overlay can be strengthened by implementing a number r of different realities. Each reality is a separate and independent coordinate space, with a single peer holding a different zone in each reality. Thus, items are stored redundantly r times, and nodes can choose in which reality to forward a query to circumvent routes blocked by churn. This also speeds up the search if a node selects close neighbors in terms of underlay proximity.

2.2.2 One-hop DHT Architectures

In contrast to the DHTs described in the last section, a one-hop DHT stores all peers in the overlay as neighbors in the routing table of each node. In this fully meshed overlay network, each node only needs a single hop to forward a query to its destination, hence the name one-hop DHT. Thus, the search process itself is greatly simplified and shortened. With routing being unnecessary, the structure of the overlay and its identifier space can also be kept simple. It is typically assumed to be the identifier ring as used by Chord or Pastry. However, since every node has to maintain a complete view on the overlay, the maintenance effort to keep all routing tables up to date is higher than for a multi-hop system. Every node has to be informed about every state change caused by node churn, which leads to a higher maintenance bandwidth consumption.

Still, the overlays considered here share the features of consistent hashing and self-organization under churn conditions with multi-hop DHTs. By knowing all other peers in the overlay, a node can locally determine on which peer a given item should be stored. Joining nodes still automatically take over items they have to store, while the data of leaving nodes can be kept in the overlay via redundancy mechanisms. In order to improve the load distribution for smaller systems, it is often discussed to map a number of virtual nodes to each physical node. This number can be tuned to the capacity of the physical node. As a side effect, this feature decreases the effort for single nodes during reorganization.

Related Work

A general analysis of the necessity of a multi-hop routing substrate in a DHT is presented in [37]. The authors argue that routing indirection only adds complexity to the system and should be avoided if it is not absolutely necessary. To judge when a multi-hop architecture is needed, a basic analysis of the necessary node bandwidths is conducted. The effort needed to keep the complete state up to date at one node in a one-hop DHT is compared with the data maintenance traffic needed in any case, e.g., for downloading content to new nodes that are

responsible for it. It is shown that only systems in the range of 10^7 nodes and larger need a multi-hop structure when realistic node lifetimes are assumed.

In [34], an architecture for a one-hop DHT is presented that scales to up to 10^5 nodes. The main indicator to determine the scalability is the bandwidth overhead needed to distribute state change information. This information has to be made available in the whole overlay since all nodes have to keep their forwarding table up to date in order to ensure a low query failure rate. To this end, the identifier space, which takes the form of a ring, is segmented into so-called slices. Each slice has one node selected as its slice leader. A slice is again segmented into units that have their own unit leaders. This structure is used to forward all state change information generated in a slice to the slice leader. This node collects all information within a certain time interval and then distributes the information to all other slice leaders. These can then forward the notifications to the nodes in their slices via the unit leaders.

An analysis in [34] shows how the query failure rate in this system depends on the length of the time interval in which state change information is collected, as well as the time it takes to determine a state change and to forward the information. The bandwidth overhead is also influenced by these times, but can additionally be influenced by the number of slices and units. The bottlenecks of the system with respect to the used bandwidth are the slice leaders, which carry the highest load and limit the scalability of the architecture.

The scalability properties of one-hop DHTs and their viability in comparison to multi-hop overlays are again discussed in [52]. Here, enterprise DHTs are identified as a major field of application, which shows different features in comparison to peer-to-peer end user systems. Thereby, the node lifetime and therefore the overlay stability is much higher in enterprise networks. The same is true for the available capacity, both in terms of bandwidth and processing. However, also the load on these systems is increased, as the number of queries per second is significantly higher than in an end user 'best effort' application. These observations and assumptions coincide with our own.

Similar to [34], the focus of the analysis provided in [52] is on the bandwidth used by the overlay. Here, both the traffic needed for topology maintenance and the query traffic is considered. It is concluded that one-hop DHTs scale to a few hundred thousand nodes. The operating points where a one-hop DHT is more efficient than a multi-hop DHT are identified in dependence of system size, node lifetimes, and lookup rates. Neither the node load nor a redundant storage of items in a single one-hop DHT are considered. However, an architecture for a system of several site-redundant one-hop DHTs as well as a global hierarchy of one-hop DHTs is proposed. The former provides resilience to network partitions by storing data fully redundant on several sites, while the latter circumvents high-latency hops which are an issue with other hierarchical DHTs.

A one-hop DHT is used as a central architecture element in SEATTLE [69], a proposal for combining the scalability of IP with the ease of configuration of Ethernet. It uses the DHT to store the mappings of Medium Access Control (MAC) addresses of hosts to their location, as well as the mappings of IP addresses to MAC addresses. Thus, broadcasting protocols such as ARP and DHCP are replaced with lookups in the DHT, making the architecture scalable. Switches are used as the nodes comprising the DHT, and consistent hashing is again used as the mechanism to assign items, i.e., mappings, to the responsible switch.

The evaluated systems are in the range of several hundred nodes in the DHT. The performance study focuses on the behavior of the system for different protocol parameters, such as the timeout of cached entries. Furthermore, the routing table size, i.e., the number of stored items per node, is considered. It is shown that SEATTLE scales much better than large-scale Ethernet and performs better than comparable approaches. However, the performance characteristics of the one-hop DHT are not considered in detail.

Dynamo [57] is an implementation of an architecture very much like the one analyzed in this chapter. It provides many services in the platform of the e-commerce company Amazon. It is used by applications that do not need a complex distributed relational database, but rather a simple key-value store such as provided by DHTs. Therefore, it uses a one-hop DHT as one design component

to provide load distribution and redundancy while keeping the search times short. However, the specific environment and service expectations of the Amazon online platform lead to a focus on availability additionally to guaranteed response times. Since Dynamo is used, e.g., to provide a shopping cart service, it is absolutely necessary that write actions succeed with a very high probability in order to prevent financial consequences and to keep the customer's satisfaction high. Consistency of the stored values is then achieved during read operations. The deployed Dynamo systems in a production environment typically contain several hundred nodes. While measurements of search latencies and load imbalances in a deployed system are presented, these only cover a very specific parameter set of a one-hop DHT architecture.

Improvements and implementation alternatives to the general one-hop DHT architecture are discussed in [57] as well. The possibility of allowing clients to address specific nodes in the overlay directly is raised. Thus, clients can determine the node that holds the desired data and query it directly. This circumvents even the single hop that is normally necessary in a one-hop DHT, but necessitates that clients have a detailed view on the distributed system and cannot treat it as transparent.

In contrast to the previous work described above, we focus on a more general view on the system, its parameters and how they influence the performance. For this, we do not focus on the bandwidth overhead, but take into account the query load on the nodes themselves, which is neglected in the related work. Our system differs from other architectures since it implements a lookup layer situated between the applications and the data storage. This, however, does not invalidate the derived results for the case where the application data is stored directly in the overlay, since the most important system characteristics are found in both cases.

2.3 Architecture of a Distributed Lookup Overlay

As discussed in Section 2.1, we focus on a high-performance distributed index based on a one-hop DHT. To this end, we develop and describe in this section our

own specific architecture suited for a relevant application, namely a distributed database, similar to Dynamo [57] but with different requirements. Thus, the system discussed below implements a lookup layer for a virtualized database which consists of a number of physical database servers. This means that the nodes in this lookup layer only hold pointers to the storage location of the real data. The one-hop DHT we develop an analytical model for is placed in this practically relevant setting, which we therefore outline in the following.

2.3.1 Layers of a Distributed Database

We consider a distributed database which stores a large number of entries, e.g., a subscriber database in a mobile network operator domain or a user database of a webshop. Due to the large amount of stored data, the database is distributed among several dedicated database servers, which form the back-end. To locate specific data entries in this back-end, a front-end layer offers the necessary lookup and forwarding functionality, i.e., the front-end resolves queries to the database and forwards them to the correct back-end server. This lookup layer stores pointers to back-end servers, one pointer for each database key. These pointers take the form of $< key, value >$ pairs, with the key being a reference to the original database key queried and the value holding the address of the back-end server where the associated database entry is stored, cf. Fig. 2.5.

The number of lookup entries which have to be stored in the front-end depends on the amount of data in the back-end. A complete lookup table may very well be several dozen Gigabyte in size, e.g., for a database of a national mobile service provider. With even larger databases and more keys, this value grows. This seems small if compared to hard disk space commonly available. However, the additional time consumed in the lookup layer should be kept to a minimum, which leads to this data being kept in the Random Access Memory (RAM) of the nodes. RAM, while becoming more and more inexpensive, is in these orders of magnitude still an important capital expenditure factor in the equipment of servers. Therefore, keeping the amount of data stored on a single node low is a

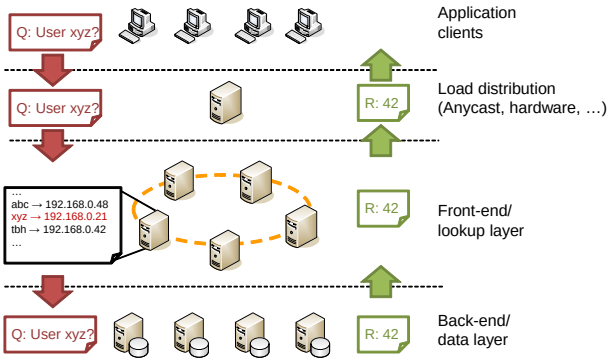


Figure 2.5: *The basic system architecture*

question of cost efficiency, and guidelines to system dimensioning are valuable.

Applications, in case of a subscriber database e.g., accounting or location management, issue queries to the database system, normally searching for one entry at a time. These queries may be Lightweight Directory Access Protocol (LDAP) messages or conform to other protocols suitable for accessing a database. Since the pointers stored in the front-end layer only change when new entries are added in the back end, or if entries are moved from one back-end server to another, the bulk of queries is expected to consist of read operations for the front-end, i.e., lookups for applications.

It would not be viable for external applications to know the internal structure of the database, such as the addresses of specific nodes in the lookup layer. Instead, the queries are sent to the system as a whole, e.g., in form of an anycast address like in the DNS architecture, or to one defined address which serves as a load distributor. Thus, application queries only see one virtual data storage system, while its internal complexity remains hidden.

This entry point into the database system does not need much logic and can therefore perform very efficiently. It forwards the queries to the front-end

servers of the system. Basic load distribution may be conducted before forwarding queries to the individual front-end servers. However, this work focuses on the lookup layer itself. Once the query is in the front-end layer, the database key it contains is resolved to the address of the back-end server holding the associated entry. Finally, the response with the queried data is forwarded again via the front-end to the application. This is done in order to necessitate only a single connection between the application and the database system, as well as to hide the internal complexity of the latter.

2.3.2 Implementation by a One-hop DHT

The lookup layer of the basic architecture described above is now implemented by a one-hop DHT. The interface of a DHT, namely the ability to store and retrieve pairs of keys and values, is perfectly suited to such a system. The original database search keys, e.g., LDAP keys, can be hashed to serve as hash table keys. The storage locations of the associated data sets are the associated values. The particular implementation details of our system are described in the following.

We consider a typical one-hop DHT with an identifier ring as its structure, similar to [34] or [57]. The lookup entries are stored in the DHT according to their keys, as described in Section 2.2. Each entry is stored on its successor node in the identifier ring. Additionally, it is stored on $R-1$ successors of that node to ensure redundancy and to prevent data loss, cf. Fig. 2.6. Thus, node A is primarily responsible for each entry falling in the ID range between A and its predecessor. Due to the redundancy, A can nevertheless resolve queries for all entries up to the responsibility range of its $R-1$ predecessors. Here, R is a tunable parameter that enables a trade-off between resource savings on one hand and system load and availability on the other hand, as we show in Section 2.4.

Thus, the normal system operation is as follows. An application issues a query to the database of which the lookup system is part of. These queries are distributed evenly among the lookup nodes, e.g., by means of a simple round robin load distributor. When a query reaches a lookup node A, this node first hashes

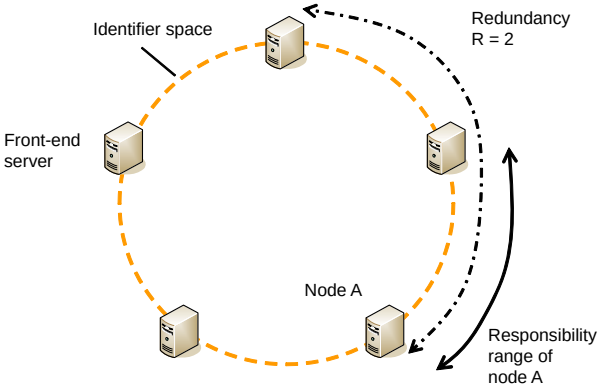


Figure 2.6: *The structure of the considered one-hop DHT front-end layer*

the database key of the query. Next, it checks whether it stores the lookup entry associated with this key locally, including the redundantly stored data. If this entry is not stored locally, the node determines which other node is responsible for it via its routing table, and requests the entry from that node. In either case, the original query is afterwards forwarded to the according back-end database server, whose address is the stored value of the $< key, value >$ pair of the lookup entry. Finally, the response to this database query, which contains the requested data itself, is forwarded to the application.

In case of an additional internal lookup, the node with the lowest ID higher than the hashed key searched for is always selected as the responsible node. This eases the routing process in case of node failures, and should still return valid results as long as no data loss has occurred, which is only possible if R consecutive nodes fail in a short time interval. While the query load for one entry could also be distributed among the R nodes storing that entry, we assume that each entry is queried with the same frequency, and therefore, no additional gain can be achieved by this measure.

As we have seen in Subsection 2.2.2, systems like this are deployed in reality with a size smaller than a few hundred nodes. This relatively small number of nodes in the front-end layer invalidates a common assumption for large-scale DHTs: the inherent load distribution. Since generally, nodes in a DHT are assigned their position in the identifier space of the overlay by a random hash function, only a large number of nodes lead to a nearly equal distribution. This in turn leads to an equal distribution of lookup entries to be stored on each node, and, assuming an equal probability for each entry to be queried, load distribution. Conversely, a low number of nodes is not necessarily positioned well-distributed in the ID space of the DHT by a random hash function, and therefore the query load may be skewed.

Moreover, the nodes storing more lookup data have to provide more memory. If it can not be predicted how the load will be distributed, all nodes have to be equipped with enough resources to handle the worst case, resulting in higher costs than necessary. One common way to circumvent this problem is to use multiple virtual nodes per physical node to reestablish an equal distribution of nodes in the identifier space.

Since we are interested in a system where the property of load distribution is achieved, we consider an alternative solution here that has the same effect but offers itself more to analysis. We assume the random overlay ID assignment known from most DHTs is replaced by a deterministic positioning of the nodes in the overlay, assuring that each node is responsible for the same amount of data. Thus, we can assume almost perfect load balancing for our analysis.

While the considered application of a critical database normally warrants the deployment of dedicated hardware with long Mean Time Between Failures (MTBF) intervals, it is nevertheless possible that a node or one of its components fails during normal operation. Node failures greatly upset the even load distribution in the system, leading to overload and/or congestion and should therefore be acted upon immediately. In order to keep the need for manual intervention low, an automatic reorganization algorithm is used to reassign the IDs of the remaining nodes, thus again placing them equidistantly on the identifier ring. Since this

also includes a change in the responsibility ranges for each node, a redistribution of the lookup data is necessary in this case.

Apart from the aspect of node failures, the data stored in a database changes over time. Generally, the amount of data grows, which means it should be easy to expand the system by adding new nodes, both to the back-end and to the front-end layer. The system should scale with the number of entries, and ideally, adding new nodes does not involve major manual configuration.

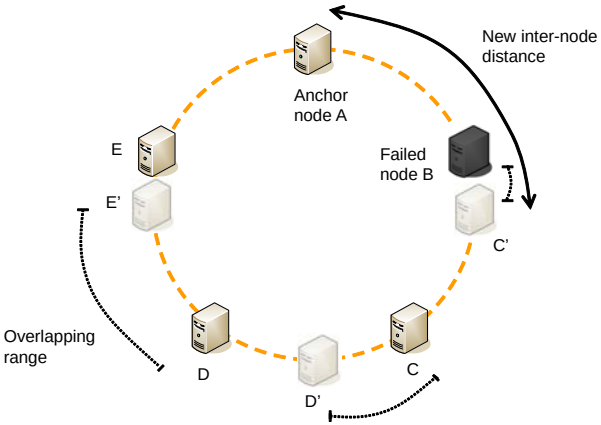


Figure 2.7: *Reorganization after one node failure*

We assume a simple heuristic is used to limit the amount of data which has to be transmitted over the network. This heuristic can cope both with node additions and with node removals or failures. Starting from an anchor node, each node is positioned in the correct distance (computed with the new number of active nodes) in the same order as before. Thus, there is a high probability that the old range of a node has a large overlap with its new range, meaning that the node already stores much of the data it needs in the new situation, cf. Fig. 2.7.

The same algorithm is used to add new nodes to the system, enabling an easy

expansion and scaling with increasing load. If more than one node is to be added at the same time, the new nodes are inserted equally distributed into the system, so that again the overlap between old and new responsibility ranges is high for the 'old' nodes.

2.4 Analytical Queueing Model

We now present our analytical model to evaluate the one-hop DHT architecture presented in the last section. It is based on a queueing model in order to consider the query load at the lookup nodes. With this model, we are able to analyze the search time spent within the one-hop DHT in dependence of two main parameters influencing the system behavior, which are the number of nodes in the system and the degree of redundancy. The number of nodes, i.e., the system size, is considered as an important parameter in the related work [34, 37, 52], while the redundancy is generally neglected in these studies. However, our results show that it has a significant influence. The search time is a performance indicator considered, e.g., in [57] and [33], and the most important parameter for the considered application [85].

2.4.1 Overall System Model

To evaluate the presented architecture, we employ the system model shown in Fig. 2.8. In this model, we make some simplifying assumptions. First, the processing times for queries are assumed to be independently and identically distributed (iid), i.e., we do not differentiate between the processing of external queries, internal queries, or response forwarding. However, we assume a larger variance in the service time to compensate for this. Second, we assume that the popularity of each database entry, i.e., the frequency with which it is queried, is the same. Our third assumption is that the aggregate traffic flows and query arrivals at each node constitute a Poisson process.

The total initial query load offered to the system is created by the applications connected to the database system. The query arrivals are assumed to follow a Poisson process with rate λ_0. We further assume an equal distribution of this load on each of the N nodes in the system, resulting from the basic load distribution described in the last section. Each node is modeled by means of a $M/GI/1$ delay system with an arrival rate λ^* and a processing time B with mean $E[B]$, cf. Fig. 2.8. The waiting time W is implicitly given by the node model and parameters. The query forwarding duration over the network is modeled by the random variable T_T for the transmission time distribution.

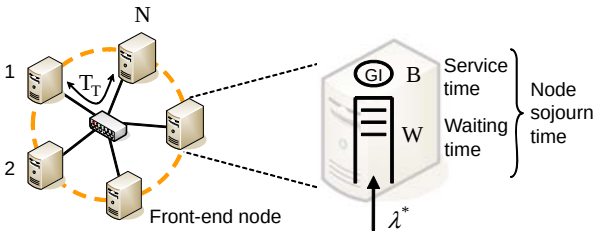


Figure 2.8: *System model*

The search procedure follows the phase diagram depicted in Fig. 2.9. The first lookup node is traversed in any case. With a probability $1 - p$, this node holds the lookup data for the query. Thus, the query can be directly sent to the backend and the response can be forwarded to the application (the upper path of the diagram). The probability $1-p$ depends on the share of lookup data a single node stores. Since we consider content replication to increase availability and perfect load distribution, each node stores a fraction of $\frac{R}{N}$ of all lookup entries, where R is the redundancy factor ($1 \leq R \leq N$). Therefore, the probability to find the queried data on the first node is $1 - p = \frac{R}{N}$ as well.

Consequently, with probability $p = \frac{N-R}{N}$ the query has to be forwarded to a different node if the local node does not store the lookup entry necessary to resolve the query. This leads to a hop within the overlay and an additional node

traversal as well (corresponding to the lower path in the diagram). The response from the second lookup node is then sent back to the initial node, taking a second hop. The initial node can then forward the application query to the database. Finally, when the database server sends a response it is again forwarded by the initial node to the application.

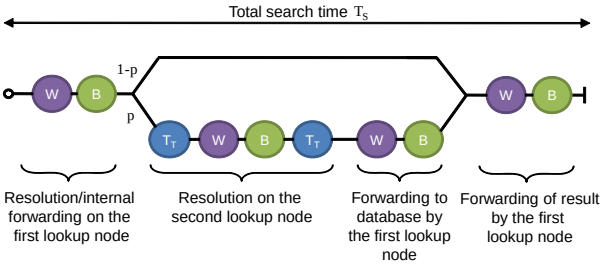


Figure 2.9: *Phase diagram for the search procedure*

An important aspect of the one-hop DHT under consideration is that not each query is resolved by the first node it encounters. Most of them will spawn subsequent internal queries. Therefore, the query arrival rate at one node must consider this additional internal load. Each node initially receives its fair share of application queries. With the total initial arrival rate being λ_0, and the system consisting of N nodes, this initial load at a specific node corresponds to $\frac{\lambda_0}{N}$, cf. Fig. 2.10. Of these queries, the node can only resolve a fraction $1 - p = \frac{R}{N}$ locally, as discussed above. The rest of the queries has to be resolved on a second node, and is consequently forwarded. This outgoing query flow is equally split among the $N - 1$ remaining nodes. Due to the symmetry of the traffic flows, the same amount of queries (with rate $p \cdot \frac{\lambda_0}{N}$) is received. This traffic is therefore added to the external query flow.

All internal queries are also answered, effectively doubling the internal traffic. This is due to the fact that the first node receiving the request is responsible for resolving the complete query and forwarding the response to the querying

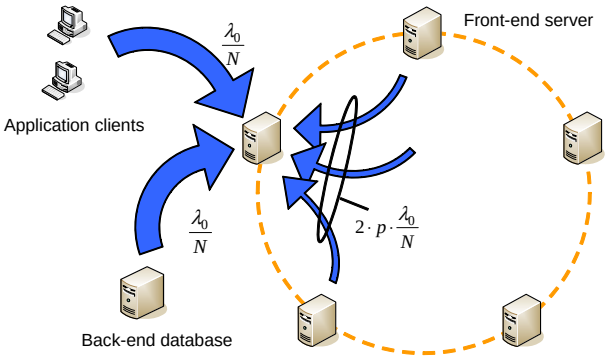


Figure 2.10: *The traffic flows contributing to the node arrival process*

application. Some traffic could be saved if the database virtualization allows for the second node or even a back-end server to answer the application query instead of the first reached node. However, we describe the most general and worst case here with the least assumptions about the interface between external application and virtual database system.

Therefore, all application queries received by the initial node are forwarded to the responsible back-end database server, and the answers are again forwarded to the first node to be forwarded to the application. Thus, the rate of $\frac{\lambda_0}{N}$ is again received from the back end and processed, leading to a total arrival rate of

$$\lambda^* = (2 + 2p)\frac{\lambda_0}{N} = (2 + 2 \cdot \frac{N - R}{N})\frac{\lambda_0}{N}. \tag{2.1}$$

This means that we have to discern two different load values: the load computed from the application queries ρ_0, and the actual node load including internal queries ρ^*. The first value, which is seen from outside the system, may be used as the primary indicator when describing the system as a whole. However, the system performance can only be evaluated with the second value. The normalized

offered initial load of one node is $\rho_0 = \frac{\lambda_0 E[B]}{N}$. With the total arrival rate λ^*, the actual node utilization ρ^* can be derived as

$$\rho^* = \lambda^* \cdot E[B] = \rho_0 \cdot (2 + 2p). \tag{2.2}$$

While this internal load increase seems much, it is still limited due to the single hop that is necessary in the DHT. In a multi-hop overlay, this increase would be larger, roughly by a factor equal to the average number of hops in the DHT.

2.4.2 Analytical Approach

In this section, we present the analytical model used to describe how different parameters affect the system performance. We focus on the search time T_S as the primary performance indicator, i.e., the total time spent by a query in the lookup layer. We provide an approximate analysis of the search time, in order to be able to evaluate the major factors influencing the system.

First, we analyze the distribution function T_S of the total search time, cf. Fig. 2.9. It comprises the node sojourn time in the first node visited by a query in the lookup layer, as well as the transmission times to and from a second node if an internal forwarding is necessary in the lookup layer. In this case, the sojourn time on that second node as well as the second sojourn time on the first node is also part of the total search time. Finally, the response being forwarded from the back-end to the application is processed as well, leading to a fourth node sojourn time. Assuming independent waiting time distributions at all nodes, we arrive at the Laplace-Transform $\Phi_S(s)$ of the search time T_S:

$$\Phi_S(s) = (1-p)(\Phi_W(s)^2 \Phi_B(s)^2) + p(\Phi_W(s)^4 \Phi_B(s)^4 \Phi_T(s)^2). \tag{2.3}$$

Here, $\Phi_B(s)$, $\Phi_W(s)$, and $\Phi_T(s)$ denote the Laplace-Transforms of the distribution functions for the service time B, the waiting time W, and the transmission time T_T, respectively. We model a single lookup node as a $M/GI/1-\infty$ delay

system, using the Pollaczek-Khintchine formula [97]

$$\Phi_W(s) = \frac{s(1-\rho^*)}{s - \lambda^* + \lambda^* \Phi_B(s)} \tag{2.4}$$

and finally obtain

$$\begin{aligned} \Phi_S(s) &= (1-p)\left(\frac{s(1-\rho^*)\Phi_B(s)}{s-\lambda^*+\lambda^*\Phi_B(s)}\right)^2 \\ &\quad + p\left(\frac{s(1-\rho^*)\Phi_B(s)}{s-\lambda^*+\lambda^*\Phi_B(s)}\right)^4 \Phi_T(s)^2. \end{aligned} \tag{2.5}$$

In general, it is numerically difficult to quickly obtain values in the time domain in order to gain basic insights into the system behavior. Thus, we directly compute the mean value and variance of the search time.

From the phase diagram shown in Fig. 2.9, the mean search time $E[T_S]$ can be given as

$$\begin{aligned} E[T_S] &= 2(E[W]+E[B]) + 2p(E[T_T]+E[W]+E[B]) \\ &= (2+2p)(E[W]+E[B]) + 2pE[T_T], \end{aligned} \tag{2.6}$$

with $E[W]$ being the mean waiting time and $E[T]$ being the mean transmission time. Again, under the assumption of a $M/GI/1-\infty$ delay system, we can express the mean waiting time in the queue of a node according to Takács [19] as

$$E[W] = \frac{\lambda^* E[B^2]}{2(1-\rho^*)}, \tag{2.7}$$

which leads us to

$$E[T_S] = (2+2p)\left(\frac{\lambda^* E[B^2]}{2(1-\rho^*)} + E[B]\right) + 2pE[T_T]. \tag{2.8}$$

It should be noted that the second moment of the service time distribution func-

tion is needed in Eqn. 2.8 in order to compute the mean search time. The coefficient of variation of the search time c_{T_S} can also be derived from Fig. 2.9. Since we cannot simply add the weighted variances of the two paths of the phase diagram, we first compute the variances of the search time of the two paths separately. From these values, we can compute the second moments of these two different search time distribution functions. Finally, we compute the second moment of the total search time distribution function from the weighted second moments, and thus the coefficient of variation of the total search process.

To this end, we define the search time distribution functions for the two paths of the phase diagram, namely X for the search time of the upper path without any additional hop, and Y for the search time in the lower path. In order to compute the the second moments of these two distribution functions, we also need the second moment of the waiting time [19]:

$$E[W^2] = 2E[W]^2 + \frac{\lambda^* E[B^3]}{3(1-\rho^*)}. \tag{2.9}$$

Since we assume the independence of all distributions,

$$\begin{aligned} VAR[X] &= 2(VAR[W] + VAR[B]) \\ &= \frac{3\lambda^*(\lambda^* E[B^2]^2 + 4E[B^3](1-\rho^*))}{6(1-\rho^*)^2} \\ &\quad + 2(E[B^2] - E[B]^2) \end{aligned} \tag{2.10}$$

and

$$\begin{aligned} VAR[Y] &= 4(VAR[W] + VAR[B]) + 2 \cdot VAR[T_T] \\ &= \frac{\lambda^*(3\lambda^* E[B^2]^2 + 4E[B^3](1-\rho^*))}{3(1-\rho^*)^2} \\ &\quad + 4(E[B^2] - E[B]^2) + 2(E[T_T^2] + E[T_T]^2). \end{aligned} \tag{2.11}$$

Now, we can compute the second moment of X and Y:

$$\begin{aligned} E[X^2] &= \frac{3\lambda^*(\lambda^* E[B^2]^2 + 4E[B^3](1-\rho^*))}{6(1-\rho^*)^2} \\ &\quad + 2E[B^2] + 2\left(\frac{\lambda^* E[B^2]}{2(1-\rho^*)}\right)^2 + 2\frac{\lambda^* E[B^2]E[B]}{2(1-\rho^*)} \end{aligned} \quad (2.12)$$

and

$$\begin{aligned} E[Y^2] &= \frac{\lambda^*(3\lambda^* E[B^2]^2 + 4E[B^3](1-\rho^*))}{3(1-\rho^*)^2} \\ &\quad + 4(E[B^2] - E[B]^2) + 2(E[T_T^2] + E[T_T]^2) \\ &\quad + (4(\frac{\lambda^* E[B^2]}{2(1-\rho^*)} + E[B]) + 2E[T_T])^2. \end{aligned} \quad (2.13)$$

With Eqns. 2.12 and 2.13 we arrive at the second moment of the search time as

$$E[T_S^2] = (1-p)\cdot E[X^2] + p \cdot E[Y^2],$$

and can finally derive the coefficient of variation of the total search time from the first moments of the service time distribution function and the transmission time distribution function:

$$\begin{aligned} c_{T_S} &= \frac{\sqrt{VAR[T_S]}}{E[T_S]} \\ &= \frac{\sqrt{(1-p)\cdot E[X^2] + p\cdot E[Y^2] - E[T_S]^2}}{E[T_S]}. \end{aligned} \quad (2.14)$$

2.5 Performance Evaluation

Using the analytical model presented in the last section, we now describe the major performance characteristics of the observed one-hop DHT. In particular, we discuss the most important parameters that can be used to influence the system behavior, described by the search time T_S. These are the system size, i.e., the number of lookup nodes N, and the redundancy factor R. Finally, we take a closer look at the system behavior under expected churn conditions.

For the following results, we assume a service process with mean $E[B] = 1\,\mathrm{ms}$ and a coefficient of variation of $c_B = 2$. A node has to manage different tasks, i.e., checking if a query can be resolved locally, forwarding queries to other front-end nodes, creating back-end database queries, and forwarding the results to applications. We reason that these different tasks lead to a high variance in the processing time of such a node. To model the internal network transmission, i.e., query forwarding from front-end server to front-end server, we use an exponential distribution with mean $0.3\,\mathrm{ms}$. We do not consider the querying of the back-end database here, but focus on the time a query spends in the lookup system itself. The values both for the mean service time as well as the mean transmission time are based on measurements conducted with off-the-shelf hardware. It is expected that they are lower in high-performance systems. However, since we provide our results for the mean search time normalized by the mean service time and in dependence on the system utilization, the absolute values are of less importance.

2.5.1 Influence of the System Size

The first parameter of the system under consideration is its size. The more nodes are used to construct the overlay, the less resources are needed on the single nodes. Since typically less powerful equipment costs a fraction of the price of high-end hardware, this may mean a significant cost advantage, as explained in Section 2.3. However, the number of lookup nodes also has a direct influence on

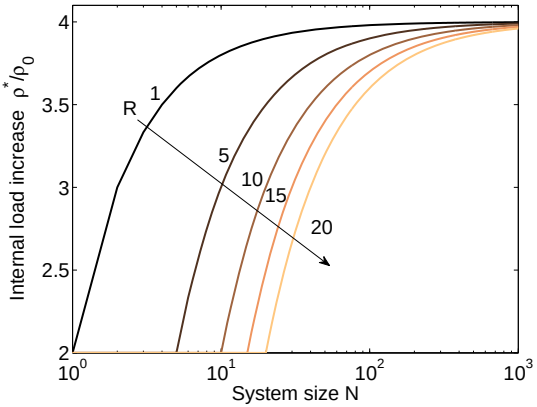


Figure 2.11: *Internal load increase in relation to the system size*

the probability p that a query has to be forwarded within the lookup layer. Since $p = \frac{N-R}{N}$, more lookup nodes lead to a higher probability for two internal hops if R is kept constant. According to Eqn. 2.2, this increases the effective utilization ρ^* of the nodes.

Figure 2.11 shows this internal load increase in dependency of the system size N and for different redundancy factors R. We can observe that the internal load increases quickly once the system gets larger than the redundancy factor. However, after a size of a few hundred nodes is reached, the internal load increase is already close to its maximum and grows only slowly. Therefore, and since realistic systems are of this size [57], we do not consider larger systems. For systems with less nodes than R, no internal load increase exists, since the information is stored fully redundant.

Now, we evaluate how this internal load increase affects the main performance indicator of the system, the search time. To this end, Fig. 2.12 compares the mean search times $E[T_S]$ normalized by $E[B]$ for an initial system utilization $\rho_0 \in]0; 0.25]$, and for numbers of front-end nodes ranging from $N = 5$ to

$N = 100$. Additionally, the hypothetical case where every query is forwarded internally, i.e., $\lim_{N\to\infty} p = 1$ is represented as an upper bound (dashed line). Larger values for ρ_0 are not meaningful, since the effective utilization is up to 4 times larger due to the internal forwarding and therefore $\rho^* \in]0;1]$ for the chosen range of ρ_0, cf. Eqn. 2.2. The redundancy factor is set to $R = 3$, a typical value used in practice [57].

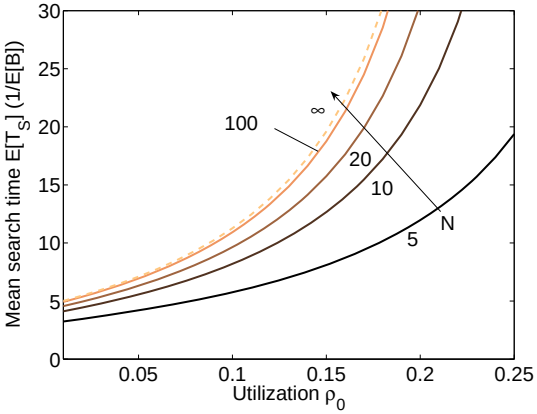


Figure 2.12: *Mean search time for different system sizes*

Our first observation is that the mean sojourn times increase with a larger system. However, the load increase resulting from more front-end servers diminishes for already large systems. This is due to the fact that the forwarding probability p grows fast for smaller systems, as shown in Fig. 2.11. Apart from this, the search times increase with a higher system and therefore node utilization. This is expected, since the waiting time of queries increases when the nodes are under higher load.

Since time-critical applications do not only rely on a short average search time, but also on a low variance of this time to guarantee a good service in most cases,

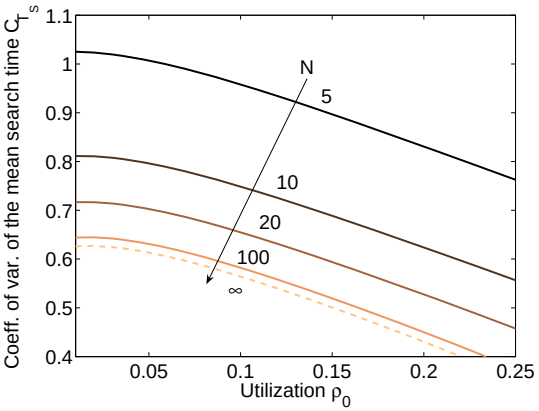


Figure 2.13: *Coefficient of variation of the search time for different system sizes*

we also take a look at the coefficient of variation c_{T_S} of the search time. This value is shown in Fig. 2.13 for the same parameters as the mean search time before.

Here, a larger system has a positive effect, since it reduces the variance of the search time. In a large system, almost all queries take an internal lookup and therefore visit lookup nodes more often. While each processing time and the associated waiting time may show a high variation, experiencing these times repeatedly reduces the overall variance of the total search time. For the same reason, the variance in the search time is reduced with a higher system load. Since more queries experience waiting times if the node utilization grows, the average search time increases, but the difference in the search times experienced by individual queries decreases.

For many applications, not only the variance of the search times is important, but also the percentiles. Services in this case have to guarantee that, e.g., 99.9% of all searches take less than a given time. However, since this value depends on the specific service time distribution, no general results can be provided.

2.5.2 Influence of the Redundancy

The second important parameter that influences the system behavior is the redundancy factor R. It is easier to configure than the system size, since it is a part of the software implementing the overlay functionality. Thus, the redundancy factor can be changed during the system's lifetime and even during normal operations. In contrast, the system size is a result of dimensioning considerations made before the system deployment. Similar to the system size, the redundancy has a direct influence on the probability that a query has to be forwarded internally in the lookup layer, and therefore on the effective load ρ^* a front end server experiences.

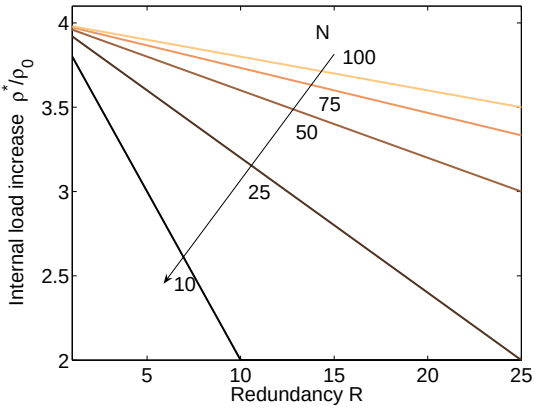


Figure 2.14: *Internal load increase in relation to the redundancy*

This is visible in Fig. 2.14, where, similar to Fig. 2.11, the internal load increase is shown, this time in relation to the redundancy factor R. Different system sizes are compared as well. An increase in redundancy gets less effective for larger systems, where the internal load is reduced less. Furthermore, one should keep in mind that an increase in redundancy necessitates a larger amount of re-

sources per node, so that R is also limited by the node capacities. This is opposed to the aim of larger systems to distribute the load among more nodes, with each of them needing less resources.

Figure 2.15 shows the mean search times of queries normalized by $E[B]$ for an example system with 20 nodes and different redundancy factors, ranging from $R = 1$ (no redundancy) to $R = 20$ (full redundancy, no forwarding is required). Again, only values of ρ_0 that do not lead to system overload are used. We choose this relatively small system size in order to see a significant effect of the forwarding probability for lower values of the redundancy.

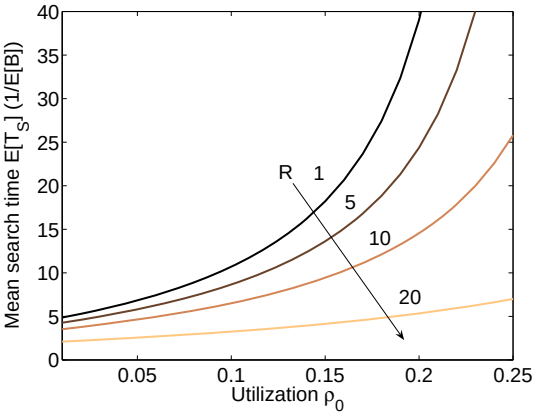


Figure 2.15: *Mean sojourn time for different redundancy factors for a system consisting of 20 nodes*

We can observe the same effect of the forwarding probability on the search times. Systems with a higher redundancy and therefore a lower probability p show lower mean search times. On the other hand, a higher redundancy also means that more data has to be stored on each node. In any case, the redundancy factor can be used as a parameter to tune the system to the needs of the operator. It can balance the resource efficiency of the lookup architecture with its perfor-

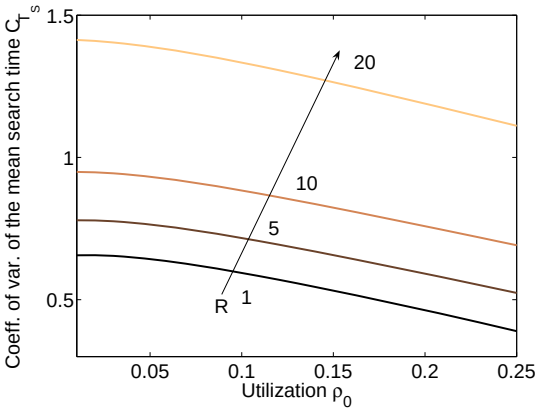


Figure 2.16: *Coefficient of variation of the search time for different redundancy factors*

mance, while being less intrusive than a change in the number of nodes.

We again consider the coefficient of variation of the search time in addition to its mean value, cf. Fig. 2.16. For the same reasons as explained for the system size analysis, the variance in the search times decreases with a higher load for all values of R. Similarly, due to the higher number of node traversals of the queries, systems with less redundancy show a lower coefficient of variation for the total search times.

2.5.3 Reorganization Effort in Case of Failures

One of the major reasons to employ an overlay for a lookup system is its ability to react to changes in the system in a self-organizing fashion. This is mainly needed in case of churn. The cost for this feature is the overhead needed to detect changes in the system topology, and the mechanisms adapting the overlay to the new situation. While in the scenario considered here, the rates at which nodes

join or leave the system are negligible in comparison to a DHT utilizing end user hosts in the Internet, there is no guarantee that nodes will never fail or that the system will never be expanded. Therefore, we want to give an impression of the effort needed to cope with such a situation without any manual intervention.

Since load distribution is a critical characteristic for the efficiency of the described architecture, it has to be restored as soon as possible after one or several node failures. The same applies to inserting additional nodes into the system. However, we assume that an expansion is executed in a more planned and controlled manner. Apart from this, the reduction of servers also increases the load on the remaining servers, even if an equal load distribution can be achieved. Therefore, it is the worst case for the reorganization algorithm.

In order to conduct a first evaluation of the rather straightforward method described in Section 2.3, we conduct a Monte-Carlo simulation for different node failure scenarios. We vary the number of nodes that fail concurrently in a system consisting of 40 nodes. For a given number f of node failures, we select a random subset S of the nodes with $|S| = f$. This experiment is repeated 10,000 times for one value of f, and confidence intervals are given for a confidence level of 99%. We record the amount of data that has to be moved during the reorganization phase in order to achieve equal load distribution again, relative to the total amount of stored data. We assume here that the data is placed with roughly equal density on the identifier ring. Furthermore, we neglect cases where enough successors of a failed node also fail, which results in the loss of data. Since in this case less data has to be transmitted over the network, the presented results are an upper bound, even if data loss is an undesirable event.

Figure 2.17 shows these results for three different replication factors, $R = 2$, $R = 3$ and $R = 4$. The amount of data that has to be transmitted increases for higher replication grades and a larger number of node failures. The maximum amount of moved data in the worst case equals $R \cdot 100\%$, meaning that more data has to be moved than there is in the ring. This initially counter-intuitive characteristic stems from the fact that the responsibility areas of the reorganizing nodes may overlap, meaning that several nodes have to retrieve the same data sets

if they have not stored them before the reorganization.

Due to fact that each node is responsible for a single, continuous range of the identifier space, the number of data entries that have to be moved to a new node is not equally distributed among all nodes. Especially the successors of a failed node normally have to be moved across a larger distance in the identifier space than the following nodes. It is expected that different schemes, such as the one proposed in [57], might be able to reduce this unfairness, lowering the maximum amount of data that has to be moved to a single node, and consequently the time it takes to reorganize. Still, the absolute amount of moved data remains the same, since it only depends on the system size and the redundancy.

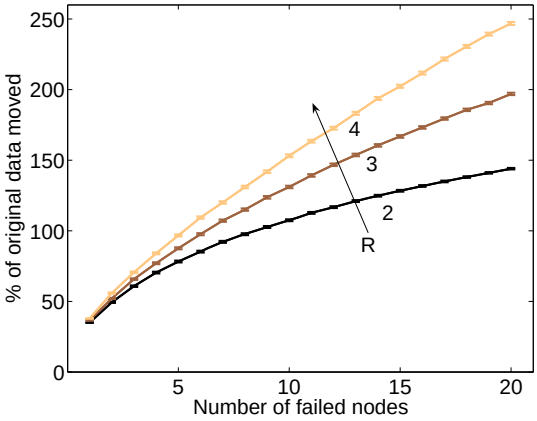


Figure 2.17: *Relative amount of data that has to be moved during reorganization in a system with 40 nodes*

2.6 Lessons Learned

In this chapter, we saw that DHTs, although offering a scalable and reliable search functionality, still have to be optimized in order to be viable solutions for high-

performance applications. The demand for a fast search leads to one-hop DHTs, which do not require a lengthy routing process. Still, these systems come with performance challenges of their own. While the internal load is increased in every overlay which involves the forwarding of queries, this effect has a significant impact on architectures where resources on the participating nodes are scarce.

There is an implicit trade-off between a fast search, where a fully redundant storage of the data would be the best case, and a smaller amount of storage resources per node, which necessitates a time-consuming internal routing process. This is complicated by the fact that lookup data distribution is not efficient for all resources, since internal traffic means more consumed bandwidth and a higher query load that has to be processed on the servers.

In our analysis, we identified this general trade-off between the node load and search speed on the one hand, and the necessary storage resources and overhead for load distribution in case of failures on the other hand. Although fast lookup is the main performance optimization goal, the amount of resources per node and the number of nodes are essential for a cost analysis and for system dimensioning. Additionally, these values govern in which load area the system can be used.

This trade-off can mainly be influenced by two parameters, namely the system size and the redundancy with which data is stored in the overlay. Larger systems in general lead to a higher node load and longer searches, even in case of a one-hop DHT. However, a system with more nodes needs less storage resources per node. A higher redundancy factor, which is easier to configure and change, has the opposite effect: it lowers the search time and the number of queries a node has to process in a given time, but each node has to store more data.

In small-scale systems, an additional effect can be observed. Load distribution is no longer automatically given in these overlays if nodes are placed randomly in the identifier space. Thus, a larger effort has to be made to ensure that the query load is evenly distributed. This is especially true when node failures occur and the content in the overlay has to be re-distributed. Here, a higher redundancy leads to more generated traffic and therefore longer reorganization times. The results concerning the internal load increase of one-hop DHTs can be extrapolated for

multi-hop DHTs. It can be generalized that more hops also lead to an even higher increase in the internal load apart from the longer search time. This only underlines that one-hop DHTs are better suited for high-performance applications than multi-hop DHTs.

3 File-Sharing Overlays

Opportunities multiply as they are seized.
Sun Tzu (544 BC - 496 BC)

Emerging in the late 1990's as an application, file-sharing is at the time of this publication the largest single bandwidth contributor in the Internet according to recent studies [75, 77]. File-sharing applications can offer good performance in terms of download speed for end users, at least for popular files [42]. Its popularity can be explained by the fact that file-sharing networks provide content for free, even if the legality of copying and distributing the often copyright-protected content is questionable.

The aim of file-sharing is to distribute data to all users in a fast and self-organizing fashion, using resources like disk storage and upload capacity from the clients themselves. It is also common for a file-sharing application to include an index service that allows end users to search for specific files and sources for them [35]. This service can take any of the forms discussed in the last chapter, including an overlay. However, in this chapter we focus on the actual content distribution of a file-sharing application.

The most widely used file-sharing applications and protocols, such as BitTorrent [66] or eDonkey [35], use overlays to distribute data. In contrast to the search overlays of the previous chapter, these overlays typically do not form a pre-defined structure, but create a random, mesh-like logical network. Neighboring peers in this network exchange content data and do not just route search queries. However, signaling traffic is still necessary to manage the data exchange.

The simplest forms of these overlays are loosely connected structures. Here, a peer only has to find another peer storing the complete requested file. The querying peer then downloads the complete file like it would from a server. Thus, each peer only has a few connections which are used to exchange data [30]. This is different in newer and more efficient overlays, where parts of a file may be downloaded from a single peer [35, 66]. This enables a download of the same file from many different sources at the same time, referred to as multi-source download. Consequently, the number of overlay connections used for data exchange is larger in these systems.

For the same reason, the complexity of the overlays is higher as well. Mechanisms to achieve an efficient utilization of resources and a fair load distribution are part of currently popular overlays [35, 66]. Moreover, the systems have to take the effects of churn into account, although these are less critical than for the DHTs of the last chapter. Since no fixed structure has to be maintained for the overlay for it to function properly, churn is less dangerous to the functionality of the overlay here.

File-sharing overlays utilize the upload bandwidth and storage capacity of all participating users. Each peer is supposed to upload data it has already received and which is needed by other peers. Since this technique can be used to relieve load on servers, P2P file-sharing is interesting for traditional content providers to provide software patches or whole software distributions [86].

The popularity of file-sharing applications and the consequently enormous amount of traffic and costs generated by them leads to challenges for the Internet Service Providers (ISPs) of the end users participating in file-sharing networks. We discuss these challenges in the Section 3.1. Then, we introduce the most popular file-sharing protocol, BitTorrent, which is used in our performance evaluation, and present results from literature that are related to our work in Section 3.2.

Our first contribution is a new approach to locality-awareness, the most extensively studied solution to the problem considered here. The specific implementation of our approach as well as that of the dominant mechanism from lit-

erature are followed by our model for a simulative performance evaluation in Section 3.3. This model specifically includes swarm topologies based on measurements of live BitTorrent swarms, in order to test whether ISPs and end users do profit from locality-awareness. Finally, we provide results from this evaluation in Section 3.5, comparing the different approaches and considering both relevant perspectives, i.e., that of the ISPs and that of the end users. We characterize the basic features of the two approaches and evaluate the effects they have on realistic swarms as currently found in the Internet. Apart from this, we show how the relevant parameters of the two mechanisms influence their behavior. The chapter ends by summarizing the important conclusions from our experiments.

3.1 Challenges in File-Sharing Overlays

Ideally, every user in a file-sharing overlay uploads data to other peers. Since file-sharing overlays grow to sizes of several 10,000 peers, and since there are an even larger number of different overlays [78], the generated amount of data is enormous. P2P traffic is estimated to make up 50% of the total consumer Internet traffic in 2009, amounting to about four exabyte [75]. Resulting from this, one of the currently most pressing challenges is the difficult management of the generated data flows by network providers.

One of the reasons for this is that the source and destination of a single data transmission are determined only by the overlay. Neighbors in a file-sharing overlay are generally not chosen according to their position in the underlay [35, 66]. This underlay-agnostic structure leads to overlay connections spanning several Autonomous Systems (ASes) and in the worst case several transcontinental links. One ISP may administrate more than one AS, depending on its importance in the Internet hierarchy, i.e., the Tier it belongs to. A Tier1 provider with several ASes typically has peering agreements with other Tier1 providers. Tier2 providers may only administrate a small number of ASes or even only a single AS and forward traffic to the rest of the Internet via their Tier1 provider at a cost. Similarly, a Tier3 provider uses and pays for the services of a Tier2 provider. Thus, over-

lay connections do not only span ASes, but also provider networks, incurring costs [40].

The longer such an overlay connection is, the more resources it consumes and the more costs it creates for the underlay network providers. Especially traffic flowing from Tier3 or Tier2 providers to higher tiers is costly for these ISPs, as is traffic on intercontinental connections that has to be transported via oceanic cables. These costs caused by an underlay-agnostic overlay structure constitute a large saving potential for providers. Thus, it is currently a major interest of ISPs to reduce these costs by managing P2P traffic more efficiently.

Unilateral approaches from the ISPs to tackle this problem have been tried in the past. One simple solution is to throttle the bandwidth of P2P connections, so that less traffic is generated by them. In reality, this mechanism has proven to be problematic [59]. End users view this bandwidth reduction as a reduction in service quality, if not as an intrusion in their way of using the Internet. Thus, user satisfaction is severely reduced.

In addition, overlay providers and end users react to bandwidth throttling or other harmful interventions of the ISPs by making P2P application traffic harder to identify and to manage. The flexible utilization of ports as well as the encryption of P2P traffic [91] are steps in an arms race against ISPs which only leads to even higher costs on the service provider side.

Thus, a new way of approaching the problem of managing P2P traffic is needed. The experiences described above lead to the conclusion that any successful management scheme has to include the overlay providers and the end users. This approach is recently followed, e.g., by Economic Traffic Management (ETM) [1] or the efforts of the Application Layer Traffic Optimization (ALTO) IETF working group. Here, all participating partners have to benefit or at least not to be penalized by the traffic management, so that they can partake voluntarily. More formally, it is required that the download performance of the end users should improve or at least must not be decreased, while ISP costs should be lowered. This latter task is achieved by reducing the costly inter-AS traffic. Next, we present some mechanisms and architectures that implement this basic idea.

3.2 Background and Related Work

In this section, we describe all relevant mechanisms of BitTorrent, the currently most popular file-sharing protocol. BitTorrent is in widespread use and creates a significant share of today's Internet traffic [77]. We describe the key mechanisms of standard BitTorrent. A detailed description of BitTorrent including all mechanisms and values mentioned below can be found in [50] and [66]. A large number of BitTorrent application clients exist, introducing several modifications of this protocol. However, we focus on the common and standard features here, which define the behavior of the predominant BitTorrent clients and are therefore the most relevant. This protocol is used in our performance evaluation of locality-aware mechanisms, which are discussed in the second part of this section.

3.2.1 The BitTorrent Protocol

The BitTorrent protocol forms a mesh-based overlay and utilizes multi-source download to distribute content. For each shared file, one overlay is formed, a so-called swarm. To facilitate the multi-source download, a shared file is split into smaller pieces called chunks. These chunks are in turn again separated into sub-pieces or blocks. The size of these chunks and blocks can be set by the initial source, but should depend on the size of the file to be shared. Typical values are a size of 256 or 512 KB for chunks and 16 KB for blocks.

Once a peer has downloaded a complete chunk, it can share this chunk with its neighbors in the overlay. Thus, only a part of the file has to be downloaded by a peer in order to utilize its upload capacity. It is also easier to download different parts of the same file from different peers. As a result, the file spreads much faster than without the file partitioning.

Neighbor Set Management

Each peer has only a limited number of other peers in the swarm to which it has direct contact. A peer joining a swarm typically initializes its neighbor set by

contacting a tracker, i.e., an index server with global information about the peer population of a swarm, information containing the addresses of all peers, and typically statistical information about the overall download progress. A standard tracker responds to queries with a random subset of all peers. Peers obtain the address of the tracker for a swarm by downloading a .torrent file from a website. Once a peer A has received a list of contacts in the swarm, it tries to establish connections to them. If it is successful, the according remote peer B is added to A's neighbor set and vice versa.

A new peer tries to establish a minimum number of neighbor connections, typically 40. Once it has reached this number, it does no longer actively seek new neighbors. However, it will accept incoming neighbor connections until it has reached a maximum number of neighbors, by default 80. Should a peer lose too many neighbors by churn so that it has less than 40 neighbors, it will contact the tracker and request new connections until the minimum number of neighbors is reached again.

Each peer sends a message to new neighbors containing its currently available chunks and updates its neighbors whenever it has downloaded a new chunk. Thus, neighbors know about each other's download progress, i.e., which chunks the other has already downloaded. This enables a peer A to signal its interest in downloading chunks to a neighbor B holding chunks that peer A is still missing. We say that peer A is interested in peer B. This allows peer B to consider only interested peers, i.e., peers that can actually download chunks from B, in its decision which peers should be allowed to download.

Choke Algorithm

A peer uploads data only to a limited number of its interested neighbors in order to avoid splitting its upload rate between too many peers. A peer that may request blocks from the local peer is called unchoked. All other peers are therefore choked, which is the default state. In standard BitTorrent, each peer has four unchoke slots that it assigns to interested neighbors, three regular unchoke slots

and one optimistic unchoke slot.

Every 10 seconds, a peer decides which of its interested neighbors it will regularly unchoke for the next 10 seconds. The regular unchoke slots are awarded to the peers that offer the currently highest upload rate to the local peer. This strategy is called tit-for-tat and provides an incentive for peers to contribute upload bandwidth to the swarm. If the local peer has already downloaded the complete file, i.e., it is a seeder, the slots are given to all interested neighbors in a round-robin fashion.

Additionally, every 30 seconds a random peer that is currently choked is selected for optimistic unchoking for the next 30 seconds. This allows a peer to discover new mutually beneficial data exchange connections. It is assumed that an optimistically unchoked peer reciprocates via the normal tit-for-tat mechanism.

Chunk Selection Algorithm

Once a peer A has been unchoked by a neighbor B, it has to choose which chunk to download from B. From all the chunks that B can offer and that are not completely downloaded by A, A chooses the chunk that is seen the least in its neighbor set. This rarest first or least shared first strategy tries to prevent single chunks from being shared much less than others. In the worst case, this can lead to the vanishing of such chunks from the swarm, since they are stored on only a few peers that go offline at some point. This so-called chunk starvation can prevent peers from downloading the complete file. Even if chunks exist in the swarm but are very rare, this increases the download times of peers. The upload capacity of the few sources of a rare chunk becomes the bottleneck in this case.

Once a chunk has been selected, A requests missing blocks from this chunk from B. In order to lessen the impact of the RTT between A and B and to take advantage of pipelining, several blocks are requested in parallel, with B serving the requests one after another. A can request new blocks from B as long as it remains unchoked at peer B.

In normal operation, each block is only requested from one peer at a time. This

is changed in endgame mode, when a peer only misses a few blocks to finish the download. In order to speed up this last phase, a peer requests the missing blocks at all peers where it can get these in parallel. After a successful download, the now superfluous requests are canceled again.

3.2.2 Approaches to Locality-Awareness

The random structure of a BitTorrent overlay, and of file-sharing overlays in general, leads to many overlay connections spanning several AS and ISP boundaries. The term locality-awareness denotes mechanisms that try to utilize some knowledge of the underlying network, so that the overlay can adapt to its underlay.

Locality-aware mechanisms need a metric defining which peers are 'close' in the underlay, and which are 'remote'. Since a common aim is to reduce cross-ISP traffic, a typical metric is that two peers are close if they are in the same AS [15, 46, 70]. The related number of AS hops between two peers can also serve as a metric. The RTT is another option, but is less stable and depends not only on the topology, but also on the traffic load conditions in the network.

The impact P2P overlays have on ISPs networks, and the potential gains that locality-awareness can provide, are analyzed in [40]. It distinguishes the roles of users, content providers and ISPs. For each of these groups, the impact of P2P usage in comparison to Content Distribution Networks (CDNs) or servers is characterized. While the negative effects of P2P on ISPs are acknowledged, locality-awareness is presented as a simple mechanism that can be used to diminish these effects. It is also compared to a caching mechanism implemented by the ISPs in terms of efficiency.

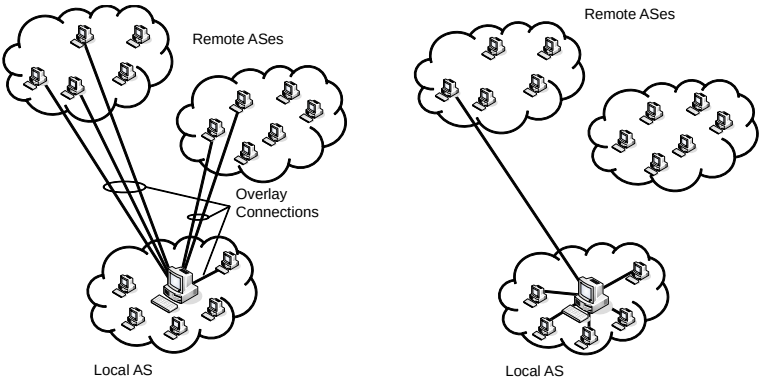
In the evaluation in [40], BitTorrent traces are used to calculate the traffic savings that can be achieved by locality-awareness. Flow traces from an access link of a university are analyzed to show that inter-AS bandwidth can be saved by preferring local sources for data, and that download times can be improved. Similarly, a tracker log is evaluated to compare theoretical systems, namely a central server, a standard P2P system, a perfect caching strategy, and a locality-

aware P2P scheme. While the idealized caching saves much more traffic than locality-aware P2P, the savings of the locality-aware solution in comparison to the standard implementation are significant as well. However, for both evaluations, a highly theoretical and idealized version of BitTorrent is considered, so that the results cannot be mapped to a realistic swarm.

One of the most influential works on locality-awareness is [46]. Here, it is proposed to use Biased Neighbor Selection (BNS) for BitTorrent-like P2P systems. With BNS, the neighbor set of a peer is modified to contain preferentially peers in the same AS, cf. Fig. 3.1. One of two basic variants to implement this is presented. Here, the tracker is modified to be aware of the ASes the peers are located in. It can then return a list of contacts which is tailored to the requesting peer, instead of a random subset of all peers. The second alternative is peer-based BNS, where the peers themselves decide which neighbors to choose. To this end, a peer requests more contacts than in the standard implementation, either by specifying a larger value in its request or by querying the tracker repeatedly. Then, it rates the contacts by using some service offering locality information such as described in [54], [74], or [68], and chooses accordingly. The latter alternative offers a higher degree of freedom for the users, since they can decide whether to support locality-awareness or not.

The evaluation of BNS in [46] uses simulations with a homogeneous peer distribution of 700 peers over 14 ASes. The results show that a large fraction of the inter-AS traffic can be saved by BNS and the median as well as the 95th percentile of the download times over all peers are decreased. Different results are reported in bandwidth throttling scenarios, where bottlenecks are introduced in the inter-AS links by the ISPs. Here, the download times increase significantly, while the traffic is reduced less than in the BNS case. Finally, the combination of bandwidth throttling and BNS shows that the peers utilizing locality can successfully avoid the bottlenecks, since the download times increase only by up to 10% in the considered scenarios. The traffic savings are increased even further in comparison to the BNS case without bottlenecks.

In [70], an approach very similar to BNS is investigated by experiments of up



(a) Random neighbor set (b) Neighbor set with BNS

Figure 3.1: *Influence of BNS on a peer's neighbor set composition*

to 10,000 real BitTorrent clients which are homogeneously distributed among a varying number of ASes. The aim of the study is to find out how far locality-awareness can be taken without degrading the robustness and performance of the swarm. According to the results, BitTorrent locality can be driven to extremes, i.e., the neighbor set of all peers contains almost exclusively local peers, without degrading the performance from the viewpoint of a P2P user. An improvement in the download times of peers can be achieved when bottlenecks in inter-AS links are present. However, in the opposite case the download times even increase in the worst case in the evaluation. The performance of the considered setup is worse when churn is included, since then a larger number of peers do not finish their download.

Another approach based on BNS is proposed in [54]. In this architecture, peers query an oracle server which is maintained by the ISP of the the respective peers. This server holds locality information and policies of the local ISP. It ranks the

peers according to the preferences of the ISP and sends this information back to the peers. The peers then base their selection of neighbors in the overlay on this ranking. Consequently, they can include traffic engineering policies in their peer selection. The evaluation in [54] is based on the Gnutella protocol. Properties of the overlay graph with and without usage of the oracle server are compared. It is shown that the overlay using the oracle server has much less connections spanning several ISP networks, while still being well-connected. Accordingly, searches can be resolved in the AS where they originate with a higher probability. As a result, content traffic is also concluded to be more localized.

The P4P project [74] goes further than earlier approaches and also considers the intra-AS topologies in addition to the AS topology itself. To this end, the concept of opaque IDs (PIDs) is used, which can stand for a set of clients connected via the same Point of Presence (PoP) of a provider, or a set of clients with the same network status. The authors propose to create an iTracker similar to the oracle of [54] which communicates to the P2P application and gives recommendations about which peers to contact. Different optimization functions are presented that can be used by the iTracker to make its recommendations.

The system used for evaluation implements the tracker-based variant of BNS, since the iTracker communicates with the tracker of the overlay application. The application tracker then returns a list of contacts to querying peers that contains, in order of preference, peers in the same PID, peers in the same AS and finally peers from the rest of the swarm. The evaluations include simulations as well as measurements in PlanetLab and in the network of Pando, a P2P vendor. One of the measured metrics is the download time for a complete swarm, i.e., the time until all of a fixed number of peers have downloaded the file. Another is the traffic on the bottleneck link of the network. It is shown that the P4P system improves the download times of the swarms between 0 and 20%, while the traffic is reduced significantly. The evaluated scenarios are not described in detail with respect to their peer distribution in the topology.

Finally, a plugin called Ono for the open-source BitTorrent client Vuze is presented and evaluated in [65]. The main difference of Ono is that it does not rely

on a central entity which guides the inclusion of peers in the neighbor set of a peer. Instead, it uses the similarity of the redirection ratio of CDN servers as a metric how close peers are. To this end, a peer sends DNS queries for a defined set of CDN servers, e.g., from Akamai. A large number of these servers exist at selected points in the network, so that DNS can resolve them in a way that IP addresses topologically close to the querying peer are returned. Each peer can then build a vector with the probabilities with which it is resolved to which IP address of a CDN server. These vectors serve to determine the distance of two peers. The Ono plugin tries to keep peers in the neighbor set that are close in the network by re-inserting them whenever they are removed from the set. The results presented show that peers that are recommended by Ono have a shorter AS distance to the local peer than random neighbors. Thus, traffic exchanged with these peers is also less costly to the network. However, it is unclear how significant these results are, since the share of recommended neighbors to normal ones in the observed clients is not given.

In [82], the authors present three pitfalls for ISP-friendly P2P design: limited impact, reduced performance and robustness, and conflicting interests. They show that locality-aware peer selection has no impact when there are only very few peers of a swarm in the same AS. They prove this by theoretically analyzing the performance potential of swarms based on collected tracker data, as well as conducting measurements with a single peer with adapted behavior in live swarms. One of these modifications is using the Ono plugin, which is shown to have a negligible effect. However, only four measurement experiments were conducted, which allow only for limited conclusions. Another result for a theoretical swarm shows that users may get less bandwidth if locality-awareness is applied. All in all, several issues with the application of locality-aware mechanisms are highlighted, but no performance evaluation of a detailed swarm model including these mechanisms is conducted.

In this work, we consider an additional and new locality-aware mechanism, Biased Unchoking (BU), that is compared to and combined with BNS. We evaluate these mechanisms under more diverse conditions than in the related work.

Apart from this, we study scenarios with swarm sizes and peer distributions observed in real BitTorrent swarms [78, 83], i.e, with heterogeneous peer distributions and heterogeneous access bandwidths of the peers. We study their impact on the performance of a BitTorrent network for the ISPs and for the P2P user and explain who can benefit from locality-awareness and who cannot. Our performance evaluation includes a detailed model of client mechanisms, in contrast to many high-level evaluations.

3.3 Locality-Awareness in File-Sharing Overlays

In this section, we describe the specific locality-aware adaptations evaluated in our experiments. We choose the BitTorrent protocol as the file-sharing architecture in which we implement these adaptations, since BitTorrent is currently the most commonly used application for file-sharing and contributes a large share of today's Internet traffic, as shown in Section 3.2.

In order to evaluate the effect of locality-awareness on the ISPs and on the end user, we consider and compare two main client adaptations that utilize locality information. From the related work, the best known approach is BNS. We shortly describe the specific implementation of BNS used in our experiments, as well as our own locality-promoting client mechanism, BU.

Both mechanisms need a locality metric to decide which peers are considered closer than others. The predominant solution in literature, e.g., used in [15,46,70], is to differentiate between peers in the the same AS (local peers) and peers in other ASes (remote peers). Therefore, we keep this simple differentiation and assume that all peers have access to the information which other peers are local or remote to them. This could be implemented in practice for example by an information service provided by the ISP [68] or by contacting public databases.

3.3.1 Biased Neighbor Selection

Since the willing cooperation of the user in any locality promotion approach is crucial for its success, we consider the peer-based BNS described in Section 3.2. Peer-based BNS leaves it to the user client to gather locality information about potential neighbors and to decide which contacts should be added to the neighbor set. Thus, the user is not forced to promote locality.

The specific implementation used in our experiments queries the tracker for a much larger number of contacts (1000) than in the standard approach (50) [50]. This number should be larger than typical swarm sizes [78], and therefore allow a peer to make the best selection among all peers in the swarm. The peer then tries to keep a ratio of l_{BNS} local neighbors in its neighbor set by an intelligent selection from this larger set of potential neighbors. Figure 3.2 illustrates this for $l_{BNS} = 0.8$ and a number of neighbors $N = 5$.

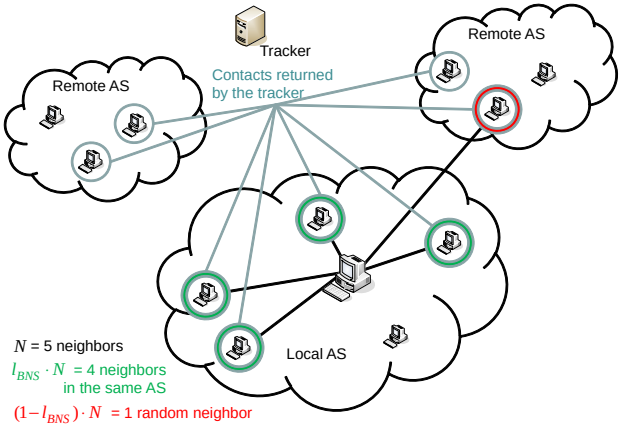


Figure 3.2: *Peer-based Biased Neighbor Selection*

To this end, connections to peers in the same AS are established until it reaches the required number of local neighbors or no more local contacts are known. In both cases, the missing number of neighbors is taken from remote peers until the BitTorrent standard minimum value of 40 neighbors is reached. Thus, the composition of the initial neighbor set of a peer can be influenced, as well as the addition of new neighbors whenever there are less neighbors than the default number. However, we do not filter which incoming connection requests are accepted, so that the actual composition of a peer's neighbor set may differ from the ideal values even if enough local peers exist.

If not mentioned differently, we set $l_{BNS} = 0.9$. This is a conservative choice compared to [70], where values up to 0.999 are investigated, and [46] where 34 out of 35 neighbors are local if possible. However, this value already serves to show the effects of locality-awareness on both P2P traffic and download times experienced by the users.

3.3.2 Biased Unchoking

The BU mechanism evaluated here is specifically tailored to BitTorrent-like P2P networks. In contrast to BNS, which affects the establishment of overlay connections, BU changes the unchoking mechanism. While the composition of the neighbor set influences the data traffic in the overlay only indirectly, a change in the unchoking procedure has a direct effect.

With BU, local neighbors are preferred in the unchoking process, i.e., chunks are preferentially uploaded to local peers. To this end, the optimistic unchoke slot is assigned to a local neighbor with probability l_{BU} if a local neighbor is present, cf. Fig. 3.3. Otherwise, a remote neighbor is chosen. Via the tit-for-tat policy of BitTorrent, this small modification has a strong impact on all four unchoke slots, since local neighbors have a higher probability to prove their worth to be unchoked regularly. Still, we only affect the optimistic unchoking slot directly, thus avoiding to interfere with the tit-for-tat policy governing the remaining slots.

If not mentioned differently, we set $l_{BU} = 0.9$. Again, that is a more con-

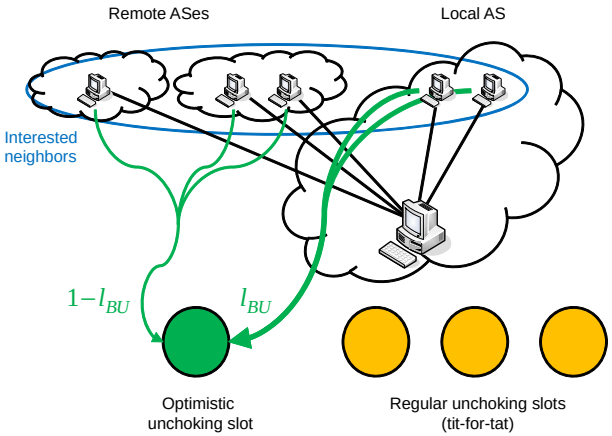


Figure 3.3: *Biased Unchoking*

servative choice than in [15], but sufficient to show the impact on the traffic and the download times. In addition, it leads to similar preferences for local peers as $l_{BNS} = 0.9$ in Section 3.3.1. Still, we also conduct a parameter study on both l_{BU} and l_{BNS} in Section 3.5.3.

3.4 Simulation Model

The performance evaluation of the different scenarios was conducted by means of a discrete event simulation. We first present the default simulation scenario for our experiments. In contrast to the related work, it is based on measurements of live swarms. Then, we describe the used simulator, which also captures all relevant overlay details in order to improve the quality of the conclusions. This level of detail is necessary to capture the specific effects that have a large impact

on the results, such as the number of local neighbors in many ASes.

3.4.1 Default Swarm and Topology Model

We simulate one BitTorrent swarm which exchanges a file of size 154.6 MB generated from an example TV show of about 21 minutes length in medium quality. This is a typical content being shared regularly and extensively in the Internet, since TV shows are one of the content categories with the largest number of users [78]. The file is divided into chunks of 512 KB and every chunk into blocks of 16 KB, which are standard values for a file of this size [90].

We simulate the swarm for five hours in the steady state, evaluating the longest state of a popular swarm after its initial flash-crowd phase. New peers join the swarm with an exponentially distributed inter-arrival time A with a mean value of $E[A] = 10\,\mathrm{s}$. The peers stay online for the full download duration of the file plus an additional, exponentially distributed seeding time with a mean value of 10 minutes. The average online times of the peers are in the range of half an hour, so that we can assume that peers do not go offline during that duration. As a result, we measured that the swarm contains on average about 120 to 200 peers depending on the specific parameters of the scenario. Thus, one simulation run consists of about 2300 downloads in the default scenario. This is a typical swarm size for live swarms, as shown in [78].

The chosen values for the inter-arrival time and for the online time ensure that the upload capacity of the system is limiting the download performance, which is typically the case for live swarms due to the widespread asymmetric DSL access or end users. On the other hand, the demand is not large enough to lead to unrealistic loads on the swarm in its steady state [78].

We simulate a multi-AS underlay network in order to evaluate the inter-AS and intra-AS traffic generated by the overlay. This network forms a star topology and consists of one transit-AS and $n = 20$ stub-ASes connected via inter-AS links, i.e., the stub-ASes are all connected to the transit-AS but not directly interconnected with each other, cf. Fig. 3.4. This number of stub-ASes allows for a

detailed view on single ASes, especially for scenarios where the AS characteristics differ.

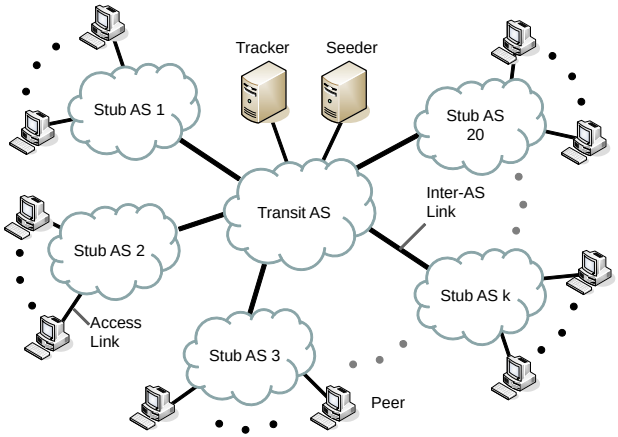


Figure 3.4: *Simulated network topology*

The stub-ASes model the Tier2 or even Tier3 ISPs connecting end users to the Internet. The transit-AS models the Tier1 core of the Internet. Since typically peering relationships exist between these Tier1 ASes, and since the links connecting them form no bottleneck, the traffic flowing between them is of no interest to our evaluation. Therefore, we reduce the topology complexity by modeling the core network as just one AS. From a client mechanism perspective, it is only necessary to differentiate between peers in the same AS and remote peers, so that this abstraction has no effect on the function of the evaluated mechanisms.

For the homogeneous peer distribution scenarios, the peer arrival process is equally distributed over all stub-ASes, i.e., when a new peer arrives, it randomly joins one of the stub-ASes. The transit-AS does not contain any regular peer. In case of swarms where the number of peers is heterogeneously distributed in

the topology, the arrival process is split among the ASes proportionally to the targeted amount of peers per AS.

If not stated otherwise, the peers are connected to their stub-AS with an access speed of 16 Mbit/s downstream and 1 Mbit/s upstream, which are typical values for ADSL access. The tracker and the initial seeder are placed in the transit-AS for symmetry reasons. The seeder has a symmetric upload and download bandwidth of 10 Mbit/s, respectively. It goes offline before the system reaches it steady state to minimize its effect. We model the inter-AS links as well dimensioned in the default scenario.

3.4.2 Flow-Based Underlay Model

For our simulation studies, we use the P2P simulation and prototyping Java framework ProtoPeer [76]. ProtoPeer contains a network model for bandwidth-dependent overlay applications like BitTorrent. Furthermore, it facilitates the development of overlay applications as only the specific peer behavior needs to be implemented within the framework.

For the underlay network, we use the flow-based network model provided by ProtoPeer. This network model mimics the property of TCP that the capacity of a link is shared among all data connections between two peers using this link. To simulate this bandwidth allocation, the bandwidth of the connections are assigned according to the max-min-fair-share principle [95]. The time a connection needs to transmit its data depends on the available bandwidth. When all data of a connection is transmitted, the connection is removed from the network. The use of such a flow-based network model for P2P simulations is proposed in [28] and [51]. These studies describe concrete implementations of the bandwidth allocation algorithms and evaluate their runtime speed. A comparison of the resulting transmission times to a packet-based NS-2 simulation is also given in [28] and shows that for our scenarios, a flow-based approach does not sacrifice exactness but is still efficient.

Since the bandwidth allocation process is a costly operation in terms of com-

putation time, we only allocate bandwidth to connections which simulate the transmission of a block of the shared file from one peer to another. These are called piece messages in BitTorrent and have a size of 16 KB. All other messages in the BitTorrent protocol are orders of magnitude smaller than piece messages and are therefore assumed to have a negligible impact on the bandwidth dynamics of the network.

While the network model was provided by ProtoPeer, the framework does not contain an implementation of the BitTorrent protocol. Therefore, we use a self-written implementation of BitTorrent according to the descriptions in [50] and [66]. It includes all key mechanisms, in particular the piece selection mechanisms, the management of the neighbor set, and the choke algorithm. Furthermore, the complete message exchange among the peers themselves, between peers and the tracker as well as between the peers and the information service for locality data, is simulated in detail.

3.5 Performance Evaluation

In this section, we present the results from our performance evaluation, using the model and simulator described in the last section. In the experiments, we compare four different peer behaviors: regular BitTorrent (abbreviated as 'Ref' for the remainder of this chapter), BitTorrent with Biased Unchoking (BU), BitTorrent with Biased Neighbor Selection (BNS), and BitTorrent with both BNS and BU (BNSBU). We evaluate the impact of the described locality-awareness mechanisms both on the ISPs and on the end users.

With BU, a peer selects a local interested neighbor to be optimistically unchoked with $l_{BU} = 0.9$, cf. Section 3.3.2. For BNS, we set the fraction of local peers that a peer tries to include in his neighbor set to $l_{BNS} = 0.9$. However, since this is only a target value and since the number of peers per AS is low in most of our scenarios, this value normally cannot be reached. Thus, the difference is made up from remote peers, as described in Section 3.3.1.

We consider two main performance indicators, which are also the most im-

portant parameters in the related work [40, 46, 65, 70, 74]. The first is the traffic between ASes. This value allows to evaluate the saving potential for providers, since inter-AS traffic is a major cost factor for ISPs. Since a variety of charging models exist that are used to calculate actual prices for traffic, we focus on the bandwidth itself to give a single value for comparison. We also consider the intra-AS traffic, i.e., the total amount of traffic that is kept within each AS. This allows for a better analysis of the efficiency of the different locality-awareness mechanisms. In homogeneous scenarios, we evaluate the total traffic in the complete network, while we take a more detailed view on an AS level in case of heterogeneous scenarios.

To compute the mean bandwidths for each scenario, we average the amount of traffic per link in each one minute interval per simulation run. We then compute the mean values and their 95% confidence intervals of the inter-AS and intra-AS bandwidth for the simulation runs with different seeds.

As the second performance indicator, we consider the download time for the shared file for the end users. Since this value is the most important performance characteristic once a file has been selected for download, it captures the service quality of the overlay. The mean value of the download time is shown over all peers within a specific group, i.e., with a given access speed or within the same AS. To this end, we average the values of individual peers within one simulation run, and then again compute the mean values and their 95% confidence intervals over a number of runs with different seeds. We first evaluate the considered mechanisms under ideal, i.e., homogeneous conditions to be able to determine the basic effects of the locality-awareness implementation, cf. Table 3.1. Afterwards, results using a more realistic swarm model that is based on measurement studies of live BitTorrent swarms are presented. Finally, we conduct a parameter study for the values l_{BNS} and l_{BU} on the effect of the degree of locality-awareness.

Table 3.1: *Overview on evaluated scenarios*

Scenario	Changed Parameter	Scenario Type	Section
Swarm Load	Seeding Time	Homogeneous	3.5.1
Swarm Dispersal	Number of ASes	Homogeneous	3.5.1
Inter-AS Bottlenecks	Inter-AS Link Type	Homogeneous	3.5.1
Partial Locality	Share of Locality-Promoting Peers	Homogeneous	3.5.1
Peer Distribution	Distribution of Peers per AS	Heterogeneous	3.5.2
Access Bandwidth	Peer Access Capacities	Heterogeneous	3.5.2
Degree of Locality	l_{BU}, l_{BNS}	Heterogeneous	3.5.3

3.5.1 Characterization of Locality-Aware Mechanisms

For the following experiments, we distribute the peers uniformly among the stub-ASes, and use the single access capacity class defined in Section 3.4. Thus, we can focus on the effects of the following overlay parameters on the efficiency of the locality-aware mechanisms. We consider different load scenarios, different degrees of swarm dispersal, and scenarios with bottlenecks in the core network. Finally, we judge how well locality-awareness works if not all peers implement the mechanism.

Effect of Swarm Load

In this experiment, we compare the performance of BNS and BU under different load conditions. Load here means the download capacity demand generated by the leechers in relation to the available total upload capacity, which is provided by both leechers and seeders. Thus, we vary the mean seeding time of the peers from 5 to 30 minutes to generate different load scenarios. A longer seeding time means a higher upload capacity in the swarm without changing the leecher arrival process. Therefore, longer seeding times reduce the load in the swarm, while shorter seeding times increase it.

Figure 3.5 shows the mean value of the inter-AS bandwidth for the different

mechanisms and load scenarios. To judge the share of total traffic that is inter-AS traffic, the intra-AS traffic of every mechanism is also shown on top of the inter-AS traffic bars (labeled 'Intra-AS'). Thus, the complete bar is the sum of both and therefore the total average bandwidth utilized.

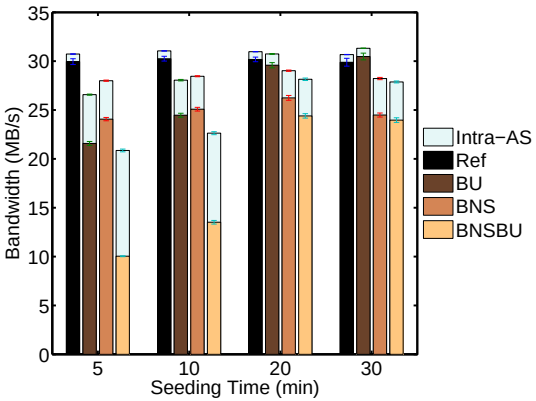


Figure 3.5: *Mean bandwidth consumption for different seeding times*

Our first observation is that the inter-AS bandwidth of regular BitTorrent is almost unaffected by varying mean seeding times. With regular BitTorrent, only a small fraction of the total traffic stays within the originating stub-AS. This corresponds to the small fraction of local neighbors of a peer, cf. Table 3.2. With BNS, a peer's neighbor set contains a higher share of local peers than with regular BitTorrent and this reduces the inter-AS traffic. The total traffic is reduced as well, because each connection between two different ASes consumes bandwidth on two links, and thus creates double the traffic that is reported for an intra-AS connection.

With BU, the amount of inter-AS traffic is smaller for short seeding times. While the inter-AS traffic is reduced significantly in the scenario with 5 min-

Table 3.2: *Mean number of neighbors of a peer which are interested (top), local (middle), and both (bottom)*

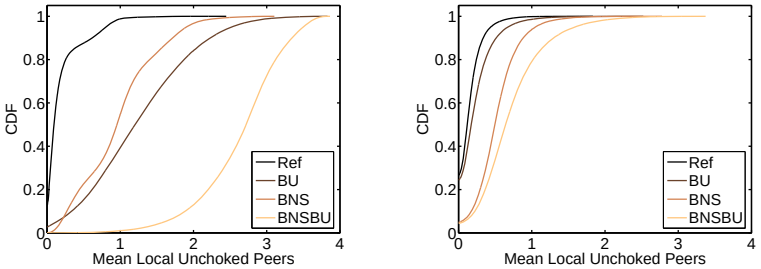
Seeding	Interested Neighbors			
time (min)	5	10	20	30
Ref	30.20	21.02	5.04	1.89
BU	30.35	20.66	4.95	1.90
BNS	30.02	20.43	4.83	1.92
BNSBU	30.03	20.67	4.79	1.90
Seeding	Local Neighbors			
time (min)	5	10	20	30
Ref	2.17	2.18	2.15	2.13
BU	2.25	2.22	2.17	2.14
BNS	6.71	6.68	7.15	9.62
BNSBU	6.85	6.79	7.19	9.71
Seeding	Local Interested Neighbors			
time (min)	5	10	20	30
Ref	1.51	1.05	0.25	0.09
BU	1.45	0.99	0.25	0.10
BNS	**4.65**	**3.19**	0.82	0.45
BNSBU	**4.46**	**3.11**	0.81	0.45

utes mean seeding time, BU has almost no effect with 20 or 30 minutes mean seeding time. This is similar for the combination BNSBU, which shows no large additional traffic reduction in comparison to BNS alone. For long mean seeding times, BNSBU cannot save inter-domain traffic. In contrast, it is especially effective in scenarios with short seeding times. There, only about half of the traffic is inter-AS traffic in our scenario. The reason is that BNS enables each peer to know the other peers in the same AS while BU assures that these peers are unchoked whenever possible.

The fact that BU and BNSBU are more effective in scenarios with high load can be explained as follows. BU and also BNSBU can only work when at least one local, interested, and choked neighbor exists in the neighbor set of a peer. Table 3.2 shows that this is only rarely the case in the scenarios with 20 or

30 minutes mean seeding time. Consequently, BU is effective when the load in the swarm is high, i.e., when peers have several interested neighbors (marked in bold). Then, they can select a local neighbor to be optimistically unchoked.

This can also be observed in Fig. 3.6, where the CDF of the average number of unchoke slots for local peers is plotted for two load scenarios, corresponding to 5 and 20 minutes mean seeding time. We can see that in the highly loaded system, BU and especially BNSBU are able to give more unchoking slots to local peers than for a low load.



(a) 5 minutes seeding time

(b) 20 minutes seeding time

Figure 3.6: *CDF of the number of unchoked slots allocated to local neighbors*

There seems to be a contradiction because BU only decides about one unchoking slot. However, optimistically unchoked peers may be unchoked regularly by the tit-for-tat mechanism after having been 'discovered' via optimistic unchoking. In this manner, BU indirectly allocates all upload slots of a peer preferentially to local neighbors.

As expected, longer seeding times and therefore a larger total upload capacity in the system lead to shorter download times, cf. Fig. 3.7. This effect would increase as long as the download capacity of the peers is not fully utilized.

However, we observe no impact of the evaluated mechanisms on the mean download times of the file. The reason for this is the fact that the only bottlenecks

in the default scenario are in the access network. Therefore, a connection between two peers in different ASes has the same bandwidth limitation as a connection between two peers in the same AS. From an end user's perspective, this means that there is no difference between a local neighbor and a remote neighbor, in contrast to scenarios with bottlenecks in inter-AS links, cf. Section 3.5.1.

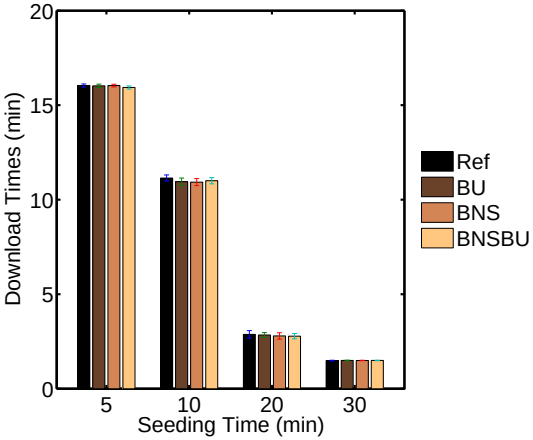


Figure 3.7: *Mean download times for different load scenarios*

Effect of Swarm Dispersal

Next, we evaluate the impact of the distribution of peers on a different number of ASes, since a smaller number of potential local neighbors means less opportunity to promote locality. To this end, we vary the number of stub-ASes in the simulated topology. Since a new peer appears in each stub-AS with equal probability, each stub-AS receives a smaller fraction of the swarm if there are more ASes. We simulate topologies with 10, 20, and 40 stub-ASes, resulting in 10%, 5%, and 2.5% of the swarm population per AS on average.

Again, we take a look at the inter-AS bandwidth savings achieved by the different mechanisms, cf. Fig. 3.8. In general, the gains made by all locality-promoting mechanisms are larger if the fraction of the swarm in one AS is large. BNS profits directly from more local peers since the share of local neighbors per peer is higher as well. Similarly, BU has a higher probability to find a local interested neighbor when there are more peers in the same AS. The combination of both mechanisms utilizes both of these advantages, leading to a further significant improvement of saved inter-AS bandwidth.

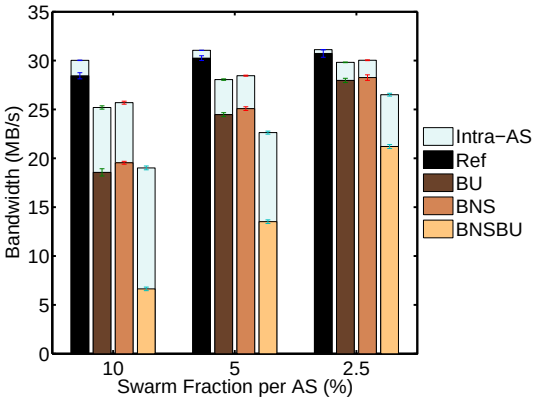


Figure 3.8: *Mean bandwidth consumption for different swarm distributions*

The inter-AS traffic reduction is decreased when the local share of the swarm gets smaller. For the scenario with an average of 2.5% of the peers in one AS, BNS and BU save only in the range of 8% of the inter-AS traffic, while BNS and BU together still reduce the traffic of regular BitTorrent by 30% in our setup. The reason is that the combination of both mechanisms tries to utilize every local neighbor. With BNS alone, the probability that a local neighbor is unchoked is small. With BU alone, the probability that a local peer is in the neighbor set is

small. Consequently, they cannot reduce inter-AS traffic alone in scenarios where only a very small fraction of the peers resides in the same AS.

Since we have no bottleneck in the network, the location of neighbors does not have an effect on the utilized download bandwidth per peer. As a consequence, the download times are affected neither by the number of stub-ASes nor by the different mechanisms. For all configurations, the mean download times correspond to the results of the default value of 10 minutes seeding time.

Effect of Inter-AS Bottlenecks

Here, we investigate the impact of inter-AS bottlenecks, i.e., bandwidth limitations of the links between the stub-AS and the transit-AS. The experiment is motivated by the fact that some providers throttle the bandwidth of P2P connections leaving their network [65].

The authors of [65] show that under these conditions locality awareness leads to a better application performance since the bottleneck link is avoided and local connections with higher throughput are preferred. To judge whether BU also works well under these circumstances, we limit the capacity of each inter-AS link in our topology to 3072 Kbit/s, i.e., three times the upload capacity of one peer. We compare the results to the scenario with no limitations on the inter-AS links, labeled 'Access bottleneck'.

The inter-domain bottlenecks result in generally lower inter-AS bandwidths for all mechanisms, cf. Fig. 3.9. No more than 7.68 MB/s can be uploaded from all 20 ASes simultaneously, because each of the 20 links from a stub-AS to the transit-AS has a capacity of only 3072 Kbit/s. Since the same traffic flows from the transit-AS to the stub-ASes, this results in a total maximum inter-AS bandwidth of 15.36 MB/s. In contrast to regular BitTorrent, the locality-aware mechanisms keep the inter-domain traffic below that limit. The reason is that inter-AS connections whose bandwidth is limited on an inter-AS link are likely to be replaced by the intra-AS connections with higher bandwidth. This is caused by the tit-for-tat policy of BitTorrent which allocates upload slots to those peers from

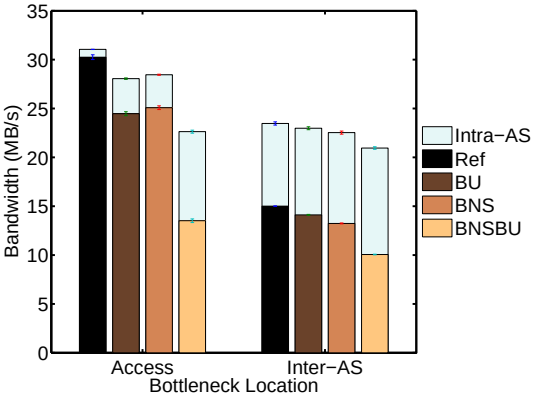


Figure 3.9: *Mean bandwidth consumption with and without inter-AS bottlenecks*

which it gets the best download speed.

With inter-AS bottlenecks, the download times are no longer independent from the mechanism, cf. Fig. 3.10, because different sources offer a different bandwidth for download. Thus, the download times for regular BitTorrent are much longer than in the scenario where connections are limited only by the access links. In this scenario, local peers with good connectivity may be discovered only via the regular unchoking process, so that many low-bandwidth connections via inter-AS links are utilized. The effective capacity of the system is reduced, leading to download times that are three times longer than without inter-AS bottlenecks in our scenarios.

The locality-aware mechanisms on the other hand foster the utilization of the better connectivity between local neighbors since these are already preferred. In our scenario, the combination of BU and BNS leads to only a slight increase in the mean download times compared to the scenario without inter-AS bottlenecks. This can be explained by the fact that the mean inter-AS bandwidth in

the scenario without inter-AS bottlenecks was already below the capacity limit introduced by the inter-AS bottlenecks. Therefore, the performance of BNSBU is only affected to a minor degree. The impact of the inter-AS bottlenecks is larger for BNS and BU alone. Still, the mean download times are considerably smaller than with regular BitTorrent. From this experiment we conclude that in case of inter-AS bottlenecks, BU improves the mean download times compared to regular BitTorrent and the combination of BNSBU leads to shorter download times than BNS alone.

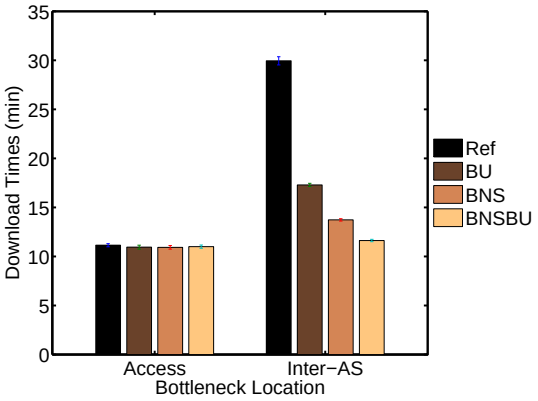


Figure 3.10: *Mean download times with and without inter-AS bottlenecks*

Effect of Partial Locality-Awareness

With this experiment, we investigate what happens if only a fraction of the peers in the swarm promotes locality, while the rest uses the regular BitTorrent implementation. We vary the share of peers that utilize a locality-aware mechanism from 0% (corresponding to the Ref case) to 100% (corresponding to the previous results). Here, we again simulate the 3 Mbit/s bottleneck in the inter-AS links.

The inter-AS traffic of the regular implementation is again capped at the bandwidth limit introduced by the inter-AS bottleneck links, cf. Fig. 3.11. The locality-aware mechanisms save some of this inter-AS traffic even if only 25% of the peers actively promote locality. The savings increase with the share of peers utilizing locality-awareness. We also see that the addition of BU again enhances the BNS mechanism, since the combination of both leads to the largest savings.

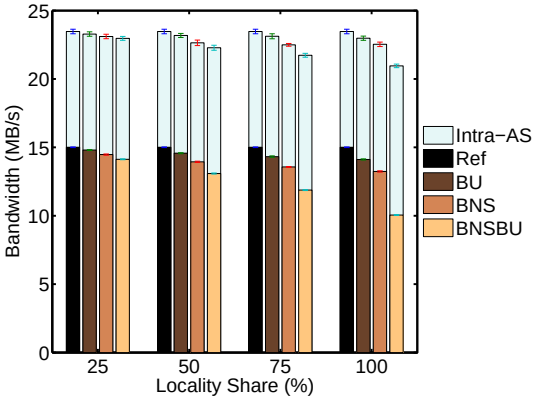


Figure 3.11: *Mean bandwidth consumption for different shares of locality promoting peers in the swarm*

As in the experiment before, the introduction of the inter-AS bottleneck has an impact on the download times of the peers, cf. Table 3.3. Here, we show the results separately for the two groups of peers, the ones that do support locality and the ones that do not. All locality-aware mechanisms lead to shorter download times than the regular implementation for both groups. Even if only a fraction of the peers supports locality, it still helps the swarm by generating new sources faster and providing more upload bandwidth to the local neighbors of the locality-promoting peers. However, BU alone performs worst of the biased algorithms.

Table 3.3: *Mean download times (in minutes) of locality promoting peers (non-locality promoting) peers for different shares of locality-promoting peers.*

Share (%)	25	50	75	100
Ref	- (29.95)	- (29.95)	- (29.95)	- (29.95)
BU	23.33 (25.26)	20.64 (21.57)	18.75 (19.01)	17.30 (-)
BNS	13.14 (22.53)	13.75 (18.78)	13.72 (16.89)	13.73 (-)
BNSBU	11.24 (20.40)	11.47 (16.49)	11.35 (14.70)	11.62 (-)

Not only do the peers supporting BU experience the longest download times, they also do not improve their performance significantly over the peers that do not support locality.

In contrast, the peers implementing BNS and the combination of BNS and BU decrease their download times of the file by more than 50% in any scenario considered here. They also perform better than the group ignoring locality, although this advantage diminishes when a larger part of the swarm is locality-aware. This again is due to the fact that regular peers also profit from the better performance of the locality-aware peers.

3.5.2 Locality-Awareness in Realistic Swarms

In the previous section, we evaluated the locality-awareness mechanisms under ideal conditions, i.e., for homogeneous access bandwidths of all peers and a uniform peer distribution. While this allows for a characterization of the algorithms, these conditions are not met in live BitTorrent swarms. Thus, we now consider swarms that are modeled after realistic swarms encountered in the Internet. These swarms have been characterized by measurement studies, such as [78] or [83].

Effect of a Heterogeneous Peer Distribution

We start by evaluating a scenario where the peer distribution among the ASes is heterogeneous. We evaluate a swarm with a skewed distribution of the peers

over the 20 simulated ASes. Measurements [78, 83] of live BitTorrent swarms show that this is a much more realistic scenario than the typically assumed even distribution of peers over the topology. Furthermore, this skewness should have an impact on the efficiency of a locality-awareness mechanism, since ASes with less peers have less opportunity for optimization, as shown in Section 3.5.1. We use a hyperbolic distribution of the peers among the ASes as proposed in [78]. Thus, we simulate a peer joining the system does this in AS k with probability $P(k)$ according to

$$P(k) = \frac{\frac{1}{k}}{\sum_{i=1}^{n} \frac{1}{k}}, k \in \{1, \ldots, n\}. \tag{3.1}$$

For the sake of readability, we use the term 'large AS' for ASes that hold a large share of the swarm, and similarly call ASes with a relatively low number of peers 'small'.

We can observe that the saving potential for inter-AS traffic grows with the share of peers in an AS, cf. Fig. 3.12. Especially when implementing locality-awareness both in the neighbor selection and in the unchoking process, larger ASes can reduce their incoming and outgoing traffic by a much larger factor than ASes with only a few peers in the swarm. If such an AS belongs to a Tier2 or Tier3 ISP which is charged by either its uploaded or downloaded traffic or the maximum of both, this translates into higher cost savings.

In contrast, ISPs with only a small number of peers per swarm are not likely to profit much from locality-awareness, simply because there are only few options for peers in these ASes to choose local neighbors. The better part of such a peer's contacts have to be from remote locations even when it promotes locality. This is in line with the results shown in Section 3.5.1.

To judge the locality-aware mechanisms in this scenario from a user perspective, we compare the mean download times for peers in the different ASes. Figure 3.13 shows the average download times of peers in the individual ASes.

When BU is used, the download times for peers in larger ASes decrease, while the peers in smaller ASes take longer to download the file. This is due to the fact a

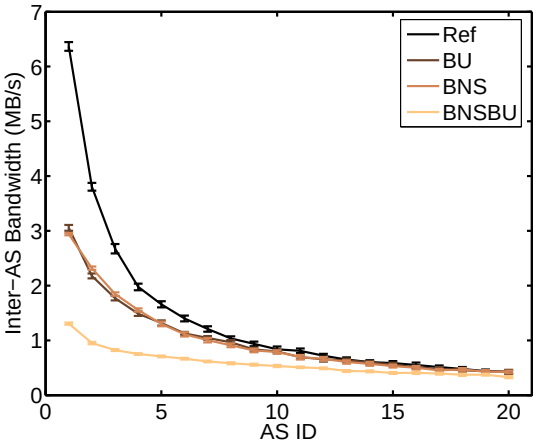


Figure 3.12: *Inter-AS bandwidth per AS for a heterogeneous peer distribution*

peer using BU preferentially unchokes local neighbors if possible, i.e, it uploads to local neighbors. However, peers in small ASes know only a small number, if any, of local interested neighbors, and therefore can only prefer them in the unchoking process in rare cases. Thus, the upload capacity of these peers is mainly distributed among all ASes. In contrast, peers in large ASes know interested local neighbors almost all the time. Consequently, they upload to a local neighbor very often. Therefore, the upload capacity of peers in a large AS is mainly utilized for connections within that AS. Furthermore, large ASes receive additional upload capacity from peers in small ASes when those peers have no interested local neighbor in their neighbor set. That shifts the global allocation of upload capacity in the swarm towards large ASes.

In contrast, BNS leads to longer download times in the largest AS in comparison to both regular BT and peers in the rest of the swarm. This effect seems counter-intuitive, but can be explained when considering the composition of the

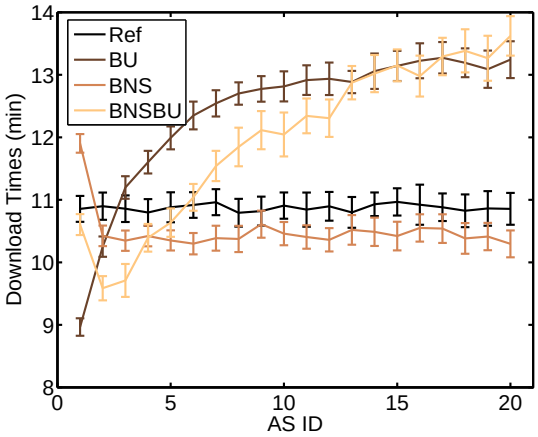


Figure 3.13: *Download times per AS for a heterogeneous peer distribution*

peer's neighbor sets. A peer in a large AS initiates connections mostly to peers in the same AS using BNS, since there are enough suitable contacts available for this. However, it still has a small number of remote neighbors because peers from other ASes initiate connections to the local peer. On the other hand, a peer in a small AS also has a high share of neighbors from the larger ASes, simply because BNS can only add the small number of contacts from the same AS and then fills the rest of the neighbor set randomly. Since more peers exist in the larger ASes, the probability for them to be chosen is also higher. This leads to a larger number of neighbors in total for peers in larger ASes, cf. Fig. 3.14. Additionally, it is more likely that a peer in a large AS is contacted by a remote peer than vice versa.

Since new peers enter the system without the file, they are interested in every other peer in their neighbor set. Let peer B join the system after peer A and recall that B is interested in A if A has chunks of the file that B still needs. Then, we

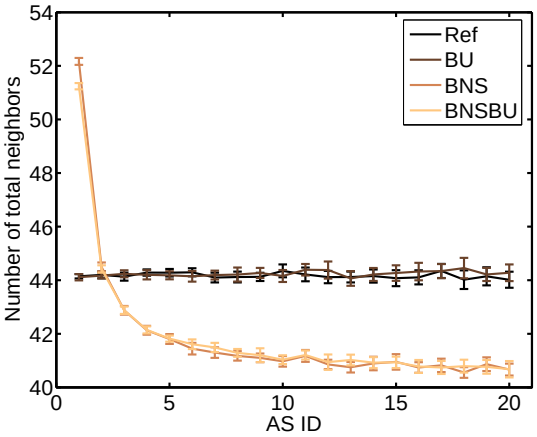


Figure 3.14: *Total number of neighbors per AS for a heterogeneous peer distribution*

can say in general that the probability that peer B is interested in peer A is higher than the probability that A is interested in B. This means that peers preferentially download data from those peers to which they initiated the connection and upload preferentially to peers which initiated a connection to them.

Since peers in large ASes are contacted by local peers as well as by remote peers from small ASes, they have more neighbors which are interested and demand upload capacity than peers in small ASes, cf. Fig. 3.15. Furthermore, peers from large ASes contact mainly local peers when they enter the system due to BNS. This means that they download mainly from local peers. In contrast, peers in small ASes contact a higher number of remote peers, including those located in the large ASes, because only few local peers exist. Therefore, they also download from remote neighbors. In addition, they utilize the upload capacity of the local peers. In summary, peers in large ASes compete for the upload capacity only

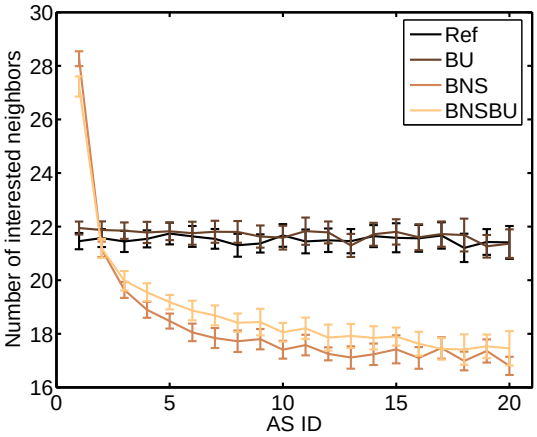


Figure 3.15: *Number of interested neighbors per AS for a heterogeneous peer distribution*

at local peers while peers in small ASes demand upload capacity in the whole swarm. Hence, on average more upload capacity is allocated to peers in small ASes and average download times can increase for peers in large ASes by using BNS.

Returning to the download times of the peers, BNSBU shows a combination of both effects described for BU and BNS. While the neighbor set composition has the same characteristics as in the pure BNS case, the unchoking policy of BU offsets the disadvantages of peers in large ASes. Therefore, the download times in larger ASes are shorter than in smaller ASes, but peers in the largest AS in our scenario still take longer to download the file than in the second largest AS.

We conclude that, for the end user, the incentive to support locality-awareness depends heavily on the type of mechanism used to do so, and on his location. It cannot be expected that a user will willingly participate in a locality-promotion

scheme if he decreases his performance. On the other hand, if a user can shorten his download times by promoting locality with certain mechanisms, not much effort will be needed to implement locality promotion.

Effect of Heterogeneous Bandwidth Distributions

The second simplifying assumption in the previous experiments is that the access bandwidths of all peers are homogeneous. We now relax this assumption by allowing peers with heterogeneous access speeds in the swarm, where the peers are again distributed uniformly over the ASes. Measurements in [58] show that the peers in a swarm can be clustered according to their access speeds. For example, 20% of the peers in a swarm have 128 Kbit/s upload capacity, 30% have 256 Kbit/s, 40% have 512 Kbit/s, and the rest is faster. The concrete numbers and cluster sizes depend mainly on the ISP where the peers are located. To keep things simple, we abstract from the concrete numbers and create two equal sized groups of peers: one with 16 Mbit/s down- and 1 Mbit/s upload capacity, as in the default scenario, and one with 4 Mbit/s down- and 256 Kbit/s upload capacity. Consequently, half of the peers in the swarm are 'fast' peers and the other half are 'slow' peers. This suffices to show the basic effect locality-awareness has on swarms with heterogeneous bandwidth distributions.

With these two access classes, we consider two scenarios. In the first one, we assign one of the access classes to each AS, which means that all peers in one AS have a homogeneous access speed, but peers in different ASes may differ in their access capacity. In this scenario, all fast peers are located in ASes with IDs from 1 to 10 and all slow peers are located in the rest of the ASes. This mimics the situation that some ISPs which are technologically more advanced than others offer their costumer higher access speeds. We denote this scenario 'Fast vs. Slow ASes'.

In the second scenario, denoted 'mixed access speeds', fast and slow peers are equally distributed among all ASes. This scenario is quite common in practice because most ISPs offer their customers a choice between different access speeds.

Scenario 'Fast vs. Slow ASes'

To give an impression of the effect of locality-awareness on the traffic, we again show the sum of the bandwidth used in the downlink and the uplink of the individual ASes, cf. Fig. 3.16. We see that fast ASes profit more from locality promotion in most cases because the peers in these ASes finish their download faster and provide additional upload capacity to peers in the slow ASes. Thus, the decrease in download bandwidth is smaller for the slow ASes.

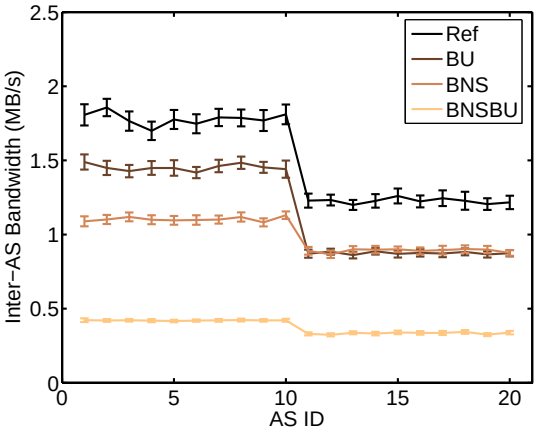


Figure 3.16: *Inter-AS bandwidth per AS for heterogeneous AS access capacities*

If the access bandwidth of the peers is tied to their location, all evaluated locality-awareness mechanisms lead to a more pronounced unfairness in the download times, cf. Fig. 3.17. While peers with a low access speed generally take longer to download the file also in the regular BitTorrent case, the difference in the download times between slow and fast peers increases significantly when the peers promote locality. This is due to the fact that fast peers tend to prefer neighbors in the same AS which, due to the considered scenario, also have

a high capacity. Thus, fast peers utilize more of their upload bandwidth to exchange data with other fast peers. Conversely, slow peers limit themselves by choosing other slow peers as neighbors and/or in the unchoking process. That is true for all investigated locality mechanisms and BNSBU, as the most stringent locality-promoting mechanism, increases the unfairness the most, i.e., it leads to the largest difference between download times of peers in the slow and peers in the fast ASes.

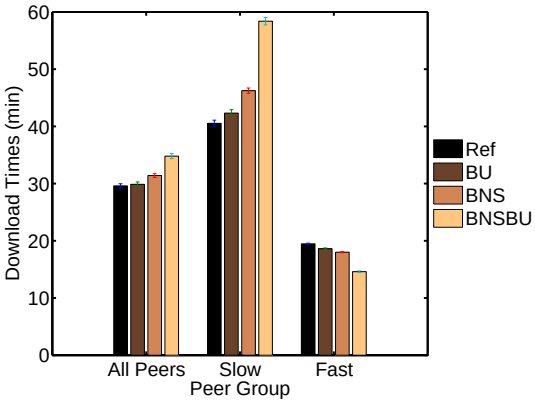


Figure 3.17: *Download times per peer group for heterogeneous AS access capacities*

Additionally, the results show that locality-awareness increases the mean download times over all peers, cf. Fig. 3.17. In other words, locality-awareness decreases the overall efficiency of the distribution process in this scenario. In general, users in ASes with a lower bandwidth than in the rest of the swarm will not profit from locality-awareness, and can therefore not be expected to adopt a locality-promoting mechanism.

Scenario 'Mixed Access Speeds'

In contrast to the previous scenario, the access bandwidths of the peers in the swarm are still heterogeneous, but both slow and fast peers are evenly distributed among the 20 ASes. As a result, the traffic savings by utilizing locality-awareness are uniformly distributed among the ASes, cf. Fig. 3.18. Again, the combination BNSBU saves most inter-AS traffic in comparison to the regular BitTorrent implementation.

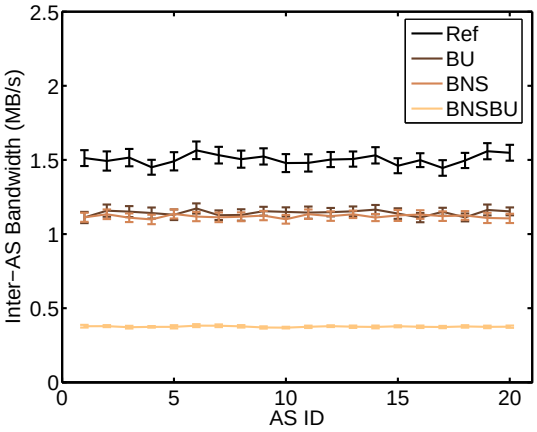


Figure 3.18: *Inter-AS bandwidth per AS for heterogeneous peer access capacities*

As expected, the mean download times for the swarm as a whole are not affected by the considered locality-awareness mechanisms, cf. Fig. 3.19. The download times are similar in all ASes as well, since there are no topological differences anymore.

While the groups of peers in the same AS experience the same average download times, the same is not true if we differentiate between access types. Figure 3.19 additionally shows the mean download times per used locality promo-

tion mechanism and for the different bandwidths of the peers. In general, the peers with the fast access take less time to download the file, which is due to their higher download capacity and because they are favored by the tit-for-tat algorithm. BNS does not differ here from the regular BT implementation. The mechanisms including BU, however, lead to shorter download times for slow peers and longer download times for fast ones. For the swarm as a whole, this can be interpreted as a fairer distribution of the upload capacity. From the viewpoint of peers contributing more resources, they have less incentive to do so if they are not rewarded. We conclude that in this scenario BU and BNSBU lead to fairer download times while they increase the unfairness in the scenario 'fast vs. slow ASes'. This shows that the actual bandwidth distribution of peers has a significant impact on the performance of locality-mechanisms experienced by the user.

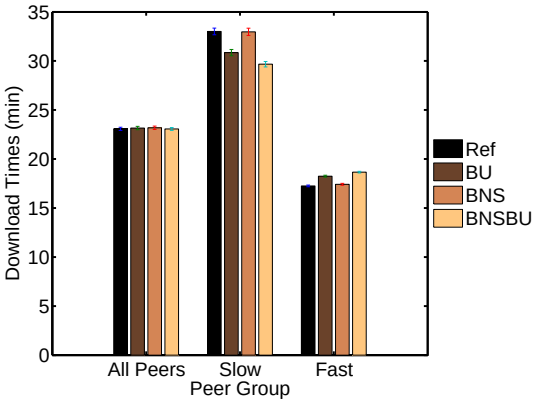


Figure 3.19: *Download times per peer group for heterogeneous peer access capacities*

3.5.3 Degree of Locality-Awareness

From the conducted experiments, we conclude that locality-awareness can under realistic conditions introduce unfairness in a swarm with a heterogeneous peer distribution. Now, we want to find out whether the degree of unfairness can be influenced by the parameters of the locality-promoting mechanism. Typically, this is a value which determines the probability that local peers are favored over remote peers. In the algorithms considered here, these are the locality values l_{BU} and l_{BNS}. Thus, we now vary both parameters, starting with l_{BU}, which leads to the most unfair download time distribution. The effect on the download times for values of $l_{BU} \in \{0.5, 0.9\}$ is shown in Fig. 3.20. For the evaluation, we use the scenario with a skewed peer distribution and homogeneous access speeds in the topology described in Section 3.5.2.

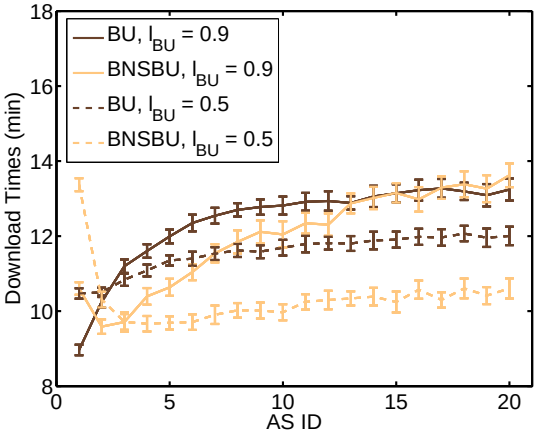


Figure 3.20: *Download times per AS for different values of* l_{BU}

With $l_{BU} = 0.5$, i.e., a less strict preference of local peers, the average download times are more uniform over the individual ASes, especially for BU alone.

For the combination of BNS and BU, the negative effect of BNS on the large ASes, described in Section 3.5.2, is offset less with this parameter. Consequently, the mean download times are equal for the small ASes, but the large ASes still are at a disadvantage.

While a lower preference for local peers can reduce the unfairness for end users, it also influences the traffic savings achieved by the locality-promotion. Figure 3.21 shows that the used inter-AS bandwidth increases with a lower degree of locality-awareness. Thus, the parameter l_{BU} can be used to influence the trade-off between unfairness in the swarm and cost savings by traffic reduction.

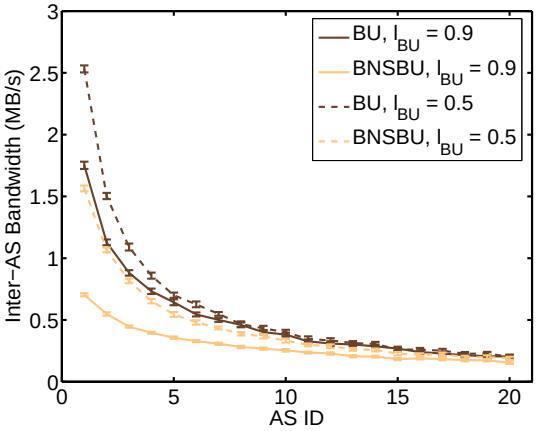


Figure 3.21: *Inter-AS bandwidth per AS for different values of* l_{BU}

A similar effect can be achieved by reducing the degree of locality in the BNS mechanism. We compare the download times of the peers in different ASes for the locality-promotion schemes BNS and BNSBU with $l_{BNS} \in \{0.5, 0.9\}$. The results are shown in Fig. 3.22. We observe that the significant increase in the average download time for the largest AS vanishes for $l_{BNS} = 0.5$, i.e., a lesser

degree of locality. With BNS alone, the download times are fairly distributed, while the combination of BNS with BU shows the heavy unfairness of the BU mechanism described earlier. This is due to l_{BU} having the comparably high default value of 0.9 in this experiment.

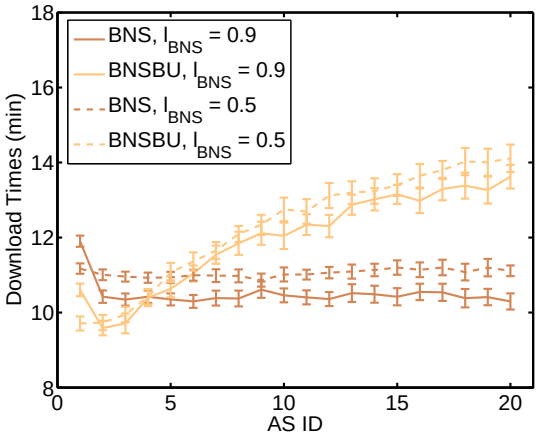


Figure 3.22: *Download times per AS for different values of* l_{BNS}

Again, the better fairness achieved by promoting locality less is paid for by less savings in traffic, cf. Fig. 3.23. The largest AS, which no longer experiences any disadvantage from a user's point of view when $l_{BNS} = 0.5$, now consumes more inter-AS bandwidth.

These results show that more conservative parameters might mitigate the negative effects of locality-awareness but reduce simultaneously the amount of inter-domain traffic that can be saved.

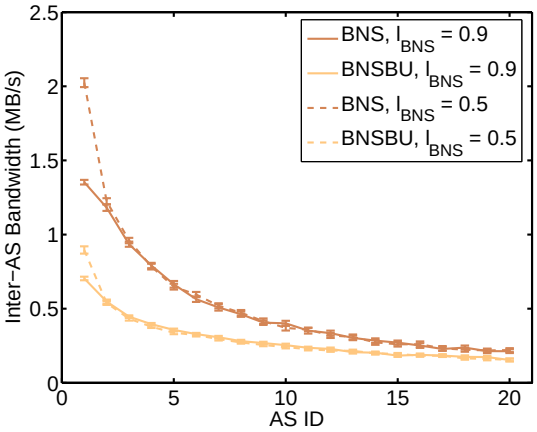


Figure 3.23: *Inter-AS bandwidth per AS for different values of* l_{BNS}

3.6 Lessons Learned

Regarding file-sharing overlays, we can conclude that it is indeed possible to reduce costly cross-ISP traffic using simple metrics and straightforward client adaptations. Especially when combining a locality-aware neighbor selection and a locality-aware unchoking mechanism, inter-AS traffic can be significantly reduced. This is true both in comparison to the default implementation as well as in comparison to the existing approach of neighbor selection alone. The combination works best under high load conditions, i.e., when the upload capacity in the swarm is scarce. A necessary condition for large traffic savings are a sufficient number of peers in one AS, since otherwise it is impossible to prefer local peers over remote ones. The larger the local share of a swarm is, the more traffic can be saved.

While locality-awareness is nearly always beneficial for providers, this is not

true for end users. It strongly depends on the swarm characteristics and on the AS a user is attached to whether a locality-aware mechanism shortens the download time or increases it. In swarms where the peer distribution as well as the access bandwidths are homogeneous, an end user may only profit from locality-awareness if the inter-AS links constitute bottlenecks. If the access networks are the bottlenecks, then locality-awareness does not influence the download times.

This changes in more realistic swarm scenarios. Live BitTorrent swarms are highly heterogeneous, both with respect to the distribution of peers among the Internet topology as well as the access bandwidth of peers. Under these conditions, locality awareness leads to different results for different groups of peers.

Heterogeneous access bandwidths only have a minor effect if the distribution of the different capacities is equal for all ASes. In this case, locality-awareness does not lead to a difference in download times between these ASes. In contrast, if the access technology differs between ASes, then locality-awareness in conjunction with the BitTorrent standard tit-for-tat mechanism leads to an even faster download for high-capacity peers. This is paid for by lower download speeds for the peers that are in any case hampered by a low access capacity.

For a heterogeneous peer distribution as observed in the Internet, where a few ASes hold a significant share of a swarm and the rest of the peers is dispersed among several ASes, the different mechanisms show varied effects. Biased Unchoking favors peers in large ASes, leading to a faster download at the cost of longer download times for peers in small ASes. Biased Neighbor Selection, on the other hand, has an inverse effect only for a small number of the largest ASes. Both is in contrast to the default BitTorrent implementation, in which all ASes show the same average download times. Thus, the stated aim of giving all end users incentives to promote locality is not met by the considered approaches.

However, this unfairness for the end user can be decreased by adjusting the degree of locality-awareness. Lesser enforcement of the preference of local peers leads to a fairer distribution of download times, but also reduces the amount of saved inter-domain traffic. This shows that the parameters of the considered locality-awareness mechanisms can be used to tune a trade-off between efficiency

for the ISPs and the utility of the overlay for the end users.

To conclude, locality-awareness does not necessarily lead to an improvement for all involved parties. Especially the effect on the application performance for the end user has to be monitored if a locality-aware mechanism is implemented. If users are meant to participate of their own accord in a locality-aware traffic management scheme, the algorithms have to be tuned in order to be beneficial or at least not detrimental for the users. Otherwise, different means of compensation have to be found by ISPs in order to implement such an architecture without resistance of the overlay users.

4 Video Streaming Overlays

> *All of the books in the world contain no more information than is broadcast as video in a single large American city in a single year. Not all bits have equal value.*
> Carl Sagan (1934 - 1996)

Next to file-sharing, video streaming is the second largest contributor to the total consumer Internet traffic by the time of this publication, estimated to amount to 2.4 TB in 2009 [75]. While file-sharing traffic is expected to grow, the share of video streaming will grow even faster in the next few years, replacing file-sharing as the top consumer traffic source by 2011.

The large bandwidth consumption of video transmissions is due to the fact that videos in a good quality are of a large size in comparison to music or e-books. This means that file-sharing networks used to distribute this larger content have to be more efficient, as shown in the last chapter. However, using a pure file-sharing application, a video has to be downloaded completely before it can be watched. In contrast, video streaming allows a user to watch the video while it is still being downloaded. Therefore, we distinguish between video distribution by file-sharing and by streaming mechanisms.

In video streaming, there is again a separation into two classes of streams, namely live streaming and video-on-demand streaming [71]. In live streaming, the content is played out roughly at the same time to all watching users, similar to current TV programs. Thus, all clients are interested in the same piece of data at the same time. This enables broadcasting content that is generated in real-time by the source. In this case, the delay between the source and the clients is a

measure of interest. For instance, users watching a live soccer game do not want to see a goal scored a significant time after their neighbor cheers.

The second class of video streams is the so-called video-on-demand (VoD) streaming. Here, the complete video content is finalized and available as a whole before it is streamed, e.g., a movie library a user can choose from. Users can request this content at any time, meaning that different users show a different progress in the video and are therefore interested in different portions of it for playout in the near future. Moreover, the video playout can be fast forwarded or 'rewinded', similar to watching a DVD. In this chapter, we focus on VoD streaming applications.

The means to provide video content are similar to the ones used for simple content distribution, although the transmission techniques may be more sophisticated. Again, the most straightforward solution is to provide one or several streaming servers which upload the video data to clients, as it is done by popular video portals such as YouTube [92]. Due to the large bandwidth demands of video streaming, large server and upload capacity is necessary, leading to high infrastructure demands and consequently high costs [71]. As in file-sharing, this promotes concepts utilizing the resources of the clients.

Thus, there exist P2P solutions that allow for videos to be streamed in an overlay. Due to their cost advantage, it can be expected that the already popular overlays will continue to grow. Thus, their efficiency is an important research topic, both due to the generated traffic as well as due to the problem of keeping end user satisfaction on a high level. P2P video streaming shares many characteristics with its cousin P2P file-sharing, including the traffic management challenges discussed in the last chapter. The solutions considered there can easily be applied to many streaming overlays as well. However, in contrast to file-sharing, the service quality of a video streaming application is highly sensitive to inefficient resource usage [94]. In file-sharing, where download times of several minutes up to hours are common [36], a delay in the range of a minute does not affect the user satisfaction to a high degree. When watching a video, a forced pause in the stream in the same time range can lead to a cancellation of the playout, since the quality is

experienced as too low. Thus, an overlay solution has to take into account higher demands by the supported video streaming application.

In the following, we discuss in Section 4.1 additional challenges for video streaming overlays that stem from the growing heterogeneity of end user devices used for watching a streamed video. Then, we review video codecs and P2P overlays from literature that support video streaming in Section 4.2, focusing on VoD applications. Our own solution for the support of heterogeneous clients is presented afterwards in Section 4.3. For the evaluation of this concept, we conduct a simulation study that tests how well heterogeneous clients are supported by this solution, and if it efficiently and automatically reacts to changes in the overlay. To this end, we first describe in Section 4.4 the used model and the most relevant parameters of the system. We then evaluate the results of a number of parameter studies and implementation alternatives in Section 4.3 and give recommendations for the design of an adaptive VoD streaming overlay. Finally, we summarize our findings in Section 4.6.

4.1 Challenges in Video Streaming Overlays

As stated above, the efficiency of a video streaming architecture has an immediate effect on the video quality experienced by the end user. Compounding this problem is a growing heterogeneity of end user devices that a video may be streamed to. A video streaming application client may be installed on a set-top box connected to a Fiber-to-the-Home (FTTH) link and to a High-Definition (HD) TV device, requesting and being able to play out the highest available video resolution. At the other end of the spectrum, mobile devices like Personal Digital Assistants (PDAs) or mobile phones now have the capability to display videos with a lower resolution [93]. These devices may be connected via wireless networks with comparably low bandwidth. Thus, it is necessary to provide a different video stream to these devices than to a fixed-line high-resolution client.

This can be managed in VoD systems by generating a set of source video files, one for each class of supported clients [22]. The end user or his applica-

tion then has to select the stream that is proper for his device. However, this does not work very well for a P2P solution, since data from different streams cannot be exchanged and therefore different clients cannot utilize each other's upload capacity. Thus, the high upload capacity of fixed-line clients is lost for low-bandwidth mobile devices. One solution for this is to take advantage of scalable codecs that allow to extract substreams in different levels of quality, such as the codec described in the next section.

Such an overlay has to ensure that a client's bandwidth is not wasted to download substreams that cannot be played out. Instead, a base quality that is sustainable by the client needs to be supported by the overlay mechanisms, and additional data should only be downloaded when this quality is achieved. Otherwise, the upload bandwidth of the peers is utilized inefficiently and the resulting overall quality is lower than possible.

Moreover, clients in such an overlay have to be able to adapt to changing network conditions, such as the capacity changes due to churn or due to a change of a wireless access network. Otherwise, mobile clients cannot be seamlessly integrated into a distributed streaming system, but have to be managed separately, at the cost of a higher complexity.

While the normal process of clients going on- and offline does not significantly change the total capacity of the overlay, most streaming overlays are supported by dedicated seeds or servers that can fail as well. This should not lead to the streaming service being unavailable. Instead, the video quality should automatically be tuned to a supportable level, without needing manual input from the user. Thus, a single client software can be used for all types of end user devices, and the management overhead by the content provider can be kept to a minimum. Still, it is necessary that the resource distribution, i.e., the allocation of upload bandwidth to requesting clients, is fair in the sense that peers contributing more are rewarded with a better video quality. Otherwise, an incentive to provide upload bandwidth would be missing, resulting in a reduced peer capacity of the network and therefore higher costs for the content provider.

4.2 Background and Related Work

In this section, we first describe approaches implementing VoD streaming functionality in a P2P overlay, to show the similarities and differences to our architecture. Then, we provide more details about the specific BitTorrent derivate Tribler, which we use as the basis for our approach to support heterogeneous video quality in a P2P streaming overlay. Since we introduce mechanisms to support scalable videos into the Tribler overlay, we describe the used Scalable Video Codec (SVC) extension of the H.264/AVC codec. Then, we review related work which also uses scalable video codecs, although for P2P live streaming and not for P2P VoD.

4.2.1 VoD Streaming via P2P Overlays

Since P2P file-sharing has proven its efficiency in practical use, approaches utilizing its scalability properties and self-organization features have been proposed in literature to support VoD streaming as well.

A VoD streaming system based on a random mesh overlay network is evaluated in [44]. Starting from a simple file-sharing network, enhancements in the chunk selection process are introduced and compared with respect to the playout rates. In contrast to other systems, the initial server is allowed to determine the chunks to be uploaded in some of these strategies. The evaluation shows that a hybrid server strategy, which considers the system as a whole, performs best. This strategy both tries to upload chunks sequentially, and additionally considers the rarity of blocks in the neighborhood of a requesting client. Regarding client strategies, a preference of chunks close to their playout deadline performs better than a random or rarest first strategy. In both cases, the addition of network coding improves the performance. Here, peers forward linear combinations of a subset of chunks they have downloaded, increasing the usefulness of data for other peers and easing the chunk selection process.

The evaluation in [44] uses a simulator that does not consider underlay effects.

The peers have an upload capacity of 500 Kbit/s in a homogeneous scenario, while the capacity is varied between freeriders with no upload and high capacity peers in a heterogeneous scenario. For the latter, no statistics for the different classes are shown. Peers have a low number of four to six neighbors, in contrast to currently wide-spread implementations of both file-sharing and streaming overlays which use a much larger number. The assumed setup times, i.e., the length of the initial buffering phase, are assumed to be 10 to 15 minutes long.

A BitTorrent Assisted Streaming System for VoD (BASS) is presented in [48]. It is argued that BitTorrent is not suitable for streaming due to its design, and therefore it is used only in addition to a dedicated streaming server. Thus, the BitTorrent protocol is not changed except that a client does not request chunks containing frames with a playout time before its current position in the video. The chunks downloaded via BitTorrent are stored for future use, while all parts of the video that could not be retrieved via the P2P overlay are downloaded from the server following an in-order strategy. It is assumed that only a small number of BASS users exist, since otherwise the assumption of chunks being available with equal probability would not hold.

For the evaluation of BASS, a queueing system simulation is used that is based on measurement data from a BitTorrent file-sharing swarm. The evaluation shows that for the investigated scenario, the necessary streaming server bandwidth can be reduced by one third through the support of a P2P overlay. The waiting time of clients can be reduced as well, both in comparison to a pure BitTorrent overlay and to a pure client-server scenario.

In [53], the assumption that BitTorrent cannot be modified for streaming is tested. An enhanced BitTorrent protocol named BiToS is presented and evaluated, which uses a mixture of in-order and rarest-first chunk selection mechanisms. Different implementation alternatives for this new chunk selection strategy are compared. However, the chunk selection is the only mechanism that is adapted and it is stated that other changes to the protocol are unnecessary. The considered changes are the separation of the chunks that still have to be played out into a high priority set, containing chunks close to their playout deadline, and

the remaining pieces. With probability p, a chunk from the high priority set is requested, otherwise a chunk from the remaining set is chosen. A strategy downloading the chunks in the high priority set sequentially is compared to the known rarest-first mechanism in both sets with $p = 0.8, 1$.

The evaluation uses a simulation of a flash crowd scenario with 400 peers watching a 10 minute video. The rarest-first strategies are shown to perform better than the sequential one, especially for a high priority set containing about 8% of the complete file. In this case, the setting $p = 0.8$ outperforms the system with $p = 1$. A dynamic adaptation of p to the swarm situation is considered, but not implemented or tested.

Toast, another BitTorrent-based support scheme for VoD servers, is presented in [56]. Similar to BASS, it shifts load from the dedicated VoD servers to a BitTorrent overlay, with parts of the video that cannot be delivered by the overlay being requested from the server. In contrast to BASS, the evaluation is based on a measurement study of an implemented system. Different chunk selection mechanisms are compared, namely the rarest-first strategy from standard BitTorrent, in-order download, a strategy based on a beta distribution for the chunk preferences, which assigns higher priority to chunks closer to playout, and finally a hybrid strategy mixing the in-order and the beta strategy by introducing priority sets.

The main performance indicator observed in the evaluation in [56] is the load reduction for the VoD server, i.e., its upload traffic savings by utilizing the overlay network. The measurement experiments are conducted for a swarm with 300 clients, which show a deterministic inter-arrival time in most scenarios. The upload capacity of the clients is varied between 1 Mbit/s, and 2 Mbit/s, i.e., the stream bit rate. Finally, different seeding behaviors are considered. The results show that longer seeding times lead to less server load, due to the larger share of available client upload capacity, and that propagation delays in the network have no impact. The load on the VoD server is reduced by up to 90% with the hybrid strategy and the in-order strategy performing better in most scenarios than the beta and the rarest-first strategy.

All of these systems, as well as the Tribler client described next, implement a VoD streaming service utilizing P2P technology. However, they are all based on a single-layer video file. We extend this functionality to scalable video codecs in Section 4.3, utilizing the properties of a SVC video. Thus, we can add support of streams with different bandwidths while being able to share all these streams in a single overlay.

4.2.2 VoD Support in Tribler

Our P2P VoD architecture is based on Tribler [73], which in turn is a BitTorrent adaptation that offers video streaming capabilities. In the following, we give an overview on the key mechanisms of Tribler and which of these differ from regular BitTorrent. In particular, these are the two most important mechanisms of BitTorrent presented in the last chapter, namely the tit-for-tat strategy used in the unchoking process and the chunk selection strategy.

Tribler uses the same structure for a shared file as BitTorrent. The complete file is separated into chunks and blocks that contain the frames of the video to be streamed. The frames are made available to the buffer of a software video player once they are downloaded. The mechanisms used for swarm and neighbor management are the same as in BitTorrent as far as our approach is concerned. However, Tribler in its current form tries to integrate several additional functionalities like user preferences, search, trust relationships and tracker distribution by heavily utilizing social aspects. Since these mechanisms do not affect the streaming mechanism itself, we skip their description for the sake of conciseness.

Chunk selection

The main difference between a file download functionality as offered by BitTorrent and a VoD service as offered by Tribler is that a user of the latter watches the video while downloading it. Thus, the timing for downloading the parts of the complete video file becomes critical, while chunks can be downloaded in any

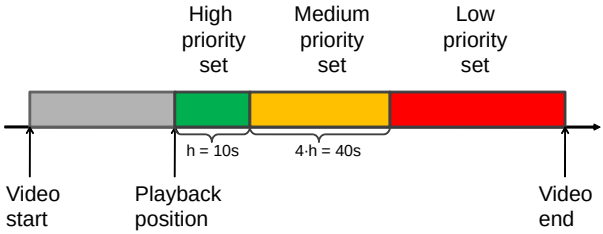


Figure 4.1: *The Tribler chunk selection mechanism*

order in a file-sharing overlay. In particular, chunks in Tribler need to be downloaded roughly in order so that a continuous playback of the video is ensured.

To this end, the rarest-first chunk selection of BitTorrent is replaced by a strategy based on priority sets, cf. Fig. 4.1. From the current playback position, all chunks until the end of the movie are separated into three sets. The high-priority set contains all chunks with frames from the playback position until 10 seconds after, while the mid-priority set contains the following 40 seconds of the movie. The rest of the chunks is in the low-priority set. Chunks are first downloaded from the high-priority set, following an in-order strategy within that set. When no more chunks can be requested in that set, the chunks in the mid-priority set are downloaded according to the BitTorrent rarest-first mechanism, and finally the chunks of the lowest priority, also following the rarest-first strategy.

Unchoking

The tit-for-tat strategy employed to rate peers in the unchoking process of BitTorrent is based on the assumption that peers can exchange content, i.e., both neighbors forming a connection have some content the other needs. Since the order in which peers download chunks does not matter in the file-sharing application, this is true for enough overlay neighbors. However, with the application VoD peers need to download chunks in roughly the order they are played out.

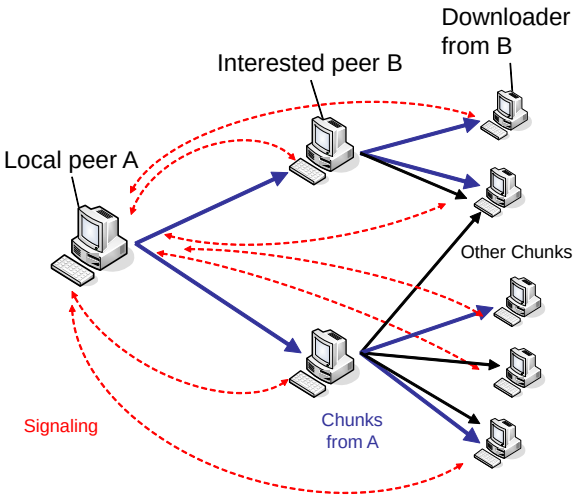


Figure 4.2: *The give-to-get mechanism*

Therefore, peers that played back a longer part of the video do generally not need any chunks from peers that are 'behind' them in the playback process.

Thus, it is much more probable that data exchange happens only in one direction of an overlay connection in Tribler. This is taken into account by replacing tit-for-tat with a strategy named give-to-get that favors peers that show a good upload behavior to other peers instead of to the local peer [72]. Consider the local peer A, which has chunks a remote neighbor B wants to download. B then reports to which other peers it has uploaded data in the last δ seconds (by default, $\delta = 10s$). Then A queries these peers to make sure that B does not exploit the mechanism, cf. Fig. 4.2.

The rating value of B then is the amount of data it forwarded that it originally received from A. If there is a tie using this metric, B's total upload is considered as a tie-breaker. This ensures that peers with a high upload capacity are not un-

choked by a large number of neighbors, but instead only by a selected subset. The three peers with the highest rating are unchoked by A every 10 seconds, similar to BitTorrent. The optimistic unchoking of a random peer is practiced as well, cf. Section 3.2.

Give-to-get thus rewards peers that upload data and aims at discouraging free-riding, similar to tit-for-tat. However, it needs much more signaling overhead and statistics for a peer's neighbors than the simpler tit-for-tat.

Playout strategy

In case frames are not received in time for their playout, streaming clients have two basic choices, frame dropping and stalling. With the first alternative, the missing parts are skipped and the player continues with the next available frames, leading to artifacts in the video picture and 'jumps' in the playout sequence. Stalling, which is the mechanism implemented in Tribler, means that the player waits for the missing frames, forcibly pausing the video during the waiting time. In this case, the only experienced degradation in the video quality is the interruption of the watching process.

4.2.3 Scalable Video Codecs

After introducing the relevant P2P VoD streaming architectures, we now consider the video codecs necessary to implement scalable video streaming in these overlays. One of the most widely-used codecs today is the H.264 ITU-T standard [39], also released as MPEG4/AVC by the ISO. It offers a much higher coding efficiency than older standards and therefore allows the efficient encoding of current HD content. However, in its original form, i.e., H.264/AVC, it only supports a single layer video. As a consequence, different video files have to be provided to support different stream bit rates and qualities.

The SVC extension of H.264/AVC [62] enables the encoding of a video file at different qualities within the same layered bit stream. This includes besides different resolutions also different frequencies (frames displayed per second) and

different qualities with respect to the Signal-to-Noise Ratio (SNR). These can be considered as a special case of spatial scalability with identical picture size for base and enhancement layers. These three dimensions of enhancements are denoted as spatial, temporal and quality scalability.

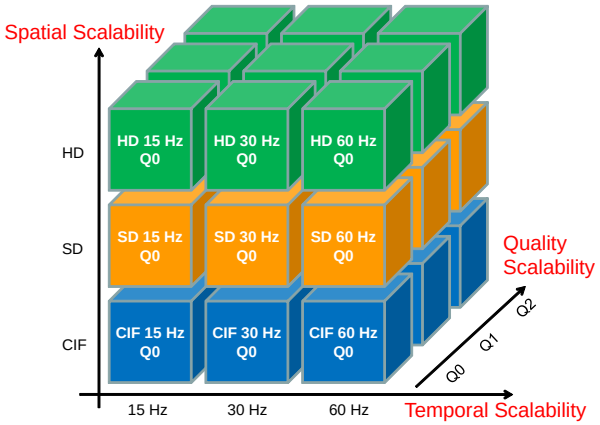


Figure 4.3: *SVC cube, illustrating the possible scalability dimensions for a video*

Figure 4.3 gives an example of different possible scalability dimensions for a video file. The scalable video file can be watched in three different temporal resolutions (15 Hz, 30 Hz, 60 Hz), three different spatial resolutions (Common Intermediate Format (CIF), Standard Definition (SD), High Definition (HD)) and three different quality resolutions (Q0, Q1, Q2). Each of the 'subcubes' represents a substream of the complete video stream in the best quality.

The left bottom 'subcube', CIF resolution with 15 Hz and quality Q0, is the base layer. All other enhancement layers reference the base layer. Therefore, no video can be played out if the necessary base layer frames are missing. Thus, each client should at least support the bit rate of the base layer to be able to stream the video.

Based on this layer, different types of enhancement layers permit a better video experience with a higher resolution, better SNR or higher frame rate, respectively. The more enhancement layers are available along any of the three axes, the higher the quality in this respect is. If all enhancement layers are available the video can be played out in highest overall quality. If all enhancement layers within quality Q0 are available, the video can be played back in HD-resolution with 60 Hz, but only with a low SNR quality.

Every enhancement layer references all lower enhancement layers and the base layer. Thus, layer n of any scalability dimension can only played out if all $n-1$ lower layers of that dimension, including the base layer, are available as well. This indicates a priorization of the layers according to their index. A client should always strive to download the base layer before considering requesting the enhancement layers, in ascending order.

Format of a Video with Temporal Scalability

Since we consider the temporal scalability feature of the SVC codec in particular, we describe it in more detail in the following. Temporal scalability implies that the frames of the complete video can be separated into layers, with each additional layer doubling the frame rate of the video, cf. Fig. 4.4. A typical H.264 video contains three types of frames, namely intra-coded (I-) frames, prediction (P-) frames, and bidirectional (B-) frames. Frames of the first type, also called key frames, are standalone pictures that can be displayed by the video application without needing any other frames. P-frames need information from earlier I- or P-frames, but can therefore also be smaller. Finally, B-frames use information both from earlier and later I- or P-frames, needing even less data in the frame itself.

The base layer of a temporal scalable video contains its I- and P-frames, while the enhancement layers are made up from B-frames. Since the enhancement layers are referencing all layers below them, they cannot be played out without these layers. Only the base layer contains frames that exclusively reference frames in

the same layer, meaning that this layer can be played out by itself.

However, the base layer also contains the largest frames, since I- and P-frames are typically larger than B-frames. Thus, there are differences in the bit rate of the respective substreams.

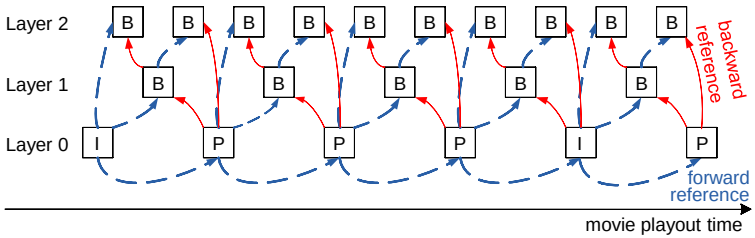


Figure 4.4: *Frame structure of a video with temporal scalability*

4.2.4 Integrating Scalable Video in P2P Overlays

An overview of adaptive video streaming techniques utilizing P2P overlays is given in [61]. Two methods to adapt the video quality and the bandwidth demand of a stream to network conditions are identified, namely SVC and Multiple Description Coding (MDC), which can be achieved by using Forward Error Correction (FEC). With MDC, the stream is encoded into several independent descriptions, and the received quality is proportional to the number of received descriptions. MDC is less efficient in terms of compression overhead than SVC, but more resilient against packet loss. It is especially useful if multi-path transmissions are used for the content, e.g., in a live streaming overlay utilizing multiple tree structures. Here, a part of the stream is transmitted in each tree, so that one description per tree can be used.

In general, the path diversity inherent in P2P overlays is concluded to be an

advantage for streaming applications. Regardless of the structure of the delivery network, i.e., whether it forms a tree or a mesh, multiple paths to the receiver enable bandwidth aggregation, packet loss de-correlation and delay reduction. It is also stated that receiver-driven approaches, where the end-points of the stream coordinate the streaming process, are a suitable strategy for streaming. However, optimal decisions can only be taken if global network knowledge is available, which is an unrealistic assumption.

A P2P live-streaming system utilizing SVC is presented in [55]. First, an overlay forming multiple trees and distributing a H.264/AVC video via this multi-tree, and a single-tree SVC overlay are compared theoretically. It is shown that the overlay based on the SVC video allows for a less complex distribution approach while delivering a better quality and being fairer when the clients show a heterogeneous capacity. Fairness here means that peers receive a quality proportional to their bandwidth contribution.

Next, a version of the Stanford Peer-to-Peer Multicast (SPPM) protocol adapted to support SVC video is used for a measurement study and compared to a standard AVC version. The video uses the temporal scalability feature of SVC, consisting of the base layer and one enhancement layer. An overlay network consisting of 100 peers is measured. The results show that the SVC version of the overlay consistently outperforms the AVC version with respect to quality, packet losses and stalling occurrences when upload capacity is scarce in the system.

The systems in this section incorporate scalable or multiple description video codecs. However, they are all designed for a live-streaming application, which differs from a VoD system. In a live streaming system, all peers demand the same content at roughly the same time. In contrast, clients in a VoD system may request the video at arbitrary times and are therefore interested in different parts of the video. Thus, the resulting demands on the chunk selection strategy differ, necessitating a different approach for VoD streaming. For this, we present our own solution in the next section.

4.3 Quality of Experience-Awareness in VoD Streaming Overlays

In order to support heterogeneous video quality, we consider the temporal scalability defined by the SVC extension of the H.264/AVC codec, as described in the last section. Measurements show that an increase of the frame rate leads to the best trade-off between a higher QoE of the end user and bandwidth preservation [93]. In order to be able to support SVC videos with temporal scalability, we change the format of the file that is exchanged in the Tribler swarm, as well as the chunk selection strategy. We describe these changes in the following.

We limit our evaluation to a video with three layers, the base layer and two enhancement layers. However, the approach described next is easily extensible to a video with an arbitrary number of layers.

4.3.1 Shared Video File Adaptation

In both BitTorrent and Tribler, one file is shared per swarm, which is separated into chunks and blocks. We adapt this mechanism by logically separating the video file to be shared into its layers. Conceptually, we treat each of the layers as a separate file to be shared. Each of these files, which contains either the base layer or an enhancement layer, comprises its own set of chunks and blocks, just like for a single file to be shared. This allows clients to easily differentiate between content belonging to different layers, while the basic mechanisms for data transfer with the exception of the chunk selection remain unaffected. Thus, the additional complexity by adding scalable videos is minimized.

The chunks of each layer are of the same size in our mechanism. Due to the different sizes of the frame types, this leads to a different number of frames being grouped in one chunk. While this means that chunks with the same index in different layers do not necessarily hold video data for the same time period, this approach enables a fragmentation of the video independent from the streamed content. It also simplifies the chunk selection, since the algorithm does not have

to take into account chunks of varying sizes.

Accordingly, the information in the torrent-file has to be structured to reflect this format and to allow peers to derive the number of chunks to download per layer. Thus, each layer has an ID that, together with the chunk ID within a layer, allows to identify all chunks of the complete movie file.

4.3.2 Adapted Chunk Selection Strategy

Our aim is to download chunks so that a video quality is attained that can be supported by the network capacity. Therefore, we prioritize lower layers over higher ones, while still keeping the set separation and general in-order download strategy of Tribler, cf. Fig. 4.5. In the high-priority set, we only download the base layer of the video to make sure we can always play out the video and to avoid stalling times. Thus, if chunks from the base layer that are close to their playout deadline are missing, they have the highest priority to be downloaded. As a consequence, a client never downloads the enhancement layers from the beginning of the video, which is a minor limitation of our architecture and could be circumvented by a longer buffering time.

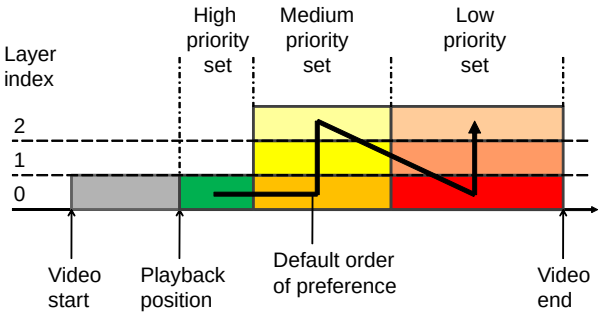


Figure 4.5: *Adapted chunk selection priority sets*

Beginning with the second or medium-priority set, we first download chunks of the base layer according to the rarest first policy within the set. If all of these are already downloaded or no chunk from that layer can be selected, we start downloading chunks from enhancement layer 1, also choosing the rarest chunk in that set first. In general, we only download chunks from a higher enhancement layer if all chunks of the lower layers are locally available or cannot be selected, with one exception that is described below. The priority of the different chunks thus reflects their relative importance, both in relation to the playout deadline as well as with respect to the layer they belong to.

This strategy only relies on the current download state, i.e., the available and missing chunks, in order to select the next chunk to be requested. Therefore, no additional measurements like average QoE or download bandwidth are used to influence the chunk selection. This keeps the complexity as low as in current overlays.

Finally, we have the same selection process in the low-priority set, which is only considered if all chunks from the other two sets are downloaded or unavailable from other peers. We adapted the sizes of the priority sets to 180 s and 360 s for the first and the second priority set, respectively.

During our evaluation, we found that due to the strict prioritization of the base-layer chunks, the content from the enhancement layers was distributed much less, which led to a lower overall quality of the video at the peers. In order to utilize seeders better, we introduce a check whether a local peer is unchoked by a seeder or not. By default, the local peer prefers enhancement layer chunks if it is unchoked by a seeder, if it cannot download a chunk from the high-priority set. It will still download base layer chunks in the mid- and low-priority sets from seeders if no enhancement layer chunks are eligible. We will compare the strategies with and without seeder distinction in Section 4.5.

4.3.3 Playout Strategy of a Scalable Video

Since we want to consider the quality of the video playout in our evaluation, we need to take into account the playout strategy of the player as well. In Tribler, the video playout is stalled whenever frames are not available in time for their playback.

Since we want to exploit the properties of a scalable video file, we adapt this strategy to only stall the video if frames from the base layer are missing. In case frames from enhancement layers are missing, the client plays out the video with the highest number of layers attainable with the locally existing blocks, i.e., it does not wait for missing frames from enhancement layers. As a consequence, a peer that finishes watching the complete movie does not necessarily store the complete video file, since it might have missed some chunks from enhancement layers.

By default, no chunks are requested that cannot be played out. Thus, a peer might never become a seed in the BitTorrent sense of the word. To reflect this difference, we therefore use the expression 'serving time' instead of seeding time for the interval a peer has finished watching the movie but is still online and providing its completed chunks to the swarm.

Additionally, we buffer the video for 60 s before starting to play it out. Since we do not consider a Constant Bit-Rate (CBR) video, but dimension the system according to the mean bit rate, we aim at having enough data available so that the varying data rate of the video does not lead to a buffer under-run during playout. We consider this waiting time to be tolerable for end users for videos of sufficient length.

4.4 Simulation Model

In our performance study, we simulate one Tribler swarm for 5 hours in the steady state. It shares an SVC video file with temporal scalability, as described in the last section. We use a self-written Java simulator that includes all key mechanisms

of Tribler and our adaptations. If not stated otherwise, we have two classes of peers with different upload bandwidths of 128 Kbit/s and 192 Kbit/s, which are in the following respectively denoted as 'slow' and as 'fast' peers for the sake of readability. These values are based on realistic access capacities of typical DSL connections with a download bandwidth of 1 Mbit/s and 2 Mbit/s. We use these relatively low access bandwidths to evaluate a default scenario where the total bandwidth demand in the swarm cannot be met by the total upload capacity. Thus, we can observe peers that are not able to play out the complete video, and investigate whether our modifications work under these circumstances.

In contrast to the model in the last chapter, we model the uplink of the peers as the only network bottleneck here and again use a flow-based underlay model to simulate the data transfer of blocks. We make this assumption due to the low upload capacity of the peers in comparison to their download bandwidth and to the video bit rate. Each block transmission constitutes one flow that shares the bottleneck bandwidth fairly with all other flows from the same source.

The shared video is based on a source video containing an episode of a popular TV show and has a length of 22 min. The file has a total size of 55.5 MB and a overall average video bit rate of 336 Kbit/s. It is separated into three layers using the Joint Multi-View Video Model (JMVM) [79], with a Group-of-Picture (GoP) size of 16 embedded frames to achieve temporal scalability. Table 4.1 gives a detailed overview on the characteristics of the individual layers.

We observe that the base layer with index 0 of the scalable video has the highest bit rate. This layer has to be played out by every peer, so that an attained bit rate lower than 229 Kbit/s will lead to stalling. The higher layers have less bandwidth demand, but are still large enough that a streaming mechanism needs to be efficient to distribute them.

The peer arrival process is modeled by a Poisson process with a mean inter-arrival time $E[A]$ of 5 s. The peers are distributed among the two access bandwidth classes according to a pre-defined share, by default half of the peers per class. Peers stay online until they have finished watching the video plus an exponentially distributed serving time S with a default mean value $E[S]$ of 10 min.

Table 4.1: *Movie layer information*

layer index	mean bit rate (Kbit/s)	mean frame rate (fps)	cumulative mean frame rate (fps)	size (MB)
0	229	5.994	5.994	37.8
1	48	5.994	11.988	8.0
2	59	11.988	23.976	9.7

Adding the buffering time of 60 s before starting the playback and the video length of 22 min, we thus get a mean number of roughly 400 concurrently online peers, neglecting stalling times. A swarm of this size is realistic for an average file-sharing swarm [78]. The swarm size is higher than in the previous chapter since the online time of the peers now is governed mainly by the video length and not by the download time of the peers.

Additionally to the peers, we have a number N_{Server} of servers in the network that act like normal peers but have the complete video from the start and are assumed not to go offline during the simulation. A single server has an upload capacity of 512 Kbit/s, allowing us to scale the total server capacity by adjusting the number of servers. A high number of servers reflects the fact that a high capacity is provided by the content provider to support the streaming service. By default, we install 40 servers, leading to a total upload capacity of 20 Mbit/s. We assume that a content provider has to support a number of different video streams, so that this value is realistic for one single stream.

4.5 Performance Evaluation of QoE-Aware Mechanisms

In this section, we present the results from the conducted simulation experiments. Our main performance indicator for the evaluation is the end-user Quality of Experience (QoE) when watching the video. While the download time is the most important performance characteristic in file-sharing, video streaming users notice several different parameters related to the watched video. We evaluate two metrics that can be easily measured and interpreted also in real systems. Full reference metrics used in the related work for live streaming, such as PSNR [29], are not applicable to our scenario, since any available frame, and especially the full base layer, can be always played back without any distortion. Unavailable B-frames are not played back at all.

The first considered metric is the number of layers played out on average. This allows us to see how many layers could be downloaded in time for playout. Since we use stalling for the base layer only, this value is always above 1 and below 3. We first compute the mean value of layers played out for one peer over the video playout time, and then average over all peers in one run. In the figures, we show the mean values and 95% confidence intervals of this value over 10 runs with different seeds.

As a second QoE indicator, we evaluate the total stalling time over the video length. The longer the user has to wait for a forcibly paused video, the lower the quality is. We cumulate all stalling times for one user, and then again compute the mean value over all peers in one run. The average values and 95% confidence intervals are again calculated over 10 runs with different seeds.

We tested the proposed architecture for different network load conditions. Load is again defined as the download demand in comparison to the total upload capacity of the swarm, similar to Section 3.5. However, the demand here is characterized by the bit rate of the video with the highest quality. To generate different load situations, we vary the number of servers and the composition of a peer set with heterogeneous access bandwidths, as well as the serving times

Table 4.2: *Overview on evaluated scenarios*

Scenario	Changed Parameter	Section
Server Capacity	N_{Server}	4.5.1
Peer Heterogeneity	Share of slow peers	4.5.2, 4.5.3
Server Breakdown	Server reduction	4.5.2
Download Strategy	Download after watching	4.5.3
Chunk Selection Strategy	Chunk selection strategy at seeders	4.5.4
Serving Times	$E[S]$	4.5.4

of the peers. Additionally, we test the different download and chunk selection strategies described in Section 4.3, and evaluate the flexibility of the algorithm in a scenario where servers fail spontaneously. An overview on the experiments is given in Table 4.2.

4.5.1 Influence of the Network Load

First, we take a look at the relation between network resources and the achieved QoE. Additionally, we want to investigate how hybrid a P2P VoD system needs to be, i.e., how many servers are needed, to provide a good quality to the end user. To this end, we vary the server upload capacity in the default scenario.

We vary N_{server} from 1 to 80, which directly translates to the upload capacity resources a content provider would offer to support the distribution effort. The resulting QoE indicators for the two peer classes are depicted in Fig. 4.6 and 4.7.

The first observable effect is that a larger number of servers lead to a better quality played out at the peers, i.e., more peers can play out more layers on average. This is expected, since more servers simply mean a higher upload capacity in the network without adding download demand. Thus, the load in the system is reduced. However, this also shows that our adapted chunk selection can make use of the offered additional capacity by adapting the number of enhancement layers downloaded in time for playback. With 60 or more servers, i.e., an accumulated 30 Mbit/s server upload bandwidth, the quality of the video is good or nearly perfect for all peers. In relation to the roughly 90 Mbit/s total bandwidth demand by

the clients, this shows that the overlay can handle a large part of the traffic load. Still, under the specified conditions it cannot provide a good quality all by itself, as the low number of layers for just one server shows.

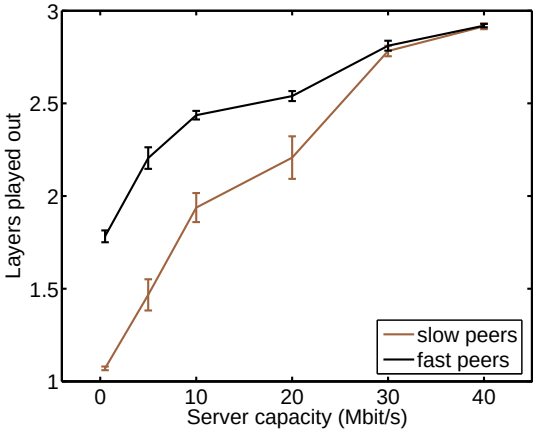


Figure 4.6: *Mean number of layers played out for different number of servers*

Another interesting aspect of the architecture is that the average number of layers played out is higher for the set of fast peers. Also, these peers experience no stalling time, whereas the peers with less upload capacity show a high mean stalling time for a low number of servers. This can be explained with the give-to-get mechanism implemented in Tribler in the unchoking process. This mechanism prefers overlay neighbors with a good upload behavior in the unchoking process of a local peer. Since peers with more capacity can upload more chunks, they get a better give-to-get rating and are therefore preferred in the unchoking process. For a P2P VoD streaming overlay, this indicates peers can actually influence the quality they get by adapting the upload capacity they allocate to the application, in case they do not utilize their full upload capacity. This is a good

incentive for peers to contribute to the overlay.

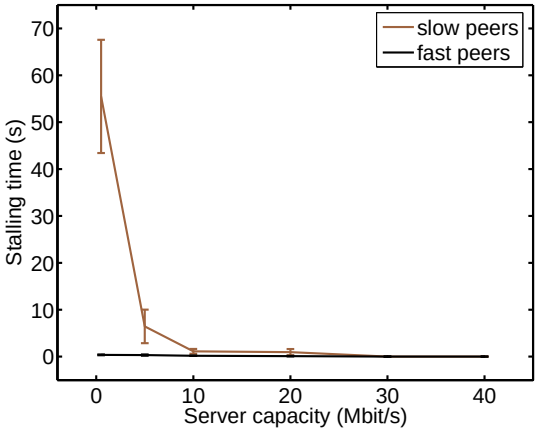


Figure 4.7: *Mean stalling times for different number of servers*

Finally, we observe that the stalling times are negligible for both peer classes for a server capacity of 10 Mbit/s or higher. We conclude that, although the according values for the numbers of layers indicate that the swarm capacity is not high enough to support the video in its highest quality, the base layer is always available. Thus, our strategy prefers the base layer enough to ensure that a peer's download slots are not wasted.

4.5.2 Influence of Peer Heterogeneity and Server Breakdown

Next, we change the composition of the peer set by varying the share of slow peers between 0%, 10%, the default 50%, 90%, and finally 100%, while keeping the total number of peers stable. A lower number of fast peers mean less total upload capacity in the swarm, so that we again influence the load with this

parameter. Additionally, we investigate how the QoE-aware strategies cope with different swarm compositions. To judge how flexible these strategies are without manual parameterization, we let half of the initial 40 servers fail simultaneously after half of the simulation runtime. We show the results for both groups of peers, i.e., peers going online before and after the server breakdown.

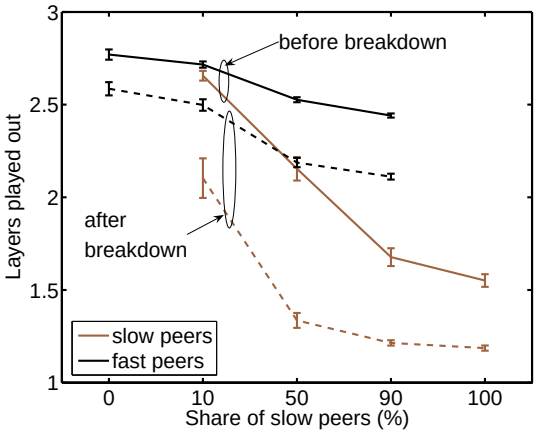


Figure 4.8: *Mean number of layers played out for different peer set compositions, before and after server breakdown*

Figure 4.8 shows again the average number of layers played out for the different peer set compositions. Similar to the previous results, the average quality decreases for both peer classes when less total upload capacity is available in the swarm. Due to the give-to-get rating, however, the fast peers have to sacrifice less quality than the slow peers. Since the fast peers in general are rated higher, they utilize the remaining upload capacity better than the slow peers, which have to bear the consequences of the swarm capacity reduction.

The same effect can be observed for the peers that go online after the reduction of the server capacity. Both, fast and slow peers, can play out less layers on

average. However, the reduction is less for the fast peers than for the slow peers in comparison to the time when all 40 servers are active. This is again due to the higher give-to-get rating of the fast peers, which are therefore preferred in the unchoking process of all peers and the servers. Fast peers give up less quality than slow peers if there are more slow peers in the swarm.

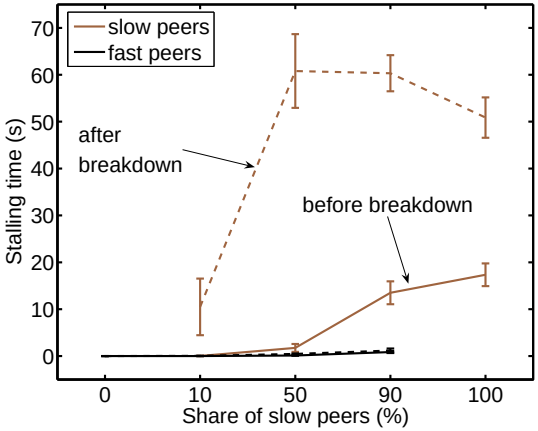


Figure 4.9: *Mean stalling times for different peer set compositions, before and after breakdown*

We again observe the occurrence of stalling just for the peers that are close to the minimum playout quality, cf. Fig. 4.9. Especially the slow peers experience stalling times of up to a minute on average after the server breakdown. However, even with the full server capacity the load in the swarm is high enough for a high fraction of slow peers that significant stalling occurrences can be observed. The fast peers, on the other hand, show no or only very short stalling times both before and after the server breakdown. This coincides with the relatively high quality of on average more than two layers in any case.

In general however, the chunk selection strategy adapted for SVC copes with

the node failures and the according load increase during the swarm lifetime without any parameters needed to be adapted to the new situation. While the load is high enough to lead to stalling, the total stalling times are low enough that the streaming process is not fully interrupted.

4.5.3 Comparison of Download Strategies

The variants of the download strategy we wish to compare in this experiment is the default strategy to stop downloading chunks when the client has finished playing out the movie, and a strategy where the peer continues to download the complete movie in any case. In the latter case a client consumes upload capacity of the swarm even if it no longer profits from it, while other peers might put this capacity to better use since they are still watching the movie. On the other hand, the peer continuing to download may be able to provide more content after it completes the movie file, and thus become a more valuable source for chunks. We compare these strategies again for different compositions of the swarm, similar to the last section.

The results depicted in Fig. 4.10 show that this consideration does not pay off for the default setting of the serving time, i.e., 10 min. If the peers continue to download after they have finished watching the movie, all peers on average play out less layers. Thus, we conclude that the upload capacity consumption of the still downloading peers offsets the gain by having more sources for the complete video file. This is also due to the relatively high load of the swarm, since there is less upload capacity available in total than would be necessary to serve every peer. Therefore, upload bandwidth is more valuable than a good distribution of chunks, since many peers cannot play out all chunks in this scenario. For less loaded swarms, e.g., for swarms where the peers stay online much longer than the movie playout time, this could change.

With respect to the stalling times, the strategy to continue downloading also performs worse than the default strategy, cf. Fig. 4.11. While the fast peers can download the base layer fast enough with both strategies, the slow peers experi-

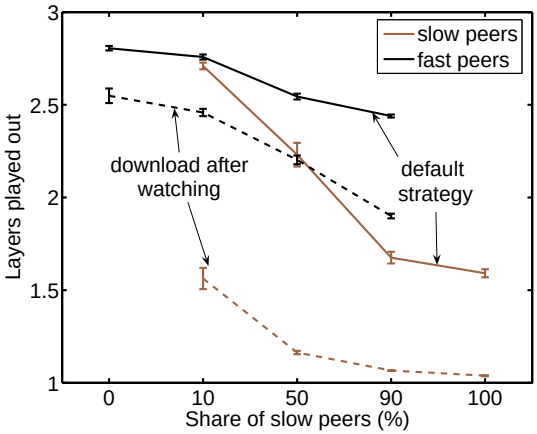


Figure 4.10: *Mean number of layers played out for different download strategies*

ence stalling times of several minutes on average if peers continue to download. In contrast, the default strategy leads to comparably short stalling times even for the slow peers, although these still can only play out the video in a relatively low quality.

4.5.4 Influence of Serving Times and Comparison of Chunk Selection Strategies

In this experiment, we influence the serving time of the peers, i.e., the time peers are online after they have finished watching the video. Similar to the seeding time, a longer serving time leads to a higher available upload capacity and therefore less load. We vary the serving times from 5 to 30 min. Additionally, we also compare variants of the chunk selection strategy with this experiment. For the chunk selection, we have the default implementation that prioritizes chunks from

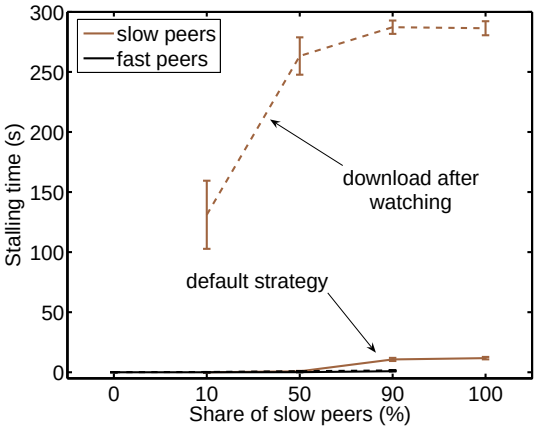


Figure 4.11: *Mean stalling times for different download strategies*

enhancement layers when being unchoked by a seeder. This strategy takes advantage of the fact that seeders are sources for the enhancement layer chunks, which have a lower availability than the base layer chunks under high load conditions. We compare this with a naive implementation that does not discern between seeders and leechers to see whether this mechanism has an effect.

Figure 4.12 shows the mean number of layers played out for both strategies with varying average peer serving times. For both chunk selection mechanisms, we see a direct effect of the higher serving times on the quality of the video. Since peers that stay online longer can upload more data, they reduce the load on the swarm. With a serving time of 30 minutes, all peers can play out the video with nearly maximum quality.

Still, we observe that the chunk selection that prefers enhancement layer chunks when being unchoked by seeders leads to a higher average video quality for both peer groups. The quality gain of this strategy diminishes, however, if

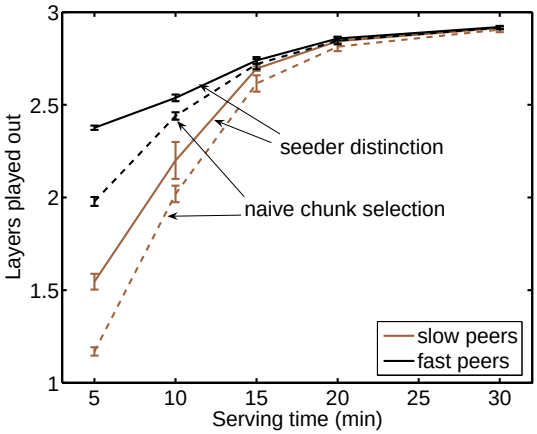


Figure 4.12: *Mean number of layers played out for different chunk selection strategies*

the peers show longer serving times, i.e., when the load in the swarm is decreased. This can be explained by the higher number of layers that can be played out by all peers, and therefore a generally higher availability of enhancement layer chunks.

On the other hand, the stalling times for the slow peers are higher with the more sophisticated chunk selection strategy in the scenario with a serving time of 5 minutes, cf. Fig. 4.13. This can be attributed to the fact that base layer chunks are shared more with the naive chunk selection, and are therefore missing less often when they are required for playback. Thus, in a swarm with a high load, it does not necessarily pay off for all peers to prefer to download enhancement layers, even if the opportunity to do so appears seldom.

Similar to the results in Section 4.5.1, we can see the preference of the base layer due to our strategy here. Stalling occurs only for a serving time of 5 minutes for the slow peers, where the number of layers played out is 1.5 or lower on

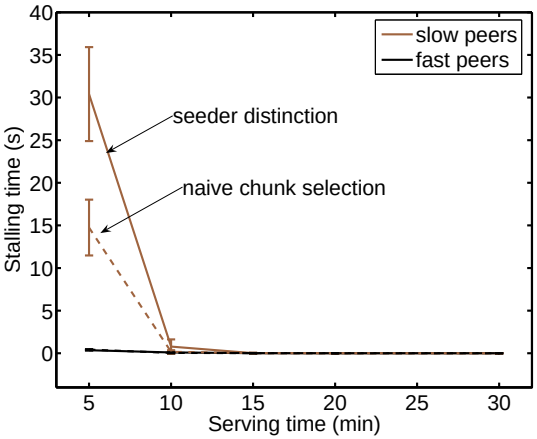


Figure 4.13: *Mean stalling times for different chunk selection strategies*

average. Thus, in the other scenarios the base layer is prioritized enough to ensure that, if enhancement layers are continuously available for playout, the base layer is as well.

4.6 Lessons Learned

From the observed results, we conclude that we can effectively adapt existing overlay mechanisms, in particular the chunk selection, to support scalable videos. The changes do not necessitate completely new overlay logic and rely only on knowledge the peers currently possess, i.e., which chunks are already downloaded and which are not. The evaluated architecture does not need QoE measurements of the video played out or network probes to adapt to changes in the streaming process. The peers still react to changes in the swarm capacity, and should be able to react to varying network conditions as well. Thus, our mecha-

nism is easy to implement in practice and introduces no additional signaling or management overhead.

Due to the give-to-get unchoking mechanism, a basic support for heterogeneity exists. Peers with a higher upload capacity are served with more bandwidth. While the upload capacity of a peer might not always be correlated with the video quality that can be supported by the according end-user device, the principal differentiation between clients with heterogeneous access bandwidths is included in the overlay mechanisms. The new chunk selection, on the other hand, ensures that the bandwidth is used for downloading the video layers in order of usefulness.

The results show that this priorization works well, since peers that can support the streaming of a higher level of quality do not experience stalling. Since stalling for scalable videos can only occur if frames from the base layer are missing, this proves that the base layer is always given the highest priority. Thus, only peers which cannot support the base layer bit rate are forced to pause the video.

We provided and evaluated implementation alternatives for the chunk selection and the download strategy. For the download strategy, we conclude that it is not feasible for peers in a highly loaded swarm to continue downloading the video after the playout process has stopped. As long as upload capacity is scarce, the negative effect of this additional bandwidth consumption outweighs the better chunk availability. Regarding the chunk selection, it pays off to prefer downloading higher layers from seeders in our scenarios. Thus, the relatively high availability of the base layer is countered for peers that can support a higher level of quality. Still, this strategy can backfire if the load in the swarm is high enough so that peers can only sustain the streaming of the base layer. The swarm situation as a whole therefore influences how well these strategies perform.

Regarding the feasibility of an overlay to support streaming, our experiments show that there is a need for a fixed server capacity in order to support an acceptable video quality for the users. The overlay alone is not able to provide the necessary upload capacity under the evaluated load conditions. This is in line with results from literature. However, the peers contribute a significant amount of capacity resources that can therefore be saved by the streaming service provider.

Therefore, using an overlay with server support is a cost-effective alternative to pure client-server streaming architectures. Moreover, since the servers can participate in the overlay with the same behavior as peers, it is simple to add to remove server capacity during the lifetime of a swarm. In comparable client-server systems, removing a server might result in a connection loss and therefore in a stream loss for the users connected to that server.

5 Conclusion

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.
Winston Churchill (1874 - 1965)

In today's Internet, building overlay structures to provide a service is becoming more and more common. This approach allows for the utilization of client resources, thus being more scalable than a client-server model in this respect [96]. However, in these architectures the quality of the provided service depends on the clients and is therefore more complex to manage. Resource utilization, both at the clients themselves and in the underlying network, determine the efficiency of the overlay application. Here, a trade-off exists between the resource providers and the end users that can be tuned via overlay mechanisms. Thus, resource management and traffic management is always quality-of-service management as well.

In this monograph, the three currently significant and most widely used overlay types in the Internet were considered. These overlays are implemented in popular applications which only recently have gained importance. Thus, these overlay networks still face real-world technical challenges which are of high practical relevance. We identified the specific issues for each of the considered overlays, and showed how their optimization affects the trade-offs between resource efficiency and service quality. Thus, we supplied new insights and system knowledge that is not provided by previous work.

For search overlays and specifically Distributed Hash Tables (DHTs), the current state of research and the evidence in form of actually implemented systems show that one-hop structures are the most efficient solution for time-critical ap-

plications. While best-effort services may use less interconnected overlays, a full mesh is viable and necessary in enterprise environments.

However, these evaluations focus on overlay traffic as the prime performance indicator, and neglect the processing and storage load on the nodes themselves. To remedy this, we presented an analysis based on a representative self-developed architecture. In contrast to the existing work, we evaluated the query load on the nodes. We showed that the internal query load is significantly higher than the externally seen load, up to a factor of 4 in the considered system. Additionally, we proved that storage space on the participating nodes can be traded for shorter search times. This trade-off can be influenced by the system size and the redundancy of stored data. Thus, such a system can be dimensioned and configured using our analytical model.

The main issue with the second class of overlays, P2P file-sharing systems, is the amount of generated inter-domain traffic. Currently, many research projects and industry initiatives work on solutions for that problem. The Application Layer Traffic Optimization (ALTO) IETF working group is in the act of creating a standard for the implementation of these solutions. The most promising approach in this context is locality-awareness. We added our own solution of Biased Unchoking to the existing Biased Neighbor Selection mechanism, targeting a different overlay mechanism. We showed that the two algorithms complement each other and that their combination outperforms each of the individual solutions alone in terms of traffic savings. Moreover, we observed the proposed mechanisms in realistic scenarios. We used input from measurement studies to model swarms that show a similar heterogeneity as live swarms in their peer distribution and their access bandwidths.

With this setup, we revealed previously unknown drawbacks of existing locality-awareness solutions, namely the introduction of unfairness in the download speed of the peers. In these scenarios, peers in small ASes take significantly longer (up to 20 percent) to download a file with the combination of both locality-aware schemes. We showed that this can be regulated by using the parameters offered by both Biased Neighbor Selection and Biased Unchoking. Still, our con-

clusion is that locality-promotion mechanisms in their current form are not beneficial for all clients, and are therefore unlikely to be accepted by the end users.

Finally, the overlays offering a video streaming service were considered. The developing problem of heterogeneous end user devices with different access speeds has been addressed only recently with the advent of efficient scalable video codecs. These have found their way into live-streaming overlays, but not into the VoD streaming class considered here.

Therefore, we developed adaptations to an existing VoD architecture, Tribler, that enable the usage of scalable video codecs. We evaluated this system in a simulation study, and showed that different types of clients can retrieve the video according to their access capabilities. A base video quality can be provided in most scenarios without any occurrences of stalling. The remaining capacity is utilized to provide a higher quality according to a peer's capacity. Thus, the aim of supporting heterogeneous clients with the same overlay is reached. Additionally, the peers adapt to the capacity available in the swarm by automatically changing the quality of the video played out. Thus, our architecture does not have to rely on measurements as configuration input, which would add additional overhead. Since such an architecture is feasible, it is most likely that content providers will harness the benefits of P2P video streaming, even if they have a highly heterogeneous set of clients.

The aforementioned contributions comprise an evaluation of current overlay challenges and the optimization potential of P2P overlay technologies. On the basis of this work, future evaluations of the considered overlays may be conducted. For file-sharing networks, the current locality-promotion schemes may be improved to introduce more fairness in real swarms. This would ensure that such a traffic management scheme will be accepted by the users as well as by the ISPs. As a consequence, Economic Traffic Management would be implemented, where all involved parties have an incentive to participate. In the case of VoD streaming overlays, the same holds true, while additionally, the work on the streaming of scalable videos could be expanded to include more than one type of scalability.

Bibliography of the Author

— Journals and Book Chapters —

[1] T. Hoßfeld, D. Hausheer, F. Hecht, F. Lehrieder, S. Oechsner, I. Papafili, P. Racz, S. Soursos, D. Staehle, G. D. Stamoulis, P. Tran-Gia, and B. Stiller, *FIA Prague Book, ISBN 978-1-60750-007-0*, ch. An Economic Traffic Management Approach to Enable the TripleWin for Users, ISPs, and Overlay Providers, . Towards the Future Internet - A European Research Perspective: IOS Press Books Online, 2009.

— Conference Papers —

[2] M. Menth, J. Milbrandt, and S. Oechsner, "Experience Based Admission Control (EBAC)", in *The Ninth IEEE Symposium on Computers and Communications (ISCC2004)*, Alexandria, Egypt, 2004.

[3] J. Milbrandt, M. Menth, and S. Oechsner, "EBAC - A Simple Admission Control Mechanism", in *ICNP 2004, 12th IEEE International Conference on Network Protocols*, Berlin, Germany, 2004.

[4] S. Oechsner and O. Rose, "A filtered beam search approach to scheduling cluster tools in semiconductor manufacturing", in *Industrial Engineering Research Conference*, Atlanta, U.S.A, 2005.

[5] S. Oechsner and O. Rose, "Scheduling Cluster Tools Using Filtered Beam

Search and Recipe Comparison", in *Winter Simulation Conference*, Orlando, U.S.A., 2005.

[6] F.-U. Andersen, K. Tutschku, T. Hoßfeld, and S. Oechsner, "Verlässliche Peer-to-Peer Technologie als Steuerungsverfahren für zukünftige mobile Zugangsnetze", in *11. ITG-Mobilfunktagung, Technologien und Anwendungen, ISBN 3-8007-2942-3*, Osnabrück, Germany, 2006.

[7] T. Hoßfeld, S. Oechsner, K. Tutschku, and F.-U. Andersen, "Evaluation of a Pastry-based P2P Overlay for Supporting Vertical Handover", in *IEEE Wireless Communications and Networking Conference*, Las Vegas, U.S.A., 2006.

[8] T. Hoßfeld, S. Oechsner, K. Tutschku, F.-U. Andersen, and L. Caviglione, "Supporting Vertical Handover by Using a Pastry Peer-to-Peer Overlay Network", in *Mobile Peer-to-Peer Computing MP2P'06, in conjunction with the 4th IEEE International Conference on Pervasive Computing and Communications (PerCom'06)*, Pisa, Italy, 2006.

[9] S. Oechsner, T. Hoßfeld, K. Tutschku, and F.-U. Andersen, "Supporting Vertical Handover by a Self-Organizing Multi-Dimensional P2P Overlay", in *2006 IEEE 63rd Vehicular Technology Conference*, Melbourne, Australia, 2006.

[10] S. Oechsner, T. Hoßfeld, K. Tutschku, F.-U. Andersen, and L. Caviglione, "Using Kademlia for the Configuration of B3G Radio Access Nodes", in *Mobile Peer-to-Peer Computing MP2P'06, in conjunction with the 4th IEEE International Conference on Pervasive Computing and Communications (PerCom'06)*, Pisa, Italy, 2006.

[11] S. Oechsner and P. Tran-Gia, "Performance Evaluation of a Reliable Content Mediation Platform in the Emerging Future Internet", in *20th International Teletraffic Congress (ITC20)*, Ottawa, Canada, 2007.

[12] S. Oechsner, S. Soursos, I. Papafili, T. Hoßfeld, G. D. Stamoulis, B. Stiller, M. A. Callejo, and D. Staehle, “A framework of economic traffic management employing self-organization overlay mechanisms”, in *3rd International Workshop on Self-Organizing Systems IWSOS’08*, Vienna, Austria, 2008.

[13] T. Zinner, S. Oechsner, T. Hoßfeld, and P. Tran-Gia, “On the Trade-off between Efficiency and Congestion in Location-aware Overlay Networks - Example of a Vertical Handover Support System”, in *8th International Conference on Peer-to-Peer Computing (P2P 2008)*, Aachen, Germany, 2008.

[14] S. Oechsner, T. Hoßfeld, and P. Tran-Gia, “Performance Evaluation of a Distributed Lookup System for a Virtual Database Server”, in *20th ITC Specialist Seminar*, Hoi An, Vietnam, 2009.

[15] S. Oechsner, F. Lehrieder, T. Hoßfeld, F. Metzger, K. Pussep, and D. Staehle, “Pushing the Performance of Biased Neighbor Selection through Biased Unchoking”, in *9th International Conference on Peer-to-Peer Computing (P2P 2009)*, Seattle, U.S.A., 2009.

[16] R. Pries, D. Staehle, S. Oechsner, M. Menth, S. Menth, and P. Tran-Gia, “On the Unfair Channel Access Phenomenon in Wireless LANs”, in *21st International Teletraffic Congress (ITC 21)*, Paris, France, 2009.

[17] K. Pussep, S. Oechsner, O. Abboud, M. Kantor, and B. Stiller, “Impact of Self-Organization in P2P Overlays on Underlay Utilization”, in *The Fourth International Conference on Internet and Web Applications and Services*, Venice, Italy, 2009.

[18] S. Oechsner, T. Zinner, J. Prokopetz, and T. Hoßfeld, “Supporting Scalable Video Codecs in a P2P Video-on-Demand Streaming System,”, in *21th ITC Specialist Seminar on Multimedia Applications - Traffic, Performance and QoE*, Miyazaki, Japan, 2010.

General References

[19] L. Takács, “A single server queue with Poisson input”, *Operations Research*, Vol. 10, No. 3, 1962.

[20] P. Mockapetris, “Domain Names - Concepts and Facilities.” RFC 1034, 1987.

[21] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, “Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web”, in *STOC ’97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, New York, NY, USA, ACM, 1997.

[22] G. J. Conklin, G. S. Greenbaum, K. O. Lillevold, A. F. Lippman, and Y. A. Reznik, “Video Coding for Streaming Media Delivery on the Internet”, in *DCC ’01: Proceedings of the Data Compression Conference*, Washington, DC, USA, IEEE Computer Society, 2001.

[23] D. Eastlake 3rd and P. Jones, “US Secure Hash Algorithm 1 (SHA1).” RFC 3174 (Informational), 2001. Updated by RFC 4634.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, “A Scalable Content-Addressable Network”, in *SIGCOMM ’01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Vol. 31, New York, NY, USA, ACM, 2001.

[25] A. Rowstron and P. Druschel, “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems”, in *Middleware ’01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Springer-Verlag, 2001.

[26] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, in *SIGCOMM ’01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, USA, ACM, 2001.

[27] R. Cox, A. Muthitacharoen, and R. Morris, “Serving DNS Using a Peer-to-Peer Lookup Service”, in *IPTPS ’02: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, Cambridge, MA, USA, Springer Berlin/Heidelberg, 2002.

[28] T. J. Giuli and M. Baker, “Narses: A Scalable Flow-Based Network Simulator”, *Computing Research Repository (CoRR)*, Vol. cs.PF/0211024, 2002.

[29] L. Lu, Z. Wang, A. C. Bovik, and J. Kouloheris, “Full-Reference Video Quality Assessment Considering Structural Distortion and No-Reference Quality Evaluation of MPEG Video”, in *in Proc. IEEE Int. Conf. Multimedia*, 2002.

[30] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, “Search and Replication in Unstructured Peer-to-Peer Networks”, in *ICS ’02: Proceedings of the 16th international conference on Supercomputing*, New York, NY, USA, ACM, 2002.

[31] P. Maymounkov and D. Mazieres, “Kademlia: A Peer-to-peer Information System Based on the XOR Metric”, in *IPTPS ’02: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, Cambridge, MA, USA, Springer Berlin/Heidelberg, 2002.

[32] D. Tsoumakos and N. Roussopoulos, “A Comparison of Peer-to-Peer Search Methods”, in *Proceedings of the Sixth WebDB Workshop*, San Diego, California, United States, 2003.

[33] A. Binzenhöfer and P. Tran-Gia, “Delay Analysis of a Chord-based Peer-to-Peer File-Sharing System”, in *ATNAC 2004*, Sydney, Australia, 2004.

[34] A. Gupta, B. Liskov, and R. Rodrigues, “Efficient Routing for Peer-to-Peer Overlays”, in *NSDI’04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, Berkeley, CA, USA, USENIX Association, 2004.

[35] O. Heckmann, A. Bock, A. Mauthe, and R. Steinmetz, “The eDonkey File-Sharing Network”, in *Informatik 2004, Informatik verbindet* (M. R. Peter Dadam, ed.), Vol. 2 of *Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, Ulm: GI, Gesellschaft für Informatik, Bonn, 2004.

[36] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, Al, and L. Garcés-Erice, “Dissecting BitTorrent: Five Months in a Torrent’s Lifetime”, in *Passive and Active Network Measurement*, Vol. 3015/2004 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2004.

[37] R. Rodrigues and C. Blake, “When Multi-Hop Peer-to-Peer Lookup Matters”, in *Revised Papers from the Third International Workshop on Peer-to-Peer Systems, IPTPS 2004* (S. B. . Heidelberg, ed.), Vol. 3279/2005 of *Lecture Notes in Computer Science*, La Jolla, CA, USA, Springer Berlin / Heidelberg, 2004.

[38] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, “Tapestry: a Resilient Global-scale Overlay for Service Deployment”, *Selected Areas in Communications, IEEE Journal on*, Vol. 22, No. 1, 2004.

[39] International Telecommunication Union, “H.264 : Advanced video coding for generic audiovisual services.” ITU-T Recommendation H.264, 2005.

[40] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, “Should Internet Service Providers Fear Peer-Assisted Content Distribution?”, in *IMC ’05: Pro-*

ceedings of the 5th ACM SIGCOMM conference on Internet Measurement, Berkeley, CA, USA, USENIX Association, 2005.

[41] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes", *Communications Surveys & Tutorials, IEEE*, 2005.

[42] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The Bittorrent P2P File-Sharing System: Measurements and Analysis", in *Peer-to-Peer Systems IV*, Springer Berlin / Heidelberg, 2005.

[43] X. Zhang, J. Liu, B. Li, and Y. S. P. Yum, "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming", *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, Vol. 3, 2005.

[44] S. Annapureddy, C. Gkantsidis, and P. Rodriguez, "Providing Video-on-Demand using Peer-to-Peer Networks", in *Internet Protocol TeleVision (IPTV) workshop in conjunction with WWW '06*, 2006.

[45] S. A. Baset and H. G. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol", in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006.

[46] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving Traffic Locality in BitTorrent via Biased Neighbor Selection", in *26th IEEE International Conference on Distributed Computing Systems, 2006. ICDCS 2006*, 2006.

[47] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data", in *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USA, USENIX Association, 2006.

[48] C. Dana, D. Li, D. Harrison, and C. N. Chuah, "BASS: BitTorrent Assisted Streaming System for Video-on-Demand", in *IEEE 7th Workshop on Multimedia Signal Processing, 2005*, 2006.

[49] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System", in *In Proc. of IPTV Workshop, International World Wide Web Conference*, 2006.

[50] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest First and Choke Algorithms Are Enough", in *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, New York, NY, USA, ACM, 2006.

[51] F. L. Piccolo, G. Bianchi, and S. Cassella, "Efficient Simulation of Bandwidth Allocation Dynamics in P2P Networks", in *Proceedings of the Global Telecommunications Conference, 2006. GLOBECOM '06, San Francisco, CA, USA*, 2006.

[52] J. Risson, A. Harwood, and T. Moors, "Stable High-Capacity One-Hop Distributed Hash Tables", in *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications*, Washington, DC, USA, IEEE Computer Society, 2006.

[53] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "BiToS: Enhancing BitTorrent for Supporting Streaming Applications", in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*, 2006.

[54] V. Aggarwal, A. Feldmann, and C. Scheideler, "Can ISPs and P2P systems co-operate for improved performance?", *ACM SIGCOMM Computer Communications Review (CCR)*, Vol. 37, No. 3, 2007.

[55] P. Baccichet, T. Schierl, T. Wieg, and B. Girod, "Low-delay Peer-to-Peer Streaming using Scalable Video Coding", in *Proc. International Packet Video Workshop - PV2007*, Lausanne, Switzerland, 2007.

[56] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai, “Improving VoD Server Efficiency with BitTorrent”, in *MULTIMEDIA ’07: Proceedings of the 15th international conference on Multimedia*, New York, NY, USA, ACM, 2007.

[57] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s Highly Available Key-Value Store”, *SIGOPS Oper. Syst. Rev.*, Vol. 41, No. 6, 2007.

[58] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, “Characterizing residential broadband networks”, in *IMC ’07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, New York, NY, USA, ACM, 2007.

[59] P. Eckersley, F. von Lohmann, and S. Schoen, “Packet Forgery By ISPs: A Report On The Comcast Affair.” https://www.eff.org/files/eff_comcast_report.pdf, 2007.

[60] J. F. Gantz, D. Reinsel, C. Chute, W. Schlichting, J. McArthur, S. Minton, I. Xheneti, A. Toncheva, and A. Manfrediz, “The Expanding Digital Universe.” White paper, 2007.

[61] D. Jurca, J. Chakareski, J.-P. Wagner, and P. Frossard, “Enabling Adaptive Video Streaming in P2P Systems”, *IEEE Communications Magazine*, Vol. 45, 2007.

[62] H. Schwarz, D. Marpe, and T. Wiegand, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard”, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 17, No. 9, 2007.

[63] V. Aggarwal and A. Feldmann, “Locality-Aware P2P Query Search with ISP Collaboration”, *Networks and Heterogeneous Media*, Vol. 3, No. 2, 2008.

[64] T. Bocek, W. Kun, F. V. Hecht, D. Hausheer, and B. Stiller, "PSH: A Private and Shared History-Based Incentive Mechanism", in *AIMS '08: Proceedings of the 2nd international conference on Autonomous Infrastructure, Management and Security*, Berlin, Heidelberg, Springer-Verlag, 2008.

[65] D. R. Choffnes and F. E. Bustamante, "Taming the Torrent: A Practical Approach to Reducing Cross-ISP Traffic in Peer-to-Peer Systems", *SIGCOMM Comput. Commun. Rev.*, Vol. 38, No. 4, 2008.

[66] B. Cohen, "The BitTorrent Protocol Specification." http://www.bittorrent.org/beps/bep_0003.html, 2008.

[67] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "PNUTS: Yahoo!'s hosted data serving platform", *Proc. VLDB Endow.*, Vol. 1, No. 2, 2008.

[68] J. P. F.-P. Gimenez, M. A. C. Rodriguez, H. Hasan, T. Hoßfeld, D. Staehle, Z. Despotovic, W. Kellerer, K. Pussep, I. Papafili, G. D. Stamoulis, and B. Stiller, "A New Approach for Managing Traffic of Overlay Applications of the SmoothIT Project", in *2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS 2008)*, Bremen, Germany, 2008.

[69] C. Kim, M. Caesar, and J. Rexford, "Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises", in *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, New York, NY, USA, ACM, 2008.

[70] S. Le Blond, A. Legout, and W. Dabbous, "Pushing BitTorrent Locality to the Limit." Tech. Rep. inria-00343822, 2008.

[71] Y. Liu, Y. Guo, and C. Liang, "A Survey on Peer-to-Peer Video Streaming Systems", *Peer-to-Peer Networking and Applications*, Vol. 1, No. 1, 2008.

[72] J. Mol, J. Pouwelse, M. Meulpolder, D. Epema, and H. Sips, “Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems”, in *Multimedia Computing and Networking 2008*, Vol. 6818, SPIE Vol. 6818, 2008.

[73] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips, “Tribler: A social-based peer-to-peer system”, *Concurrency and Computation: Practice and Experience*, Vol. 20, 2008.

[74] H. Xie, R. Y. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, “P4P: Provider Portal for Applications”, *SIGCOMM Comput. Commun. Rev.*, Vol. 38, No. 4, 2008.

[75] I. Cisco Systems, “Cisco Visual Networking Index: Forecast and Methodology, 2008-2013.” White Paper, 2009.

[76] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer, “ProtoPeer: a P2P toolkit bridging the gap between simulation and live deployement”, in *Simutools ’09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, Belgium, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[77] Hendrik Schulze and Klaus Mochalski, “Internet Study 2008/2009.” http://www.ipoque.com/resources/internet-studies/, 2009.

[78] T. Hoßfeld, D. Hock, S. Oechsner, F. Lehrieder, Z. Despotovic, W. Kellerer, and M. Michel, “Measurement of BitTorrent Swarms and their AS Topologies”, Tech. Rep. 463, University of Würzburg, 2009.

[79] Joint Video Team (JVT) , “JMVM (Joint Multiview Video Model) software for the Multiview Video Coding (MVC) project of the Joint Video Team (JVT) of the ISO/IEC Moving Pictures Experts Group (MPEG) and the ITU-T Video Coding Experts Group (VCEG)”, 2009.

[80] B. Liu, Y. Cui, Y. Lu, and Y. Xue, "Locality-Awareness in BitTorrent-like P2P Applications", *IEEE Transactions on Multimedia*, Vol. 11, 2009.

[81] Y. Lu, B. Fallica, F. A. Kuipers, R. E. Kooij, and P. V. Mieghem, "Assessing the Quality of Experience of SopCast", *Int. J. Internet Protoc. Technol.*, Vol. 4, No. 1, 2009.

[82] M. Piatek, H. Madhyastha, J. John, A. Krishnamurthy, and T. Anderson, "Pitfalls for ISP-friendly P2P design", in *HotNets 2009*, 2009.

[83] H. Wang, J. Liu, and X. Ke, "On the Locality of BitTorrent-based Video File Swarming", in *Proc. of the 8th International Workshop on Peer-to-Peer Systems (IPTPS'09)*, 2009.

[84] "BitTorrent User Manual." http://www.bittorrent.com/btusers/guides/bittorrent-user-manual, 2010.

[85] 3rd Generation Partnership Project, "Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Stage 2 (Release 10); 3GPP TS 23.228 V10.0.0", 2010.

[86] Blizzard Entertainment, Inc., "Blizzard Downloader F.A.Q.." http://www.worldofwarcraft.com/info/faq/blizzarddownloader.html, 2010.

[87] PPLive Inc., "PPLive." http://www.pptv.com/en/, 2010.

[88] Skype Limited, "Skype." www.skype.com, 2010.

[89] SopCast.com, "SopCast." http://www.sopcast.com/, 2010.

[90] Theory.org Wiki, "Theory.org BitTorrent Specification." http://wiki.theory.org/BitTorrentSpecification, 2010.

[91] Vuze, Inc., "Vuze - Technology." http://www.vuze.com/corp/technology.php, 2010.

[92] YouTube, LLC, “YouTube Home Page.” http://www.youtube.com/, 2010.

[93] T. Zinner, O. Abboud, O. Hohlfeld, T. Hoßfeld, and P. Tran-Gia, “Towards QoE Management for Scalable Video Streaming,”, in *21th ITC Specialist Seminar on Multimedia Applications - Traffic, Performance and QoE*, Miyazaki, Japan, 2010.

[94] T. Zinner, T. Hoßfeld, T. N. Minash, and M. Fiedler, “Controlled vs. Uncontrolled Degradations of QoE - The Provisioning-Delivery Hysteresis in Case of Video”, in *New Dimensions in the Assessment and Support of Quality of Experience (QoE) for Multimedia Applications*, Tampere, 2010.

[95] D. Bertsekas and R. Gallagher, *Data Networks*. Prentice Hall, 1987.

[96] R. Steinmetz and K. Wehrle, *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[97] H. Takagi, *Queueing Analysis: A Foundation of Performance Evaluation*, Vol. 1. Amsterdam, Netherlands: North-Holland, 1991.

ISSN 1432-8801