Update Propagation for Peer-to-Peer-based Massively Multi-user Virtual Environments

Inauguraldissertation
zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften der Universität Mannheim
vorgelegt von

Dipl.-Inform. Richard Süselbeck

Dekan: Dr. Jürgen M. Schneider
Erstreferent: Prof. Dr. Christian Becker
Zweitreferent: Prof. Dr. Alexander Mädche

Tag der mündlichen Prüfung: 10.06.2013

Für meine Großeltern

Acknowledgements

Over the years I have been repeatedly asked for advice on whether pursuing a PhD is good idea. I have always replied: Yes, but only if you find a great advisor.

It is my great fortune to have had such an advisor in Prof. Dr. Christian Becker. Thank you, Christian. Thank you for the wine lessons, thank your for the midnight pep talks and Mahalo for letting me get away from it all now and then. Above all, thank you for your trust and support.

A successful thesis doesn't just need a great advisor, it also needs a great postdoc. Once again I am fortunate to have had such a postdoc in Dr. Gregor Schiele. Thank you Gregor, without your advice and friendship this thesis simply would not have been possible.

I am also deeply indebted to all my colleagues at Wifo II for their support over the last few years. Special thanks go to Dr. Verena. Your cheerful nature made sharing an office with you an absolute joy. The English Theatre will always be legendary! Thank you also to Dr. Florian. Your support in the last few hectic weeks of my thesis was vital to its success. Speaking of support, thank you, Kerstin and Markus for all your help and for putting up with my not-so-occasional chaos.

I also want to express my gratitude to my fair and constructive thesis committee, Prof. Dr. Alexander Mädche, Prof. Dr. Paul Gans and Prof. Dr. Heiner Stuckenschmidt. A very special thanks goes to Dr. Ingo Bayer for making my graduation date possible.

Thank you also to all my friends for their support, especially Dr. Daniela for keeping me sane when the going got tough and Dr. Martin, the right man to have at your side when someone points a gun at you. Just don't ask him for directions.

Another huge thank you is due to my entire family, especially my parents, my grandparents and my brother Leonhard, whose encouragement and patience has always been unwavering.

Coming to Mannheim was the right decision for many reasons, most of all because it's where found my amazing and wonderful girlfriend Dr. Johanna. To the one who makes it all worthwhile: I love you.

Contents

Chapter 1

Introduction

Over the last decade Massively Multi-user Virtual Environments (MMVEs) have become an integral part of modern culture and business. Applications for these large-scale virtual environments range from gaming to business and scientific research. Some MMVEs reach a user base in the tens of millions and the total number of registered users has quadrupled since 2009 to an estimated 1.7 billion [1]. The market for gaming MMVEs alone is expected to reach 22 billion in 2015 [2].

Despite this success, launching an MMVEs is still a risky proposition. This is in large part due to the high cost associated with setting up and maintaining the necessary server infrastructure. One way of reducing the costs of operating MMVEs is to switch their system architecture from the current client/server-based model to one based on peer-to-peer (P2P) technologies. This has the potential to significantly reduce the infrastructure costs of MMVEs, as users bring their own resources into the P2P system and servers are no longer required, thus decreasing expenses and market entry barriers.

This thesis describes a scalable and low-latency update propagation system for P2P-based MMVEs. Update propagation refers to the exchange of information about changes in the virtual environment between users and is one of the key components of MMVEs. Thus, our system represents a key step towards operating MMVEs as fully distributed peer-to-peer systems.

Section 1.1 of this introduction provides a definition and overview of MMVEs in general. We then give a brief review of MMVE business models in Section 1.2. Section 1.3 introduces the standard client/server architecture currently used for MMVEs. In Section 1.4 we motivate the use of a P2P architecture for MMVEs and then show that

update propagation is a key component of any such system in Section 1.5. Finally, we summarize the contributions of this thesis in Section 1.6 and provide an overview of its structure in 1.7.

Figure 1.1: Screenshot of Guild Wars 2[3] - This screenshot illustrates a 3D representation of a virtual world as shown to the user. In the center is the Avatar used to interact with the world.

1.1 Massively Multi-user Virtual Environments

Application areas for MMVEs include social networks [4], collaboration [5], education [6], business [7] and research [8]. Their most successful applications however, remain Massively Multi-Player Online Games (MMOGs), such as Habbo Hotel [9] with an estimated 250 million registered users [1] and World of Warcraft [10] with over 10 million active subscribers [11]. Figure 1.1 shows a screenshot of the MMOG Guild Wars 2 [3].

Despite the importance of these virtual worlds there is currently no consensus definition for the term MMVE. For the purpose of this thesis, we present our own definition, based in part on the discussion of the subject by Bartle in [12].

Definition 1:

MMVEs are shared computer-based simulated environments in which large number of users interact with each other in real-time. The environment is shown to the user as a 3D or 2D graphical representation. Users perform their interactions using virtual representations of the themselves (called Avatars) in the virtual environment.

In this context, *large numbers of users* refers to the fact that an MMVE can theoretically be shared by an unlimited number of users. This sets them apart from other virtual worlds such as conferencing systems or certain types of multi-player games, which have an upper limit on the number of active users. This upper limit is usually less than 100 users. While the number of users in an MMVE is theoretically unlimited, there are some practical upper limits. These are based on the limitations of content generation and technology.

The limits of content generation derive from the fact that the virtual world itself must be large enough to provide its services and features to all its users. As an example, a virtual town built for a few hundred users cannot practically hold a few thousand of them. This means that designing and producing the content of the virtual world can be a significant bottleneck. In addition, due to the technological limitations of current MMVE architectures, any given virtual environment, no matter how large, can only support a certain amount of concurrent users before the system becomes overloaded.

Even with these limitations in place, MMVEs routinely reach concurrent user numbers in the tens of thousands. In fact, the current record for concurrent users in a single virtual world is 60453 users, as set by Eve Online in late 2010 [13]. It should be noted that many MMVEs have overall user numbers far exceeding this record. The operators deal with this by setting up a number of parallel and separate virtual worlds, sometimes called *shards* or *servers*.

In our definition, the phrase *interaction in real-time* refers to the fact that MMVEs are highly interactive systems. Users expect the system to react quickly to their inputs. Similarly they expect the other users to be informed quickly about their inputs. This means that MMVEs tolerate only small delays between system input and output.

The exact delay that can be tolerated by users is application dependent, i.e. players of a fast-paced action game have lower tolerance for delay than participants of a virtual

world designed purely for social interaction. However research has shown that the tolerated delay can be as low as 60ms [14].

Finally, note that the terms virtual environment and virtual world can be used interchangeably.

1.2 MMVE Business Models

MMVEs are supported by a variety of business models, from charging a monthly subscription fee to selling virtual goods. In the following we will give a brief overview of these models.

The earliest MMVEs charged a time-based usage fee, usually an hourly charge. This model was pioneered by titles like Habitat [15] and Neverwinter Nights [16]. These charged several dollars per hour of play in addition to the charges of the online service they were running on [17, 18]. As the charges for internet access changed from a usage-based model to monthly flat fees, this model fell out of favour for MMVEs as well and is no longer used.

It was generally replaced by a subscription model, where users are charged a monthly fee for access to the virtual world. This model is sometimes referred to as *pay-to-play* [19]. Early examples of this include Meridian 59 [20] and Ultima Online [21]. The most successful subscription-based MMVE remains World of Warcraft [10], with over 10 million active subscribers [11].

While subscription-based MMVEs are still being launched (e.g. [22]), it has become a more common approach to let users access the virtual world for free and gain revenue by charging for virtual goods or additional features. This model is sometimes called *free-to-play* [19], despite the fact that many users spend significant amounts of money while participating in the virtual world. The extent to which the MMVEs are actually available to play for free varies. Some titles merely incentivize purchases, while others eventually require them to allow continued participation or progression in the virtual world. This model has become very successful over the last few years, with several MMVEs switching from a subscription-based model to a Free-to-Play one [23] [24] [25], even increasing revenue and user numbers in the process [26] [27].

Selling virtual goods and virtual currency is one of the main revenue streams for many free-to-play virtual worlds (e.g. [28, 29]). These virtual goods can include any-

thing from small temporary improvements of player abilities (e.g. health potions in Lord of the Rings Online [23] for a few cents each) to rent for a piece of virtual real estate in Second Life at several hundred Euro per month [30]. In this context small payments in the range of several cents to several euros are often called *micro-payments*, while larger transactions are often referred to as *macro-payments* .

Users of virtual worlds can also create and sell virtual goods to other players [31]. In some virtual worlds, the virtual currency can be converted back to real world currency [30]. This has already become a significant source of income for some individuals and companies [32]. The sale of virtual goods provides another revenue stream for MMVE operators, if they take a cut from every currency conversion or virtual trade.

It should be noted that many MMVEs use a combination of these various revenue streams. Many free-to-play MMVEs also offer optional subscriptions which provide a variety of benefits not available to non-subscribing users (e.g. [33]). This type of hybrid model is often referred to as *freemium*. Similarly, subscription-based MMVE will often provide some features and services for an additional fee (e.g. [10]).

It is also common to charge a one-time upfront access fee in addition to other charges, particularly for subscription-based MMOGs [22][3]. This fee is usually in the range of 40-60 Euro. Additional revenue streams, such as advertising or selling real-world merchandise related to the virtual world are also regularly used.

As business models evolve, the MMVE market continues to grow: the number of registered MMVE accounts has quadrupled since 2009 [1].

1.3 Client/Server Architecture

All commercially deployed MMVEs are currently based on a client/server (CS) architecture. In this model, the entire virtual world is stored and run on a central server cluster. Users connect to this server cluster using client software. This client software sends updates about the users actions to the server cluster, which then determines the virtual environment's reactions to them. Any changes to the environment are then sent back to all relevant clients, which display them for the user.

The key problem of this architecture is that it requires a massive investment in the installation and operation of the server infrastructure. The initial setup of the infrastructure for an MMVE can be in the millions of euros [34]. Once the system is

running, it incurs the typical costs of data centres, such as for bandwidth, electricity, cooling, maintenance personnel and replacement hardware.

Setting up the necessary server infrastructure also results in an additional problem for the launch of new MMVEs. Before the launch of a new virtual world, it is often unclear how successful it will become and thus how many users its infrastructure must be able to support. Depending on the actual user numbers, the operator may have either over- or underprovisioned his infrastructure. In case of an overprovisioning, he has invested in more infrastructure than is actually needed. While this allows for a smooth operation of the system, it incurs unnecessary cost. More common is an underprovisioning of resources, where the operator has not provided enough resources to support all the users which are interested in the MMVE. There are many reported cases of MMVEs, whose infrastructure was not sufficient to meet demand during launch, resulting in bad service for customers as well as bad publicity [35]. This in turn can cost an MMVE both current and potential future customers.

Other costs for MMVEs are the initial and continued production of assets for the virtual world, e.g. new areas and activities for MMOGs, and user support.

1.4 Peer-to-Peer Architecture

A potential way to reduce or even eliminate this infrastructure cost is to use a peer-to-peer infrastructure to run the virtual world. With a P2P architecture the entire system is run by the end-users (peers) which serve as both clients and servers. All functionality is distributed among the peers, without need for any central or pre-deployed infrastructure. Figure 1.2 compares the basic C/S and P2P architectures.

In a P2P-based MMVE the entire infrastructure of the system can be provided by the users. By distributing the virtual world among the peers and thus eliminating the need for servers, the cost of operating an MMVE is significantly reduced. P2P-based MMVEs also have the potential to eliminate the problem problems of over- and underprovisioning infrastructure resources. As all users bring their own resources into the system, the system can potentially scale with the number of users in the system, without requiring a priori knowledge about this number. P2P can therefore potentially be a disruptive technology for the MMVE market, reducing the barrier of entry and

operating costs. This significantly reduces the financial risks associated with launching an MMVE.

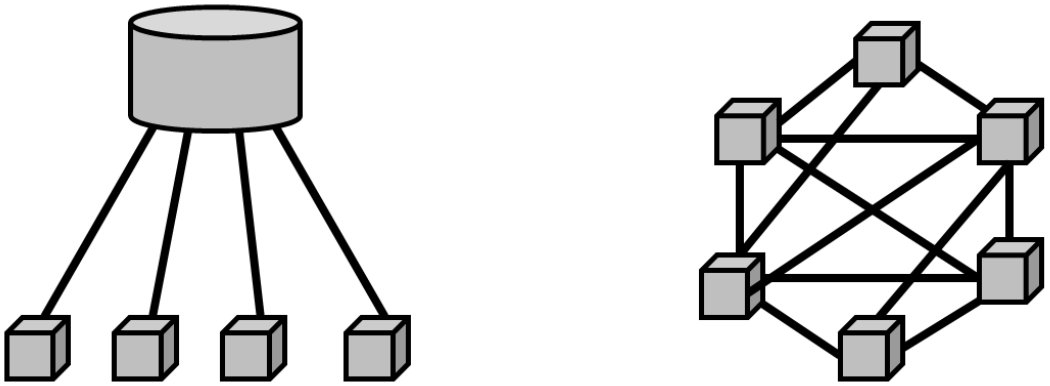

Figure 1.2: Client/Server vs. Peer-to-Peer - In a C/S-based MMVE, all users run clients which are connected to a server cluster which stores and runs the virtual world. In contrast, a P2P-based MMVE has no central architecture and the virtual world is stored and run entirely on the peers.

1.5 Update Propagation for P2P-based MMVEs

Achieving the goal of a P2P-based MMVEs is an open topic of research, with a significant number of challenges such as consistency [36, 37, 38], security [39, 40] or persistency [41]. One key challenge is to develop a suitable update propagation system [42].

The actions of users' avatars influence both the virtual world and other users' avatars. It is therefore necessary to inform the other users about the resulting changes to the virtual world. This happens through updates messages which contain the relevant information about these changes. These update messages need to be sent (propagated) to the relevant users. We thus define update propagation in MMVEs as follows.

Definition 2:

Update propagation in MMVEs is the task of forwarding update messages containing information about changes in the virtual to the peers which need to be informed about these actions.

Update propagation is one of the core features of any MMVE. Without a scal-

able, real-time update propagation system, an MMVE cannot function. A P2P-based update propagation system which meets these requirements is therefore a necessary requirement to enable P2P-based MMVEs.

1.6 Contributions

The goal of this thesis was therefore to develop a scalable, interactive update propagation system to enable peer-to-peer-based massively multi-user virtual environments. This has been achieved through three core contributions.

- **Update Propagation Overlay**: We have developed a dynamic update propagation overlay network. This overlay is based on the peers' positions in the virtual world and their available resources. It uses super-peers for overlay management and dynamic zoning of the virtual world for coordinator assignment.
- **Coordinator Election Algorithm**: We have also developed a scalable coordinator election algorithm which allows the election of suitable super-peers for our overlay. This algorithm is based on the existing dynamic overlay and samples a subset of peers to provide a scalable solution.
- **Available Bandwidth Estimation**: Finally, both our overlay and election algorithm require information about the available bandwidth in a peer-to-peer system. We have therefore developed an algorithm to estimate the available bandwidth of peers in a fully distributed peer-to-peer system. This algorithm is based on a combination of passive observation and active measurement to provide an accurate, responsive solution with limited system interference.

1.7 Thesis Organization

Based on the contributions outlined above, this thesis is structured as follows.

Chapter 2: System Model

Chapter 2 presents the system model on which our approach is based. We give a more detailed description of MMVEs and discuss the requirements for our approach which result from this. We also give an overview of the peers@play project in which

this thesis was developed and describe the parts of the peers@play system which our approach utilizes.

Chapter 3: Related Work

In Chapter 3 we give a detailed overview of related work in P2P-based MMVEs, specifically regarding update propagation. We also do the same for related work in the areas of election algorithms and available bandwidth measurement, with special focus on P2P systems. We then discuss the differences and advantages our approach provides over this related work.

Chapter 4: Update Propagation

Chapter 4 presents a detailed design of our update propagation approach. First, we present our core contribution in Sections 4.1 to 4.3: The dynamic update propagation overlay, based on available resources and quad-tree-based zoning.

Any hierarchical overlay requires suitable coordinators. This necessitates both the ability to determine what constitutes suitable coordinators and the ability to select them. In Section 4.4 we present both a metric to determine suitability and an election algorithm which allows us to select coordinators based on this suitability.

Finally, our suitability metric and update propagation overlay require information about the available bandwidth of the peers. In Section 4.5 we therefore present an algorithm to estimate the available bandwidth for peers in a fully distributed P2P system.

Chapter 5: Evaluation

In Chapter 5 we perform a detailed evaluation of all three contributions provided in Chapter 4. The results show that our approach provides a scalable and low-delay solution for update propagation in P2P-based MMVEs.

Chapter 6: Conclusion and Outlook

This thesis concludes with a summary of the presented work and an outlook on possible future work in Chapter 6.

Chapter 2

System Model & Requirements

In this chapter we will present our system model. First, we will take a more detailed look at MMVEs. Second, we will introduce the architecture of the peers@play project, for which our update propagation algorithm was developed. Finally, we will look at the requirements for update propagation in P2P-based MMVEs.

2.1 Massively Multi-user Virtual Environments

We have previously given a definition of MMVEs and will now present this concept as our system model in more detail, based on our work in [42]. Our system model consists of a number of users that want to participate in an MMVE, their devices (hosts), a communication network (the Internet) and the P2P-MMVE software. The number of users is a priori undetermined and may change dynamically. We make no assumptions about the geographic position of these users in the real (i.e. non-virtual) world. Each user connects to the MMVEs using a device with sufficient resources to run the MMVE software, e.g. a personal computer or tablet. We call these devices *peers*. We assume the P2P-MMVE software to be installed on each device. In practice most MMVEs distribute this software via discs or internet download. Note that we do not assume the existence of any kind of central server infrastructure. A peer can join and leave the system at any time.

To participate, a user activates one of her devices and starts the P2P-MMVE software. After she is done participating, she stops the software and deactivates the device. A user may change devices between sessions in the virtual world. The virtual world

is presented to the participant through a 2D or 3D graphical representation. Each participant is depicted in the virtual world by virtual representation of herself, called her *avatar*.

The participant can use this avatar to interact with the virtual world or with the avatars of other participants. The interactions are events in the virtual world, which can result in changes to the world. When such a change has occurred, the other peers in the system must be informed about it, so the change can be presented to the user. This is done via update messages, which must be propagated to all peers for whom the change to the virtual world is relevant.

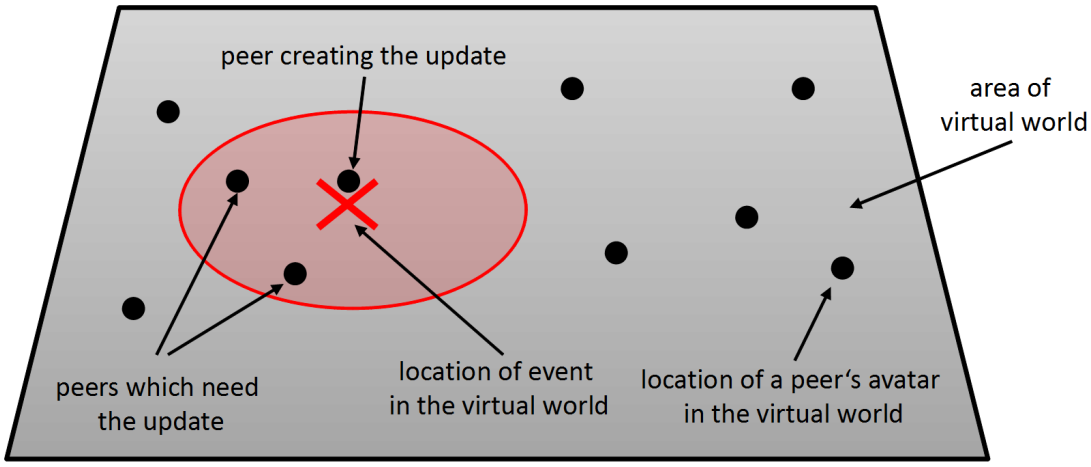

Figure 2.1: Overview of an MMVE - Schematic representation of a virtual world

Figure 2.1 shows a schematic representation of a virtual world. In this figure the user's avatars are shown as circles. Note that the avatar's location on the virtual world is decoupled from their hosts location in the real world, i.e. two hosts whose avatars are adjacent in the virtual world may be on different continents in the real world.

The figure also shows an example interaction. Consider a virtual world where a participant interacts with the world by picking up an object from the ground (at the marked location). One of the events resulting from this interaction is the disappearance of the object from the ground. Therefore an update message for this event must be created and propagated to all peers for which this event is relevant, e.g. all peers whose avatars can see the object. These peers are marked by a radius around the event

location. Note that the message does not need to be propagated to those peers whose avatars are too far away to see the event in the virtual world.

2.2 peers@play

This work was performed in the context of the peers@play project. Its goal is to "investigate how peer-to-peer technology can be used to create distributed interactive world models that are able to fulfil the highest requirements with respect to scalability, security and consistency" [43]. To achieve these goals, the project partners have developed and implemented a middleware architecture for P2P-based MMVEs. This architecture has been designed to enable evaluation of P2P protocols and overlays with large number of peers, while providing low overhead and allowing a single code base for evaluation and release software. It is depicted in Figure 2.2.

The contribution of this thesis to the overall architecture is the update propagation component, which allows scalable, low-latency propagation of update messages to all relevant peers. In the following we will briefly discuss the elements of this architecture which are utilized by our update propagation component.

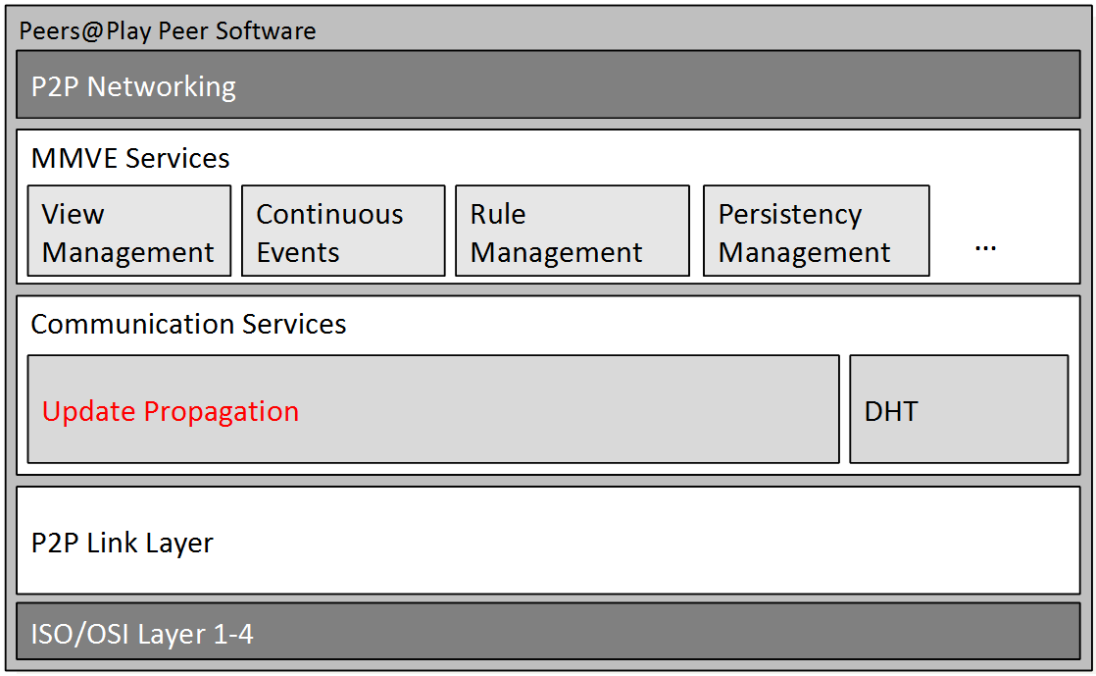

Figure 2.2: peers@play - Architecture of the peers@play peer software

2.2.1 P2P Link Layer

The P2P Link Layer, developed by Holzapfel et al. [44] is built on top of TCP/UDP transport and provides the update propagation system with two key functionalities.

First, the Link Manager allows us to open reliable direct connections between peers. These connections then form the basis of our update propagation overlay. The Link Manager can open connections even in the presence of NAT, using techniques such as relaying, hole punching [45] and Mapping Filtering Behaviour (MFB), a protocol developed to determine the NAT characteristics of a host [46]. Connections can also be reused and shared by the various overlays used by the system (e.g. a storage overlay and the update propagation overlay), in order to improve efficiency. As connections can be used by multiple overlays, the Link Manager also provides a priority system for messages, in order to deal with the problem of message delays due to queuing. Update or overlay maintenance messages can thus be sent with a higher priority than, for example, file transfers.

Second, the Link Manager provides us with bootstrapping capabilities, i.e. the ability the start our P2P system and allow additional nodes to find and enter it. To do so, it supports various techniques, including an IRC-based method described in [47].

2.2.2 P2P Storage

The update propagation also makes use of the P2P storage layer, which provides persistent storage via a DHT overlay such as Chord [48]. This storage overlay is initialized before the update propagation overlay and is used for bootstrapping purposes and to store and retrieve information about the hierarchy of the the update propagation overlay. The storage layer is designed to support a variety of DHT overlay types and provides reliability through replication.

2.2.3 Gears4Net

Evaluating large-scale P2P systems can be challenging, particularly without resorting to simulation or emulation. In order to evaluate the scalability of such a system, it is is necessary to run a large number of peers. However, it is impractical to set up a massive number of individual hosts running an instance of the P2P software. One solution to this problem is to run a large number of peer instances per host. In order to be able

to do so, the underlying system must make efficient use of the host's resources, such as memory and threads. As a consequence all components of the peers@play system have been built on Gears4Net (G4N) [49], an asynchronous programming model for scalable and distributed applications.

The G4N programming model is based on the concepts of protocols and schedulers. Protocols are highly decoupled entities which contain the main code of the application. They communicate with each other by message passing via message queues. The execution of the protocols is coordinated by schedulers. Each scheduler can have one or more worker threads which it assigns dynamically to the various protocols. This approach allows G4N the execution of many instances (protocols) of our P2P software with a fixed number of threads and enables us to run a large number of peer instances in parallel on a single host. For a full description of G4N see [50].

2.3 Requirements

We have performed an analysis of general requirements for P2P-based MMVEs [42]. Similar work has been performed by Fan et al. [51]. Based on these analyses we have identified three key requirements for update propagation in P2P-based MMVEs: (1) Scalability, (2) Interactivity and (3) Self-Organization. We will now discuss these requirements in detail.

2.3.1 Interactivity

In MMVEs users interact with each other and the virtual world in real-time. This means that high delay during update messages propagation can have a negative impact on the users' experience. This delay is the time difference between an interaction and the perception of the resulting event by another user. Studies have shown that users can tolerate between 60 and 500 ms of delay.[52] [14] [53] [54] [55] [55].

Within this range, the specific amount of delay that is tolerated by the users depends on the specific application. MMOGs in particular often feature frequent fast-paced interaction between the players, and thus are within the lower band of this range.

In short, if an MMVE is not able to provide the necessary level of interactivity, the user experience may be significantly diminished. It may even become impossible to participate in the virtual world. Therefore, the delay between a user's actions and

the MMVE's observable reactions to them must be kept as short as possible. To do so, the P2P system should try to deliver update messages containing changes to the world state as quickly as possible to minimize the experienced delay.

Interactivity is a requirement that is valid for some P2P systems, e.g., telephony systems, but only of secondary importance for others, e.g., content distribution systems. For MMVEs however, providing the needed level of interactivity is an essential requirement.

2.3.2 Scalability

As the name implies, MMVEs involve a large number of concurrent users, making scalability a key issue in their development. Eve Online [56] currently claims to hold the record of concurrent users for a CS architecture with over 60.000 players sharing the same virtual world at the same time [13]. Note that this number is limited not by player interest, but by the architecture and technology of MMVEs [57]. A common way to deal with these limitations is to distribute the users of an MMVE among a number of virtual worlds which run in parallel. The disadvantage of this approach is that users can only interact with users which are in the same virtual world as they are and not with the complete user base.

Scalability is often mentioned as one of the main advantages a P2P over a CS architecture (e.g. [58]). However, this can not be taken for granted. Without further effort, a P2P-based system may even scale significantly worse than a CS-based one. As an example, the bandwidth of each individual peer's network connection is usually much lower than that of a game server's. Therefore, it is very easy to overwhelm a single peer a large number of update messages. If peers are connected to the network using communication technologies with asymmetric up- and download bandwidth (e.g., ADSL), this is especially true for outbound traffic.

Therefore scalability is one of the key requirements for a suitable update propagation approach.

2.3.3 Self-Organization

In the P2P paradigm each users brings their own resources, such as CPU power and bandwidth into the system. This can provide an advantage when trying to fulfil the scalability requirement as each additional user also adds additional resources. However

the resources contributed by most individual users are somewhat limited. As an example, we have already touched upon the issue of asymmetric bandwidth. In order to provide its service, a P2P-based MMVE must therefore be able to make efficient use of the resources available to it.

Another challenge is the fluctuation of these resources. As users can enter and leave the system at any time, the available resource can fluctuate. P2P-based MMVEs must therefore be able to cope with these fluctuations in order to provide a stable service to it users. Therefore any components of such a system, including the update propagation require a high level of self-organization as they must constantly adapt themselves to the resources currently available to the system.

2.3.4 Additional Requirements

The three requirements we have just discussed are essential to an any update propagation system for P2P-based MMVEs. Once these have been fulfilled, additional requirements may be considered. Two requirements which we consider important for commercial deployment of P2P-based MMVEs are Consistency and Security. Consistency refers to the fact, that users are ideally presented with a consistent view a world, i.e. a view that is identical for all users in the system. Security refers to issues such as authentication and protection against fraud in the virtual world (e.g. cheating in online games). While these issues are outside the scope of this thesis, they are part of the work performed in the peers@play project. Some security features are part of the P2P Link Layer [44], while consistency features are provided by the consistency manager which is built on top of our update propagation [36][37][37].

Chapter 3

Related Work

Over the last few years there has been a significant amount of research into the feasibility of P2P-based MMVEs. In this chapter we will first give a thorough overview of related work in the area of P2P-based MMVEs, with a focus on update propagation. The various approaches we will discuss can be broadly divided into two main categories: hierarchical and non-hierarchical systems. In non-hierarchical systems all peers perform the same functions, i.e. no peer has more authority or responsibility than other peers. In these systems, the overlay has a flat structure. In contrast, hierarchical systems use so-called super-peers in their overlays to organize the system. This results in an overlay where certain peers are assigned special tasks and responsibilities and a hierarchical structure of the overlay, where peers rely on these super-peers for certain functions.

A special case of hierarchical overlays are hybrid systems, which are not fully distributed systems, but utilize some P2P techniques while retaining some server infrastructure. In these systems, P2P technologies are generally used to reduce the load on the servers, while avoiding some of the challenges associates with fully distributed P2P systems.

We will first introduce the various approaches for P2P-MMVES that have been developed over recent years, with particular focus on their update propagation components. We compare and contrast these components to the system developed in this thesis. Subsequently we will do the same for their super-peer election subsystems, while also providing an overview of related work regarding election in distributed systems. Then we will look at related work regarding the issue of determining available bandwidth in fully distributed P2P systems.

Finally we will give a summary of how our system is different compared to existing work and which advantages it confers in the areas of update propagation, super-peer election and available bandwidth estimation.

3.1 Update Propagation Overlays

In the following we will look more closely at the three categories of approaches for update propagation overlays in P2P-MMVEs: non-hierarchical, hierarchical and hybrid systems.

3.1.1 Non-hierarchical

An early non-hierarchical system was Solipsis [59] [60]. In Solipsis, peers create direct connections to all peers in their awareness radius. The awareness radius is the part of the virtual which their corresponding avatar can perceive and is also often called *area-of-interest* or *AoI*. (See Chapter 4.1 for a more detailed discussion of this concept.) A peer is thus connected to those peers whose avatars are located nearby in the virtual world. These neighbours then inform each other when new peers enter in each other's AoIs and therefore when new connections need to be opened. This concept is called *mutual notification*.

A major problem with Solipsis is the global connectivity of the system, i.e. the P2P overlay can be partitioned in some circumstances. As long as a peer the remains inside the convex hull formed by the positions of its neighbours, the system generally retains its global connectivity. However there are some situations in which the detection of new neighbours - and thus global connectivity - cannot be guaranteed [61]. In addition, this method generates notable message overhead. For any given position change, Solipsis must check the distances between disconnected peers and send messages about new connections to its neighbour peers.

A common approach for non-hierarchical P2P-based MMVEs is to partition the world by using Voronoi diagrams [62] and to create an update propagation overlay based on this partitioning. A Voronoi diagram can partition a virtual world in such a way, that it is divided in n zones, one for each peer. Each zone then contains all points closer the location of the corresponding peer than to that of any other peer.

Thus in Voronoi-based approaches, each peer builds a Voronoi diagram based on its own and its neighbours' location in the virtual world. The peers (neighbours) included in this diagram are determined by the peer's AoI. Each peer thus builds its own local partitioning of the virtual world. Note that this is fundamentally similar to Solipsis. All peers have some knowledge of their neighbourhood and build direct overlay connections with their neighbour peers. All Voronoi-based approaches also feature some type of mutual notification.

Variants of this approach are used by VON [61], VCS[63], NOMAD[64], Vorogame [65] and [66] and Solipsis 2.0 [67]. We will now look at some of these approaches in more detail.

In VON by Hu et al. [61], each peer communicates directly with a small number of neighbour peers. These neighbours are determined by partitioning the virtual world using Voronoi diagrams. Each peer builds such a diagram for the part of the virtual world inside its area-of-interest. The peers covered by this diagram are either classified as *enclosing neighbours* (i.e. the peers whose regions are adjacent to the peer's region) or *boundary neighbours* (i.e. the peers whose regions overlap with the area-of-interest). It is the purpose of the boundary neighbours to inform the peer of new neighbours entering its AoI and thus to ensure the consistency of the overlay topology. In this way, peers mutually notify each other when they have to adjust the overlay.

A forwarding scheme [68] was later added to reduce the number of connections in the overlay. Using this scheme, peers only have direct connections to their enclosing neighbours. To reach boundary neighbour, the enclosing neighbours keep forwarding the update until it reaches the target peer. This approach was further optimized by constructing a spanning tree among the neighbours (called *Vorocast*) [69] or alternatively by modifying the frequency of update propagation based on the hop count distance of the target from the originating peer (called *Fibocast*) [69].

VON can also reduce the number of neighbours to which a peer connects, in order to deal with crowding in the virtual world (i.e. when a peer does not have sufficient resources to communicate with all potential neighbours) [70]. It does not attempt to maximize usage of the available resources of a peer, however, in fact bounding resource use at each peer based on the area-of-interest. In addition, a partitioning of the overlay is also possible when two nodes move too fast for mutual notification to work [68] or when two adjacent boundary neighbours leave the game simultaneously [71].

VCS [63] also partitions the world based on a Voronoi diagram and uses the equivalent Delauney graph as the overlay network. This means that all peers are connected directly to the peers responsible for their adjacent zones. Update propagation to peers which are not direct neighbours is possible through forwarding along the Delauney graph. To improve the high delay resulting from the forwarding, two extensions were proposed. First, the addition of long range contacts (LRC) [72] which represent short cuts to distant areas of the overlay, thus reducing the hop count for update propagation. This was later replaced with a more efficient approach using a hierarchical layering of Delauney graphs, each probabilistically omitting an increasing number of nodes. This enables skipping nodes when forwarding, in order to reduce the number of hops [73].

VCS has also been adapted to spherical coordinate systems [70] and augmented with a tree-based data aggregation scheme, to further reduce network load [74].

The Voronoi based approach has proven popular enough to be used as the basis for various other approaches for P2P-based MMVEs (e.g. NOMAD [64] or Solipsis 2.0 [67]). It has also been extended by other authors, such as in VoroGame [65], which combines a Voronoi based overlay network for update propagation and a DHT for data distribution and storage of world data.

Other non-hierarchical approaches, i.e., not based on Voronoi diagrams, have also been proposed. In general they share some common characteristics with Solipsis and the Voronoi-based systems. These are in particular: direct connections to neighbour nodes and a form of mutual notification for overlay maintenance. PSense [75] is a good example of this. Once again, a peer maintains connections to neighbour peers, which are classified into *near nodes* and *sensor nodes*. Near nodes are rough equivalents to enclosing nodes in VON, while sensor nodes are similar to boundary nodes and used for mutual notification in order to maintain the overlay. This approach also runs into similar problems as Voronoi-based systems, such as a potential partitioning of the overlay network and inefficient usage of the resources of the peers.

QuON [76] also has similar classes of neighbour peers, called *direct neighbours* and *binding neighbours*, once again roughly corresponding to enclosing and boundary neighbours in VON. However QuON uses a local quad-tree structure instead of a Voronoi diagram to manage and maintain its connections to these neighbours. As with PSense, overlay partitioning is still a possibility.

Kawahara et al. [77] [78] propose an approach with an extensive mutual notification component. Each peer keeps a fixed list of nearest neighbours and all nodes regularly exchange their neighbour lists. By sorting and comparing these lists, each peer can find new nodes and update its overlay connections. However, this exchange of neighbours creates significant overhead, which means only a small number of neighbours is feasible for each peer [71]. Even though this approach provides some backup mechanisms to reduce the likelihood of an overlay partitioning, it is still a possibility.

Non-hierarchical approaches consistently run into the so-called *awareness problem*[79]. Avatars must be aware of who their neighbours are in order to maintain the update propagation overlay. If they fail to open connections to qualified neighbours due to a lack of awareness, update propagation will be incomplete. Eventually this may even lead to a partitioning of the overlay and thus an effective failure of the P2P-MMVE system. The discussed approaches also struggle to make efficient use of the peers' available resources. Many systems such as VON or Kawahara actually bound resource use at each node and limit the number of neighbours at a level below what the node may be capable of handling. In addition, it is difficult to exploit the heterogeneity of available resources in a non-hierarchical system, due to the challenge of assigning higher load to peers with higher resources.

As a consequence, hierarchical systems were pursued, even as an extension of fundamentally non-hierarchical systems such as VON, of which a hierarchical variant was proposed [80] [81]. This variant allows resource-rich nodes (super-peers) to manage multiple regions in the Voronoi diagram, thus introducing a hierarchy based on available resources. As another example, COVER [79] adds a quad-tree-based super-peer hierarchy to a VON-like approach in order to provide 100% awareness. Another way to utilize supernodes with a Voronoi-based approach is to give locations in the virtual world a certain weight [82] when constructing the Voronoi diagram. By doing this it becomes possible to assign larger areas of the virtual world to more capable nodes. Some peers may even be assigned a point-sized region in the diagram, in which case their overlay dutys are taken over by the peer whose region surrounds that point.

3.1.2 Hierarchical P2P-based MMVEs

While early approaches such as Solipsis and VON were generally non-hierarchical, later approaches are built as hierarchical P2P systems, using superpeers to perform various

functions in the system. As mentioned above, even systems such as VON, which started out as fully distributed systems later added some hierarchical elements. While hierarchical and non-hierarchical approaches were developed in parallel, a trend towards increasingly hierarchical systems can be observed.

The basic idea behind most hierarchical approaches is to assign additional tasks in the system to certain peers (sometimes called *super-peers* or *coordinators*). Usually the world is divided into regions, with a peer serving as coordinator. The region size can be fixed or dynamically adjusted during run-time (e.g. based on the number of peers in a zone). The coordinator can then be responsible for overlay maintenance in its zone (thus eliminating the need for mutual notification and potentially solving the awareness problem) and can also perform some update propagation function. The latter allows peers with above average available resources to contribute more to the system than in a non-hierarchical overlay, thus making more efficient use of the system's resources.

One of the earliest hierarchical approaches was SimMUD [83]. This approach partitions the virtual world into regions and maps them onto a Pastry [84] key space. Each region is assigned a peer as a coordinator, which then serves as the root of a Scribe [85] multicast tree for that region. Maintaining virtual objects is also the responsibility of the coordinators, which manage the objects and ensure consistency. Updates to the objects are sent to the responsible coordinator, which then propagate any successful updates to the object's region.

SimMUD is a fixed zone approach which can lead to an overloaded coordinator when too many peers enter a zone. It is also not capable of propagating updates across zone borders. Due to the use of a DHT for update propagation, SimMUD also struggles to meet interactivity requirements. Here the majority of updates take between one and six hops to be delivered. However, due to relaying, some updates require more than 50 hops to reach their destination peer. The usage of DHT overlays such as Chord [48], CAN [86] or Pastry [84] for P2P-based MMVEs was abandoned early on in favour of specialized overlays, mostly due to the comparably high hop count for update propagation [87] [88] [71]. Even approaches that started out as entirely DHT-based system, such as OPeN [88] were later modified to include direct connections [89].

Iimura et al [90] use a DHT only for bootstrapping and persistent storage, similar to our approach. The world is also divided into zones, for each of which a coordinator

peer (called *zone owner*) is responsible. The peers that are part of a zone open direct connections to the zone owner. All updates are propagated via these connections and the coordinator. Similar to SimMUD, the zones have a fixed size, which means the scalability of each zone is limited by the coordinator's available resources. In fact, according to the authors, their approach is only suitable to small-sized virtual environments with less than 500 nodes. The system also is not capable of propagating updates across zone borders.

The MOPAR scheme [71] also divides the world into several regions and sends all updates in a region via a coordinator node. For the purposes of latency, proposals which propagate all updates via a coordinator (such as HYDRA [91]) achieve latency similar to a client/server-based approach. The zones in MOPAR are hexagonal in shape and a peer is part of several zones, i.e. it' AoI covers several of zones. The peers are assigned different tasks, as *home nodes*, *master nodes* or *slave nodes*. Slave nodes are standard peers, while home nodes serve to register master nodes, which in turn are responsible for one zone each. The master nodes then exchange neighbour information amongst each other, informing slave nodes of neighbour changes when necessary.

The Distributed Event Delivery scheme by Yamamoto et al. [92] divides the world into regions. Again, one peer serves as coordinator for each region, responsible for update propagation. In addition to previous approaches, the coordinator can build a hierarchy of sub-coordinators within its zone in order to spread load in a crowded zone. While coordinators are not selected based on their resources, the system occasionally removes and replaces coordinators with high delay. MM-VISA [93] extends the concept of a zoned world with coordinators by classifying virtual objects into various types based on their movement speed in the virtual world. Within each zone, the coordinator then builds multicast clusters based on this classification. Depending on its type, an object may be part of the clusters of neighbouring zones as well. The system uses hexagonal zones.

3.1.3 Hybrid P2P-based MMVEs

In addition to fully distributed and hierarchical systems, there have also been some proposed approaches which attempt to create hybrid systems. These systems combine P2P techniques with a server infrastructure. They have to deal with all the potential problems of a client/server approach, e.g. the infrastructure cost. However, using P2P

techniques can mitigate the issues caused by these problems, e.g. by reducing the load on and thus the cost of the servers. At the same time, having access to a server infrastructure can eliminate or mitigate some of the potential problems of P2P systems, such as dealing with bootstrapping (i.e. initial system set-up) or node failures.

Hybrid approaches generally fit into two classes. The first class uses its server infrastructure to solve issues such as bootstrapping, churn (i.e. handling node join and leave operations) and persistent data storage, but leaves update propagation to the peers. The second class essentially takes a hierarchical overlay, but replaces the coordinators with servers.

Time Prisoner [94] [95] [96] is part of the first class. It uses a server infrastructure to provide a bootstrapping service to the system and handles peer join and leave operations while the system runs. However the actual update propagation is handled like a traditional hierarchical system, which dynamically divides the world using a quad-tree. In this quad-tree,the quads are split and joined again based on the number of peers in the zone. Each zone has a coordinator peer which is responsible for update propagation.

Another example of the first class of hybrid systems is Lee et al.'s approach [97]. Again, this is a zone-based approach which uses a main server for login and bootstrapping purposes, but uses peers as zone coordinators. Similar to MOPAR [71], each peer can be interested in multiple zones and therefore connect to several coordinators. Chen et al. [98] also use servers for authentication and verification purposes but use peers as coordinators for the regions of the virtual world.

HYMS [99] is an example of the second class. Once again the virtual world is divided into zones. Usually a server is responsible for update propagation, but when a suitable client is available, it takes over responsibility for update propagation in the zone. The other clients in the zone then connect to this client instead of the server. This enables HYMS to make use of resource-rich peers and reduce load on the servers. Similarly Wehrle et al. [100] dynamically split the world into regions based on CAN [86]. The approach allows either a server or another peer to serve as coordinator. Another example for using servers as coordinators is Hyperverse ([101] [102]), which also uses a dynamic quad-tree to divide the world. A server is assigned to each quad and deals with overlay management and streaming of world content.

Beyond systems specifically designed for hybrid operation, it is generally possible to run those based on a hierarchical overlays as hybrid systems. Any coordinator can in theory be replaced by a server with no impact to system performance (in fact, in many cases performance can be increased due to the higher performance of a server over a peer). This applies to our approach as well (see Chapter 6 for more details on this).

3.2 Coordinator Election

Coordinator election for P2P-MMVEs requires the coordinator to be selected among a massive number of candidates. In addition, it is not sufficient to select any unique leader, but the leader must be suitable, i.e. have sufficient available resources to perform coordinator tasks.

Election is a well-known problem in the area of distributed systems. Classic algorithms to "elect a unique leader from a fixed set of nodes" [103] include the Bully [104] and the Chang and Roberts algorithms [105]. However these algorithms are often designed for a specific network topology and system model (e.g., a ring [105][106]) and have a time and message complexity ($O(n\log n)$) not suitable for a massive number of candidate nodes in the system [103]. In addition, they are generally only concerned with finding a unique leader and don't take the suitability of nodes into account.

Additional work has been done on leader election in ad hoc networks ([107] [108] [109] [110]). These approaches are primarily designed to deal with a highly dynamic non-hierarchical topology. Some of this work also takes suitability into account (e.g., [111][109]). However, the main issue remains that these approaches have not been designed with massive user numbers in mind. An election involves all nodes in the network, which is not feasible when dealing with the potential number of peers in a P2P-MMVE. On the other hand, if a hierarchical overlay is used for the P2P-MMVE, the election can potentially utilize the this overlay and does not have to deal with a non-hierarchical structure.

Only a limited amount of work has been done on coordinator election specific to P2P-MMVEs. Even though many hierarchical approaches assume the ability to elect a coordinator, they either do not provide an algorithm for doing so, or use a very simple solution. A number of systems use the topology of the underlying P2P overlay network

to select coordinators. In [83] each region is assigned a key from the Pastry key space. The peer whose ID is the closest to the region ID becomes the coordinator. In [98], each peer manages those regions that are numerically nearest its ID in the DHT. [97] uses JXTA to organize peers in groups: if a region needs a new coordinator, a peer from the group is used. Some approaches use the avatar position in the virtual environment to select coordinators. A peer is selected if there is no coordinator in the user's zone [90] [71], or if the user's position is the closest one to the geometric center of the region [79].

Even though all of these approaches put additional load on the coordinators, they do not select the coordinator based on its specific suitability. That is, they select their coordinators based on criteria unrelated to their suitability. This means that they cannot take advantage of resource heterogeneity, and might even select peers as coordinator which do not have sufficient resources to handle this additional load. The pool of potential coordinator candidates is also often limited to one zone. In contrast, a suitable coordinator election algorithm would be quickly able to elect coordinators based on their suitability, among a massive number of peers in the system.

3.2.1 Election Parameters

In order to be able to select the most suitable peers as coordinators, we must also be able to determine the suitability of an individual peer in order to make comparisons. This suitability depends on the specific tasks that the coordinator needs to perform (e.g. in order to perform update propagation tasks, it needs sufficient bandwidth).

Some approaches note that their coordinators have higher requirements than usual, e.g. regarding bandwidth [92] [112] [81], CPU power [92], delay [112], trustworthiness [81], stability [81] and storage capacity [93]. However none of these approaches provide specific techniques for determining the values for these parameters. While the ability to determine suitable peers is crucial to them, they explicitly or implicitly assume their ability to do so.

3.3 Bandwidth Estimation

Determining the available bandwidth between two nodes is a well-known problem [113]. However, most approaches focus on estimating the data rate for a specific connection

between two hosts (e.g., [114] [115] [116] [117] [118]). In contrast, for P2P-MMVEs it is more relevant to know the overall network capacity of a peer, which it can utilize for connections to other peers. This information can be used when building and maintaining the overlay or for the selection of suitable super-peers. In the following we therefore discuss existing P2P approaches that deal with measurements of a peer's total capacity.

Eigenspeed [119] uses passive observation only. Each peer observes the traffic to its neighbours in the overlay and stores the maximum bandwidth it has achieved with each of them. This information is then aggregated across multiple nodes and combined into a consensus view using principal component analysis. The primary contribution of Eigenspeed is on the issue of trust. It assumes that peers cannot be trusted to report their own bandwidth to the system. Thus the focus is on preventing attackers or colluding groups of attackers from intentionally reporting incorrect bandwidth. From the viewpoint of bandwidth estimation, Eigenspeed has a number of drawbacks. First, little traffic between two peers can lead to significant underestimations. This is mitigated somewhat by combining observations from multiple peers. However, this does not work if a peer is experiencing low traffic volume to all its neighbour peers. Second, Eigenspeed assumes stable network capacities. In contrast to this, our approach gets more accurate results in low traffic scenarios by injecting traffic periodically and is able to handle fluctuating network capacities.

ThunderDome [120] uses active measurement only. Its key concept is the use of pairwise active measurements between peers, called bandwidth probes. As the result of such a measurement is always limited by the slowest member of each pair, ThunderDome proposes an algorithm that schedules pairwise bandwidth probes for all peers in the system. Using a tournament system, peers whose bandwidth has not yet been determined are repeatedly paired and execute a bandwidth probe. This approach takes some time to converge. The authors first prove a logarithmic lower bound for the tournament's runtime, but also show that improvements are possible under certain assumptions about the bandwidth distribution in the system [121]. However, as with Eigenspeed, the results may already be outdated once the tournament has finished. Also, the system relies on existing connections between peers. Thus the bandwidth is depending on the underlay characteristics of the connections. Alternative connections

might show different behaviour. Also, the estimation may change if the background traffic changes during measuring the different connections of a peer.

3.4 Summary

As detailed above, non-hierarchical systems struggle to maintain overlay cohesion. As each peer has the exact same duties in the system, they also cannot take advantage of more resource-rich peers. In fact, many non-hierarchical system bound the resource use of a node, thus wasting additional resources (e.g. bandwidth, CPU) available to the system. They do often provide an advantage when it comes to interactivity, however. Most updates are propagated to their target peer via direct connections (one hop in the overlay). In contrast, many hierarchical systems use the coordinator for update propagation, resulting in propagation over a minimum of two hops. Hierarchical systems also have the advantage of being easily converted to operation in a hybrid mode, where some server infrastructure is added to increase performance of the system.

Existing approaches generally make inefficient use of the resources available to the system on both peers and coordinators. Resource usage on peers is often bounded, even though they could contribute further to the system. When it comes to coordinators, existing approaches often make two incorrect assumptions. First, that they can efficiently select coordinators among the massive number of peers in the system. Second, that they know how much bandwidth (and other) resources a peer can contribute to the system.

In this thesis we have built a hierarchical system that is based on some familiar concepts, such as quad-tree-based dynamic zoning and area-of-interest-based update filtering. We advance the state of the art by making maximum usage of available resources, especially bandwidth resources. Peers open a maximum possible number of direct connections to their neighbour peers, only using coordinators for update propagation when peer resources are insufficient. This allows one hop update propagation in most cases, with the advantages of a hierarchical structure.

We also provide a means of efficiently selecting suitable coordinators based on their available resources. Our election algorithm utilizes the existing hierarchical overlay structure to sample a random subset of peers in the system. This allows us to significantly reduce the number of messages and peers involved in any given election. While

this means that we cannot guarantee the election of the most suited candidates, the large number of peers in the system makes it likely that the sampled subset contains sufficient suitable peers.

As the construction of the overlay requires information about the available bandwidth resources of the peers, we also provide a algorithm for the estimation of available bandwidth. In contrast to existing approaches, our bandwidth estimation is a combination of passive traffic observation and active traffic injection. Hence, results are immediately available. We also do not assume static capacity of the traffic/channels and adaptation to fluctuations in capacity can be done quickly.

Chapter 4

Update Propagation Overlay

Based on the system model and requirements presented in Chapter 2 we have developed a scalable, low-latency, self-organizing update propagation system for P2P-based MMVEs. This update propagation system consists of three components. These components are (1) Update Propagation Overlay, (2) Coordinator Election and (3) Available Bandwidth Estimation.

The update propagation overlay is a P2P overlay network which is used to propagate update messages to the interested peers. It is built to maximize the usage of available bandwidth resources, to improve scalability and to reduce the amount of hops each message needs to traverse in the overlay to improve interactivity. The overlay consists of two parts. First, a horizontal overlay is formed between standard peers, based on proximity in the virtual world. Second, a vertical overlay is constructed based on dividing the virtual world into zones and assigning specially selected coordinator peers to each zone. These peers then perform additional overlay management functions.

The coordinator election component allows the system to select coordinators which are suitable to perform these additional functions. It consists of a suitability metric to determine which peers have the resources to perform coordinator functions and an election algorithm which allows a scalable selection of coordinators among the large number of peers in the system. It does so by sampling a subset of all peers, utilizing the existing propagation overlay for this purpose.

Finally, the available bandwidth estimation algorithms allows the system to determine how much bandwidth is available to each peer. This information is vital to both

the horizontal overlay construction as well as to determine the suitability of peers as coordinators.

The complete update propagation system offers reliable delivery of update messages to those (and only those) peers which have an interest in them. It provides an interactive service, by minimizing the number of hops in the update propagation overlay. By utilizing interest management and dynamic adaptation based on the available resources, it provides a scalable solution.

Chapters 4.1, 4.3 and 4.2 will describe the update propagation overlay in detail. We will discuss the coordinator election in Chapter 4.4 and the bandwidth estimation in Chapter 4.5.

4.1 Interest Management

In order to fulfil the requirement of scalability we first need to reduce the number of update message transmissions between peers. In a naïve approach each update message must be sent to each peer in the system. In practice this method does not scale, even for systems with a moderate amount of peers, i.e. more than 100 (see Chapter 5.1.2). Thus we need to reduce the number of peers an update message is sent to. We call this process *Update Filtering*.

A well-known technique to perform Update Filtering is Interest Management [122]. The concept of Interest Management is based on the fact that each avatar only perceives a small part of the virtual world at any given time. This is analogous to the real world: a person can only ever observe a very small area of the world. Therefore, the avatar is not interested in most events that occur in the virtual world, but only the subset of events that occur in the part of the world which it can observe, i.e. those that occur near it. By taking this locality of interest into account, it is possible to significantly reduce the number of updates that are sent to peers and thus to enhance the scalability of the system.

Interest Management is generally based on the well-known concept of the Area-of-Interest (AoI) [122]. Our update filtering algorithm which we will describe in the following, extends this with the additional concept of an Area-of-Effect [123].

The **Area of Interest** (AoI) is defined as the area of the virtual world, in which a certain peer's avatar can perceive events. This means that the peer is interested in

all updates which occur within this area or which influence it. The size of the area is determined by the perceptive abilities of the avatar. For example, an avatar may have a certain view range and can only see events occurring within this range. Thus, the AoI is circular with a radius equal to the view range and centered on the peer.

We extend this concept with a second type of area, called **Area of Effect** (AoE). We define the AoE as the area in the virtual world, in which a certain event affects the world. The size and shape of this area depends on the event it is associated with. As an example, picking up an object influences an area that is equivalent to the size of the object. As another example, if an avatar makes a sound, the AoE is determined by the distance the sound travels in the virtual world.

For our approach we will be using only circular-shaped AoIs and AoE. More complex shapes are possible and can provide improved update filtering through more accurate expression of interests and effects. The downside of more complex shapes is increased overhead. As this discussion is outside the scope of this thesis, we refer to Heger et al. [124] for more information.

In summary, the AoI is associated with an avatar and represents the part of the virtual world in which the avatar can be affected by events. The AoE is associated with an event and represents the part of the virtual world in which the event can affect peers. This means that by delivering an event to all peers whose avatar's AoI intersects with the event's AoE, we can ensure that the event is delivered to all peers whose avatars are interested in it. The algorithm to determine all peers p that a given update u should be delivered to is thus very simple: it compares the AoE of u with the AoI of each p. If they intersect, u is delivered to p. Otherwise, it is not delivered.

This algorithm is given in Algorithm 1. For a given update u it determines all peers p it should be delivered to. To do so, the algorithm intersects the AoE of u with the AoI of each p. If they intersect, u is delivered to p. Otherwise, it is not delivered.

Figure 4.1 illustrates this with an example. Avatar A is generating events E1 and E2. The circular areas around the events represent their AoEs, while the circular areas around avatars A, B, C and D represent their AoIs. As can be seen, event E1's AoE does not intersect with any avatar's AoI. This means there is no peer which has an interest in it. Therefore no update needs to be sent for this event at all. The AoE of event E2 intersects with the AoI of both avatar B and Peer C. This means that both avatars are affected by the event and need to receive its corresponding update. Peer

Algorithm 1 Interest Management Algorithm

```
procedure InterestManagement(u, listOfPeers)
    for all (peerinlistOfPeers) do
        if (u.aoe ∩ peer.aoi) then
            propagate u to peer
        end if
    end for
end procedure
```

D however is not interested in the event. Peer A therefore delivers the update E2 to peers B and C. In contrast, update E1 does not need to be delivered to any peer.

Using our update filtering algorithm we can now determine the target set of peers that should receive a given update. With this information, we then need to deliver the update to these peers. To do so we need an update propagation overlay, i.e. connections between peers which we can use to deliver the updates in a way that meets our requirements of scalability, interactivity and self-organization.

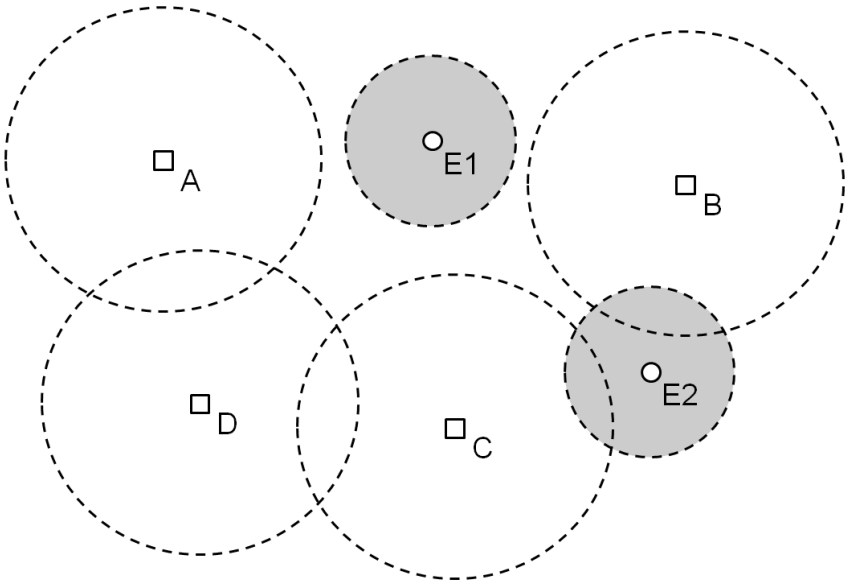

Figure 4.1: AoI/AoE Example - The AoE of Event 1 intersects with the AoIs of Peer C and B.

Our approach consists of two integrated overlay networks. First, a horizontal non-hierarchical overlay which allows peers to distribute updates among themselves. Second,

a vertical hierarchical overlay, which uses coordinator peers to manage the horizontal overlay and provide backup propagation when peer resources are insufficient. This allows us to provide low-latency update propagation via single hops in the horizontal overlay, while the coordinators enable a stable system and efficient usage of available resources.

Please note that in the following we will assume that the user's avatar is the only virtual object which is managed by a peer. Due to this, we will use the terms avatar and peer interchangeably. When we refer to a peer being in a certain location of the virtual world we are indicating that its avatar is located there. While this is not a realistic assumption, our approach can be easily extended to work with multiple virtual objects per peer. See Chapter 6.1 for more on this issue.

4.2 Horizontal Update Propagation Overlay

In this section we will describe the horizontal element of the update propagation overlay. The core concept of this overlay is the **Area of Propagation** (AoP). In the following we will define the AoP and describe how to use it for achieving interactive update delivery.

In our approach, each peer is associated with its own surrounding AoP. A peer's AoP is the spatial area in the MMVE for which the peer has all necessary information to perform both update filtering and propagation on its own. More specifically, this means that the peer knows the current location and AoI of all peers in the AoP. The peer uses this knowledge to filter updates and deliver them to peers in the AoP directly. Updates affecting the MMVE outside the AoP are forwarded to the coordinator and processed there. The coordinator is a peer in the system which has been selected to perform additional overlay management and update propagation functionality in addition to serving as a normal peer. As mentioned, we will discuss how to select it in Chapter 4.4.

The size of the AoP is determined by two factors: its peer's maximum available upstream bandwidth and the location of its neighbouring peers. The available upstream determines the number of neighbours n to which the peer can send updates simultaneously. Given this upper limit, we choose the peer's AoP as a circular area whose diameter is set such that the AoP contains at most n neighbours.

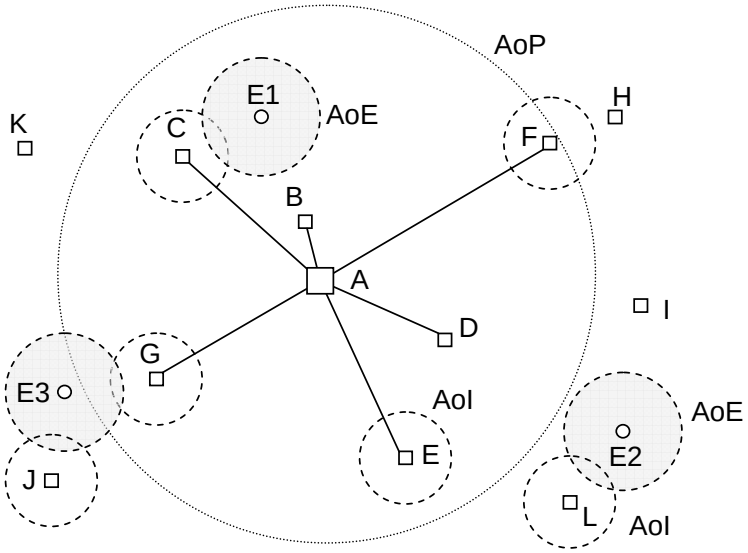

Figure 4.2: AoP Example - The AoP of peer A covers five other peers (B,C,D,E,G)

Figure 4.2 illustrates this concept. The peer of avatar A has sufficient bandwidth to send updates to six neighbours. Its AoP is therefore set to contain its six closest neighbours B to G. Updates affecting these peers are sent to them directly, resulting in a single hop in the overlay for update propagation. For example, this is the case for update E1. However, there is not enough bandwidth to communicate directly with the peers of avatars H to L, as they are outside A's AoP. To deliver an update to them, A sends the update to the coordinator which determines its target peers and forwards the update to them. An example for this is update E2. A special case occurs if the event is relevant both for peers inside and outside the AoP, e.g., event E3. In this case A delivers the update to all relevant peers in its AoP, while the coordinator delivers it to all relevant peers outside the AoP.

A more detailed description of this algorithm – including several additional special cases – is given in Section 4.2.2.

4.2.1 AoP Generation

In this section we describe how the AoP is created and maintained. As discussed previously, a peer's AoP is determined dynamically depending on its available upstream bandwidth and the location of the peers' avatars in the virtual world. In this section we

present the algorithm to do so. This algorithm determines the list of neighbours that are included in a given peer's AoP. It is executed regularly by the coordinator for all peers. If the members of an avatar's AoP have changed, the coordinator sends the peer an updated list of the members of its AoP. The coordinator also sends all movement updates of the AoP members to the peer. This algorithm is given in Algorithm 2. The algorithm has one input parameter, the peer p for whose avatar the AoP is to be determined. It uses the set of all peers ($peers$) to return the updated AoP for p ($p.aop$).

First, the algorithm calculates p's available upstream bandwidth by taking its maximum upstream bandwidth of p and subtracting the maximum bandwidth that is required by p's events. This subtraction is done because the peer needs bandwidth reserves to send updates to the coordinator, as the coordinator needs to know the location of all peers in order to execute this algorithm. This means that at a minimum each peer needs to send position updates to the coordinator regularly. In addition p needs bandwidth to send event updates to the coordinator, in case they occur outside its AoP. In some instances the AoP can be so small that all events occur outside the AoP. Therefore we reserve enough bandwidth to send all updates to the coordinator. The algorithm then takes p's available upstream bandwidth and divides it by the maximum bandwidth that is required by p's events. This gives us the maximum number of direct communications to other peers that the peer can support ($maxComm$).

Algorithm 2 AoP Generation Algorithm

1: **procedure** GENERATEAOP($peer$, $listOfPeers$)
2: $peer.aop = \emptyset$
3: $usedBw = maxEventBw + maxPosUpdateBw$
4: $availBw = p.upstream - usedBw$
5: $maxComm = round(availBw/p.maxEventBandwidth)$
6: $listOfPeers.sortByDistanceTo(p)$;
7: **for** $(i = 0 \rightarrow maxComm)$ **do** $peer.aop.add(peers[i])$
8: **end for**
9: **return** peer.aop
10: **end procedure**

The algorithm then sorts all peers' avatars by their distance to the input peer's avatar and calculates the AoP. To do so, it iterates through the sorted list of peers, beginning with the closest neighbour. As long as $maxComm$ is not reached, it adds

neighbours to the AoP. Based on the output of the algorithm, the coordinator constructs a list of the members of the AoP and sends it to the peer. When the peer has received this list, it determines the radius of its AoP by measuring the distance to the most distant avatar's peer in the AoP. It sets the radius of the AoP to this distance. Note that a special case occurs if some peers have exactly the same distance to p, e.g. peers H and J in Figure 4.1. This can lead to a non-circular AoP if the avatar of the first peer not in the AoP has the same distance as the avatar of the last peer in the AoP. To solve this case, the server reduces the size of the AoP before sending the information to the peer by iteratively removing the last peer from the AoP until the problem is resolved.

4.2.2 AoP-based Update Propagation

After detailing how the AoP of each avatar is determined, we specify our approach for update propagation in more detail. We assume that the coordinator executes Algorithm 2 regularly and that each peer knows the size and members of its current AoP. When an event occurs on one of the peers, it initiates the Update Propagation Algorithm given in Algorithm 3. The algorithm first compares the event's AoI with the peer's AoP. There are three possible cases: (1) the AoI can be completely within the AoP, (2) completely outside of the AoP, or (3) overlap its border and thus be partially inside and partially outside the AoP. If the AoI is completely within the AoP (Case 1), then the peer can determine the update's recipients and deliver the update without the assistance of the coordinator. This is the case, because the peer has all relevant information to do so as well as enough bandwidth to send the update directly. If the event's AoI is completely outside of the AoP (Case 2), then the peer cannot deliver the event itself, since it does not know any of the potential recipients. Therefore, the peer sends the update to the coordinator. The coordinator then determines the update's recipients and delivers it to them. Finally, if the event's AoI overlaps the edge of the AoP (Case 3), some of the event's potential recipients are members of the AoP, while others are not. Thus, the update needs to be delivered both by the peer and the coordinator. First, the peer determines all target peers inside its AoP and sends the update to them. In addition, the peer sends the update to the coordinator. The coordinator then determines all target peers outside the AoP and sends the update to them.

Algorithm 3 Update Propagation Algorithm

```
1: procedure PropagateUpdate(update)
2:     if (u.AoE ⊂ (p.AoP - max radius of AoI)) then
3:         local peer executes Algorithm 1
4:     else if (u.AoE ∩ p.AoP) then
5:         deliver update to coordinator
6:         coordinator executes Algorithm 1
7:     else
8:         local peer executes Algorithm 1
9:         deliver update to coordinator
10:        coordinator executes Algorithm 1
11:    end if
12: end procedure
```

There are two special situations to consider, shown in Figure 4.3 and Figure 4.4. The first one (see Figure 4.3) can occur in Case 1. If a peer A is located just outside of another peer B's AoP, A's AoI may intersect with B's AoP. Thus, A may be interested in events that occur near the border of B's AoP, even if the event's AoE is completely inside the AoP. For example, this is the case for event E1 in the figure. To solve this situation, we modify the check for Case 1 as follows. Instead of checking whether the event's AoE is completely within the AoP, we also check if the event's AoE is close enough to the edge of the AoP to potentially intersect with another AoI (using a predefined maximum AoI size). If this is the case, then the peer can not deliver this update completely by itself. Instead, the event is treated as if it belongs to Case 3.

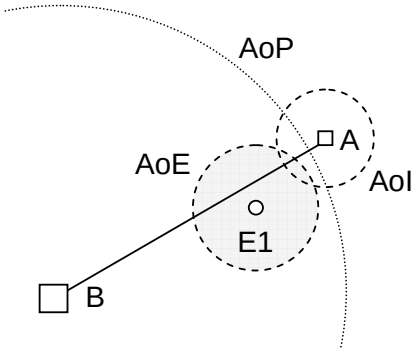

Figure 4.3: Special Situation I

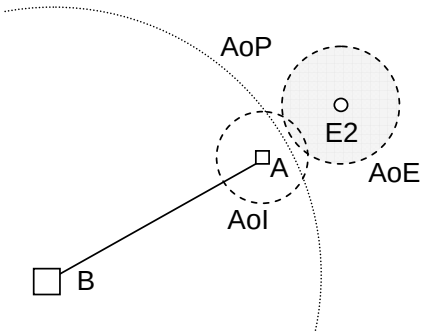

Figure 4.4: Special Situation II

The second special situation (see Figure 4.4) can occur in Cases 2 and 3. If an avatar A is located just inside another avatar B's AoP, A's AoI may be partially outside the AoP. In this case it is interested in some events that are completely outside of the AoP. To solve this, while checking which peers are target peers for an event, the server not just considers all peers outside the AoP, but also those peers whose avatar's AoI is partially outside of the AoP.

4.3 Vertical Update Propagation Overlay

Until now, we have assumed that the coordinator peer has sufficient resources to perform its coordination and propagation tasks for all peers in the system. This is clearly not a reasonable assumption, given the large number of peers in the system. While some peers can bring significant resources into the system, it is not realistic to assume that any peer in the system has enough resources to serve as coordinator for all other peers.

It should be noted that if the coordinator functionality is placed on a suitable server instead of a peer, the MMVE could be run as a hybrid P2P/Client-Server system. In this system the horizontal overlay would ensure that resources brought into the system by the peers are utilized before load is put onto the server. The server in turn would perform primarily coordination functions and only perform update propagation in those cases where the peers' resources are insufficient. This would reduce the load on the server, and thus result in a reduction of server resources required to run the system. This would in turn mitigate many of the problems associated with the traditional client/server architecture.

However, the goal of this thesis is to develop a fully distributed P2P update propagation system. Thus we cannot make the assumption that a server is available to perform coordinator functionality. In order to deal with this issue, we are adding a horizontal element to our update propagation overlay, which allows us to distribute coordinator functionality and thus load among a number of coordinators.

This distribution is based on the well-known concept of *zoning* (e.g. [90][92][93][91]). Such an approach divides the virtual world into several disjoint zones. Each zone is assigned its own coordinator peer, which performs coordination and propagation tasks only for the peers in its own zone.

There are two basic approaches for zoning in MMVEs, static and dynamic zoning. In static zoning (e.g. [125][83][90]) the virtual world is statically divided into fixed zones, which do not change over time. The advantage of this approach is the limited overhead it generates. However, such a system can fail if there is high load in a certain zone. For example, if there are so many peers in a given zone that the coordinator no longer has sufficient resources to perform its tasks, then the system will fail.

To solve this problem, dynamic zoning has been introduced (e.g. [126][99][79]). With dynamic zoning the size of the zones (and thus the number of coordinators in the system) can be dynamically adjusted according to current load. For example, when the number of peers in a zone results in more load than the coordinator can handle, the system can divide the zone into several new zones, assign an additional coordinator to each new zone and thus distribute the zones' load among these coordinators. Similarly, if several adjacent zones have few peers in them, thus resulting in very low load for their respective coordinators, these zones can be joined again. Their load is then combined on one coordinator, freeing the other peers from coordinator duties and reducing overhead in the system.

For our system, we have chosen a simple dynamic zoning approach based on a quad-tree. Each zone is a square, with the entire world initially covered by a single zone and thus managed by a single coordinator. As more peers enter the system and the load on the coordinator exceeds a threshold based on the coordinator's available resources, the zone is split into four equal sized squares. Four new coordinators are selected and one of them is assigned to each of the four new zones. The new coordinators then take over the overlay management and update propagation within the zone. The existing coordinator remains assigned to its existing zone, i.e. the zone covering the four new zones. It also establishes a direct overlay connection to the new coordinators, thus becoming their *parent coordinator*. If and when the number of peers in the new zones also exceed their coordinators thresholds, these zones continue to get split.

Similarly, if four adjacent zones which have previously been split, i.e. which are covered by a common parent coordinator, all have fewer peers than a certain minimum threshold, the zones can once again be joined into a single zone. This is the case, as the original coordinator now has sufficient resources available in order to manage all the peers in the four zones. Thus we can remove the overhead of the four additional coordinators. When the number of peers falls below the minimum threshold in a zone,

the responsible coordinator informs its parent coordinator about this. It also informs the parent coordinator if the minimum threshold is exceeded again in the future. If and only if all four zones have informed the parent coordinator that their number of peers is currently under the minimum threshold, the parent coordinator performs a zone join. It takes over the overlay maintenance and update propagation functionality from the four coordinators and joins the zones into one unified zone.

As peers continue to enter and leave the system, split and join operations are regularly performed. The result of these operations is a partitioning of the virtual world into zones with a vertical hierarchical overlay network among the coordinators. This overlay has the structure of a quad-tree, with the coordinators as nodes and the overlay connections between them as edges. The partitioning of the virtual word is based on this quad-tree structure. Figure 4.5 shows a quad-tree partitioning of the the virtual world, with the coordinator quad-tree on which this partitioning is based.

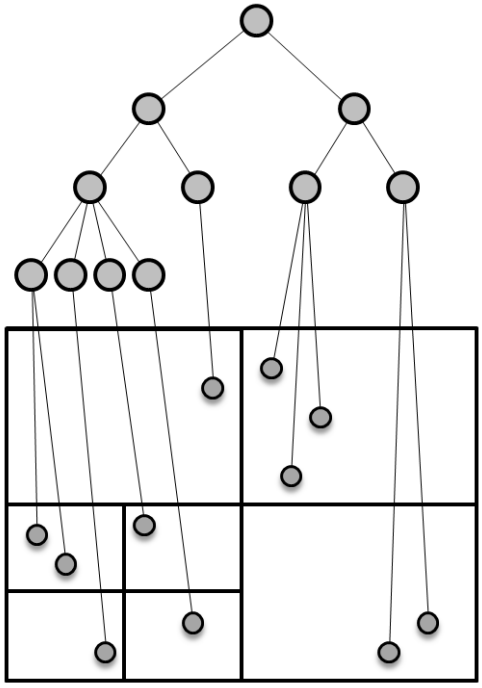

Figure 4.5: Partitioning of the Virtual World - This example shows a partitioning of a virtual world based on a coordinator quad-tree.

Coordinators which are at the bottom of the quad-tree are called *leaf coordinators*.

The function of the leaf coordinators in the quad-tree remains identical to the ones described in Section 4.2. However, it is now possible for an AoE to be not just outside of a peer's AoE, but also outside of the peer's zone. In this case, the update message is propagated up the quad-tree hierarchy of coordinators until it reaches a coordinator which covers the target zone(s) of the AoE. This coordinator then propagates the message down the hierarchy until it reaches the appropriate leaf coordinators, which then deliver it to the target peers.

In the following we will look in detail at the the split and join operations, as well as the update propagation along the coordinator hierarchy.

4.3.1 Zone Split

Algorithm 4 Zone Split Algorithm

```
1: procedure SplitZone(listOfPeers)
2:     if (listOfPeers.count > splitThreshold) then
3:         newCoordinators = SelectCoordinators(4)
4:         childZones[4] = quadTreeSplit(zone)
5:         for (i = 0; i++; i < 4) do
6:             assign newCoordinator[i] to childZones[i]
7:             for all (peers p in listOfPeers) do
8:                 if (p.position ∈ childZones[i]) then
9:                     assign P to newCoordinator[i];
10:                end if
11:            end for
12:        end for
13:    end if
14: end procedure
```

The zone split operation is shown in Algorithm 4. This algorithm is executed each time a peer enters a zone and connects to the coordinator. If the total number of peers exceeds a pre-determined threshold *splitTreshold*, then the split is performed. This threshold is set based on the available resources of the coordinator, i.e. how many peers a coordinator is able to manage.

First, we select four new coordinators. We will discuss how to select suitable peers in detail in Section 4.4. For now we, can simply select four random peers from the zone.

Then, the coordinator's zone is split into four equal-sized rectangular sub-zones. These new zones are the child zones of the coordinator and each of the four coordinators is assigned responsibility for one of them. Finally, each peer connected to the coordinator is informed that it needs to switch coordinators. This is done based on the location of the peers. Each peer is assigned to the coordinator of the child zone in which it is located. The peer connects to the new coordinator and disconnects from the old one. Once all peers have disconnected from the old coordinator, the split operation is complete.

4.3.2 Zone Join

In order to perform zone join operations, each leaf coordinator regularly checks whether the number of peers in its zone has dropped below a certain threshold defined as *joinThreshold*. It does so by running the procedure *CheckForZoneJoin*() in Algorithm 5 each time a peer enters or leaves its zone. The threshold is determined as half the *splitThreshold* of the parent coordinator divided by four. This ensures that a zone join is only performed when the parent coordinator has enough resources to handle all peers currently connected to the child coordinators. It also ensures that it has enough resources to handle additional peers in the zone, in order to avoid the zone getting split again in the near future.

If the number of peers in the zone drops below the threshold, the parent coordinator is sent a `zone_join_ready message`. This message informs it that this child coordinator has few enough peers to justify a zone join operation. As all four child coordinators must be ready for a zone join before it can be performed, each child coordinator also checks whether the threshold has been exceeded again. If so, it sends a `zone_join_not_ready` message to the parent coordinator, thus signalling that it once again has enough peers to prevent a zone join.

The parent coordinator executes Procedure *JoinZoneParent*() in Algorithm 5 each time it receives a `zone_join_ready` message from one of its child coordinators. First it checks if it has received a `zone_join_ready` message from all child coordinators. Only if this is the case, does it perform a zone join. To do so, it first sends a `zone_join message` to all child coordinators. It then waits for a `zone_join_confirmed` message from the child coordinators.

When a `zone_join` message is received by a child coordinator, it executes Procedure JoinZoneChild() from Algorithm 5. This procedure sends a message to all peers in the zone, informing them that their new coordinator is the child coordinators parent coordinator. The peers then connect to their new parent coordinator and disconnect from their current one. Once all peers have disconnected, the child coordinator sends a join confirmed message to their parent coordinator and seizes to operate as a coordinator. At this point they revert to being a peer only.

The parent coordinator waits until it has received a `join_confirmed` message from all peers and then disconnects from the child coordinators. At this point it is a leaf coordinator and the zone join operation is complete.

Algorithm 5 Zone Join Algorithm

```
1: procedure CheckForZoneJoin(listOfPeers)
2:     while (isCoordinator) do
3:         if (listOfPeers.count < joinThreshold) then
4:             send zone_join_ready message to parent coordinator
5:         else if (listOfPeers.count > joinThreshold) then
6:             send zone_join_not_ready message to parent coordinator
7:         end if
8:     end while
9: end procedure
10: procedure JoinZoneParent
11:     if (all childCoordinators ¡ joinThreshold) then
12:         send zone_join message to all child coordinators
13:         wait for zone_join_confirmed message from all child coordinators
14:         close connections to child coordinators
15:     end if
16: end procedure
17: procedure JoinZoneChild(listOfPeers)
18:     for all (peers p in listofPeers) do
19:         send parent coordinator as new coordinator to p
20:     end for
21:     send join_confirmed message to parent coordinator
22:     stop coordinator operations
23: end procedure
```

4.3.3 Update Propagation

We will now present updated algorithms for update propagation, which take the hierarchical overlay into account. Algorithm 6 shows how a coordinator in the hierarchical overlay handles an incoming update message.

First, we check if the update's AoE intersects with the zone which the coordinator is responsible for. If this is the case, the coordinator must propagate the update down the coordinator quad-tree, as at least some recipients of the update are in its zone. To do so, it first checks if it is a leaf in the coordinator quad-tree, i.e. whether it has child coordinators or has direct overlay connections to the peers in its zone. If it is a leaf, it can execute Algorithm 1 and deliver the update to all interested peers. If it is not a leaf, it must pass the update down the tree to all child coordinators whose zones the update's AoE intersects with. It therefore checks for the intersection of the AoE with the four zones and delivers the update to the appropriate child coordinators. These child coordinators in turn execute Algorithm 6.

Finally, the coordinator must check if there are potential recipients for the update, which are outside of its zone. In this case, it must propagate the update up the quad-tree overlay to its parent coordinator. It therefore checks whether the update's AoE intersects with the virtual world (minus its own zone). If it does, it delivers the update to its parent coordinator, which in turn executes Algorithm 6 as well.

Note that an update can be propagated both down and up the coordinator overlay, e.g., when an update's AoE intersects the border of a coordinator's zone. In this case, the update is first propagated down the quad-tree to the child coordinator(s) responsible for the appropriate zone(s). The same update is also propagated up to the parent coordinator which propagates it onrward to the coordinator(s) responsible for the zones which are outside of this coordinator's zone.

Finally, we present an updated version of Algorithm 3, which incorporates Algorithm 6. The new algorithm is shown as Algorithm 7. Instead of directly executing Algorithm 1 when a coordinator receives an update, it executes Algorithm 7 instead. This allows it to use the quad-tree overlay to propagate the message to all interested recipients.

When a new update is generated, the peer now executes Algorithm 7. The peer thus first delivers the update to all interested peers in its AoP (using Algorithm 1).

Algorithm 6 Coordinator Update Propagation Algorithm

```
1: procedure PropagateUpdate(update, listOfPeers)
2:     if u.AoE ∩ (zone \ u.AoP) then
3:         if (isLeaf) then
4:             execute Algorithm 1
5:         else
6:             for all (coordinators cC in childCoordinators) do
7:                 if (u.AoE ∩ cC.zone) then
8:                     deliver update to cC
9:                     cC executes Algorithm 6
10:                end if
11:            end for
12:        end if
13:    end if
14:    if u.AoE ∩ (world \ zone) then
15:        deliver update to parent coordinator
16:        parent coordinator executes Algorithm 6
17:    end if
18: end procedure
```

Algorithm 7 Update Propagation Algorithm

```
1: procedure PropagateUpdate(update)
2:     if (u.AoE ⊂ (p.AoP − maxRadiusOf(AoI))) then
3:         local peer executes Algorithm 1
4:     else if (u.AoE ∩ p.AoP) then
5:         deliver update to coordinator
6:         coordinator executes Algorithm 6
7:     else
8:         local peer executes Algorithm 1
9:         deliver update do coordinator
10:        coordinator executes Algorithm 6
11:    end if
12: end procedure
```

If necessary, it passes the update on to its coordinator. This coordinator delivers the update to all interested peers inside its zone to which the update has not yet been delivered (also using Algorithm 1). If necessary, the coordinator can then pass the update further up the coordinator overlay, where the coordinators continue to propagate the update up and down the overlay until it is delivered to all recipients (using Algorithm 7).

4.3.4 Zone Handover

We must also consider the situation when a peer moves from one zone to another. In this case, it must switch coordinators. This switch is initiated by the coordinator, when it notices that a peer's position is outside its own zone. It uses the peer's position to route a handover message with the peer's address to a coordinator responsible for the peer's coordinates. This coordinator then opens a connection to the peer and informs it that it is now part of a new zone with a new coordinator. The peer then closes the connection to its former coordinator and the hand-over is complete.

4.3.5 Routing Examples

We will now give two examples for routing in the combined horizontal and vertical update propagation overlay. In Example 1 (given in Figure 4.6), peer P1 generates an event E1. As shown, the AoE of this event covers peers P2 and P3 and thus their AoIs. Note that the AoIs of AoPs of P1 and P2 are not shown, and that we assume no intersection of E1's AoE with any other peer's AoI.

P1 must now deliver the corresponding update to both P1 and P2. However, P1 only has enough resources to directly communicate with one neighbour peer, and its AoP therefore only contains P2, but not P3. As shown, P1 can directly deliver the update to P2. In order to deliver it to P3, it has to forward it to its coordinator (C1).

When C1 receives the update, it checks which of its peers are interested in the update. As P3 is the only interested peer which has not yet received the update, C1 delivers it to P3. C1 also checks whether the AoE corresponding to the update intersects with any part of the virtual world which C1 is not responsible for. This is not the case, however, and update propagation for event E1 is complete. Note that P1 directly delivers the update to as many peers as its resources permit. It then uses the coordinator to deliver the update to the remaining target peer.

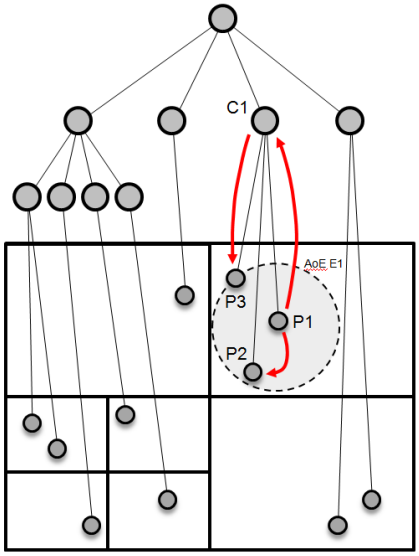

Figure 4.6: Routing Example 1

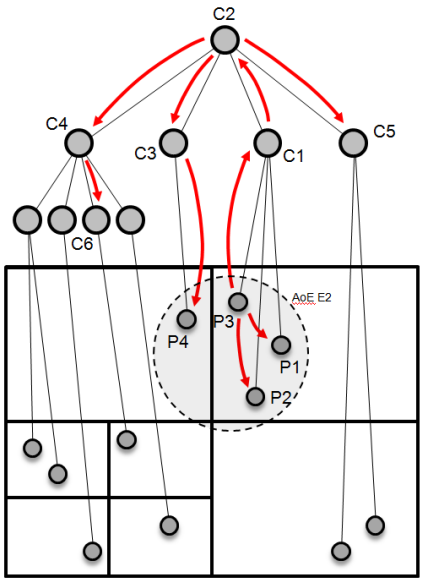

Figure 4.7: Routing Example 2

Example 2 (shown in Figure 4.7) is slighty more complex. Peer P2 generates an update E2. Once again, the corresponding AoE is depicted. Not that this AoE covers at least a small part of three different zones. In this case, peer P2 has sufficient resources to have all other peers in the zone within its AoP. It can thus directly deliver the update corresponding to the event E2 to peers P1 and P3. However, as the AoE extends beyond the borders of the zone, P2 also forwards the update to its coordinator C1. C1 does not need to deliver the update to any of its peers, as all interested peers in its zone have already received it. It therefore forwards it up the quad-tree to coordinator C2.

C2 cannot directly deliver the update to any peers as it is not a leaf coordinator. C2 thus checks if it needs to forward the update up or down the quad-tree. As it is the root coordinator, it is responsible for all zone intersecting with the update's AoE. Therefore the update must only be forwarded down the quad-tree. It covers all of the child zones of C2 and the update is therefore sent to all child coordinators, with the exception of C1 (which is the origin of the update).

The child coordinators each handle the update depending on whether they are a leaf coordinator or not. C5 is a leaf coordinator and checks whether any of its peers' AoI intersects with the updates AoE. This is not the case, and it can discard the update. C3 is also a leaf coordinator and performs the same check. It detects that the update's AoE

and peer P4's AoI intersect and thus delivers the update to P4. Finally, coordinator C4 is not a leaf coordinator. It thus checks whether the update's AoE intersects with any of its child zones. This is the case only for the zone of coordinator C6. C4 thus forwards the update to C6. Similarly to C5, coordinator C6 can discard the update, as the AoE does not intersect with any of its peers' AoI.

The update has now been delivered to all interested peers and update propagation for event E2 is complete.

4.4 Coordinator Election

While our coordinator overlay allows us to distribute the coordinator load among a number of peers, it makes the assumption that either all peers are suitable as coordinators or that suitable coordinators can be easily selected. This is not the case, however.

First, coordinators experience higher load than other peers in the system. This is due to the fact that they perform their coordinator functions in addition to being a participant in the system. Performing AoP calculations, zone operations and update propagation in the hierarchical overlay thus requires additional resources (such as CPU or bandwidth) over those already required for the update propagation via their AoPs.

As the peers in the system exhibit heterogeneity regarding their available resources, some peers are more suitable as coordinators than others. If a coordinator does not have sufficient resources to perform its tasks, the operation of the system can be become impaired. While unsuitable coordinators can be replaced by new ones, thus preventing system failure, this replacement results in additional overhead. In contrast, when only suitable peers are selected, we can make use of the resource heterogeneity of the peers, thus providing better performance and stability for the system and fulfilling our self-organization requirement.

There are two challenges when selecting suitable peers. First, we must determine what constitutes a suitable peer. In Section 4.4.1, we will therefore analyse which resources are relevant for our system and propose a simple metric to determine the suitability of a peer as coordinator. Second, once we can determine whether a peer is suitable or not, we must select suitable peers from all peers in the system. Due to the large number of participants it is not possible to simply query all peers in the system for their suitability score or use a traditional election algorithm. Instead, we present an election algorithm which samples a subset of peers using our existing vertical update propagation overlay in Section 4.4.2.

4.4.1 Suitability Metric

We have previously performed a thorough analysis of potential parameters for a suitability metric in [127]. Based on this analysis we have identified three types of resources which are particularly relevant to coordinator suitability in our approach: available

CPU resources, stability and available bandwidth. The higher its available CPU resources, stability, available bandwidth are, the more suitable a peer is as coordinator. We will first discuss these three parameters in detail and then present our suitability metric.

We consider three properties when analysing these parameters: cost, fluctuation and reliability [127]. Determining the current value of a utility parameter is associated with overhead. This means that determining the utility of a peer uses the resources of the system and comes with a *cost* to its performance. Due to the nature of our system, i.e. due to regular split and join operations, coordinator selection is a regular occurrence. It is therefore important to consider the cost of determining each parameter.

All parameters have a certain *fluctation* over time, i.e. their value does not remain the same. As a consequence, each value has to be measured repeatedly. There are three aspects that have to be considered when looking at the fluctuation of a parameter: the value's variance, the frequency of the value's changes and the magnitude of the value's changes. Variance indicates how much the value of a parameter can deviate from the mean. Consider the available bandwidth of a system. This value can vary significantly, depending on the amount of bandwidth used by all applications and systems utilizing the peer's Internet access. On the other hand, the number of a system's CPU resources has no variance in practice as CPUs are rarely if ever changed. Finally, the frequency of changes indicates how often a parameter's value changes. Frequent changes can necessitate regular re-measurements of a value, particularly if the parameter's variance is high. The magnitude of changes indicates how large the individual changes to the parameter's value can be. Some parameters may change frequently, but each individual change of the value tends to be small.

Not every measurement of a parameter's value necessarily returns a correct result. While some values, such as remaining hard disk space can be determined exactly, measuring others relies on methods that return imprecise result. The *reliability* of a measured value is therefore an important consideration when determining a peer's suitability as coordinator.

4.4.1.1 CPU

The CPU performance of a peer is a key parameter, as the tasks performed by coordinators require the execution of various algorithms which require available CPU resources.

The most CPU intensive task coordinators perform in our system is the calculation of their peers' AoPs. As this algorithm runs in $O(n^2)$, CPU can be a bottleneck for the number of peers in a zone, particularly since a coordinator is also expected to perform the duties of a peer and run the MMVE client software.

Available CPU resources may be limited even on powerful systems, as MMVEs place high load on the system through calculation of physics, artificial intelligence or real-time 3D graphics. This makes both the CPU hardware (e.g., number of cores) and the current CPU load relevant parameters.

The cost of determining both CPU hardware and current CPU load is low, as these values can usually be determined by a simple request to the operating system. Consequently, reliability is high. In theory CPU hardware has both high variance and a high magnitude of change. The value only changes when the CPU of a system is replaced, which commonly happens in order to install a more powerful unit, thus resulting in high magnitude and variance. In practice however, the CPU is rarely changed. This means the frequency of changes is so low that fluctuation can be ignored in practice. The current CPU load on the other hand has high variance, high magnitude and high frequency of changes. While the MMVE itself can be expected to generate consistent CPU load, the value can fluctuate due to other software running on the system.

4.4.1.2 Stability

Replacing coordinators is associated with overhead (e.g., from an election to find a replacement coordinator). As such it is desirable for a peer to remain coordinator for long periods of time. Using information on the user's past and current behaviour, it is possible to make predictions about her session time [128]. For example, some users have longer average session times than others. In addition, users which have just logged into the system are more likely to leave the system soon, than users which have already spent a certain time in the system. Peers which tend to leave the system gracefully, i.e. which shut down cooperatively, are also preferable over peers which regularly leave the system non-gracefully, e.g., due to system crashes. This is due to the fact that a non-graceful exit from the system can increase the overhead of replacing the coordinator, or lead to degraded performance of the system.

This type of behaviour is referred to as stability. The more stable a peer is, the more likely it is to remain in the system for a long period of time and leave it gracefully afterwards. Stability is a parameter with low reliability. Even when a long session time is predicted for a particular peer, it may still leave the system at any time.

As the value of this parameter is based mostly on a history of previous behaviour, this value adapts slowly to the changing behaviour of the user. As such the magnitude of changes is low. As the value changes only when new information is added to the history, the frequency of changes of also low. Variance can be large however, as a user's behaviour changes over time. Cost is low, because session time can be measured with minimal overhead.

4.4.1.3 Available Bandwidth

In our system coordinators incur a communication overhead compared to to operating as a standard peer. If they are a leaf coordinator, they regularly receive position updates from all peers in their zone. They also have to regularly send out updated AoP information to their peers. In addition, all coordinators have to forward updates in the quad-tree. Note that this communication effort is in addition to the existing bandwidth usage they incur in their role as a peer.

This makes bandwidth one of the key parameters for peer utility. There are three aspects that need to considered, when looking at bandwidth: a) maximum bandwidth b) overall available bandwidth c) available bandwidth to a specific peer. For each of these aspects, upstream and downstream bandwidth need to be considered separately, as bandwidth is often asymmetrical, with upstream bandwidth usually being the more limited resource.

The maximum bandwidth is the theoretical maximum bandwidth available to the peer. It is determined by the type of Internet connection of the peer's system. Due to several factors, such as sharing the connection with other systems, bandwidth usage by other applications on the peer's system and under-provisioning by the Internet service provider, the maximum bandwidth cannot always be achieved.

Available bandwidth refers to the actual bandwidth that is currently available to the peer for coordinator functions. The available bandwidth is the maximum bandwidth minus all currently active bandwidth reducing factors. Note that this includes the bandwidth used by the peer for non-coordinator related functions in the MMVE.

It should be noted, that while a peer may have high available bandwidth, it does not follow that this bandwidth is necessarily available to all other peers in the system. The actual data transfer rate from one peer to another depends on the available bandwidth of both peers as well as any possible bottlenecks on the network route between the to peers. As such, even if a peer has high available bandwidth it may still have a low utility with regard to bandwidth, if its position in the network topology means that data sent to or received by it passes through a bandwidth bottleneck. It can therefore be necessary to consider the actual available bandwidth to a specific peer.

When considering bandwidth as a parameter, it is important to consider upstream and downstream bandwidth separately. Depending on the coordinator's functions, incoming and outgoing traffic requirements may be different and many consumer Internet connections have asymmetric bandwidth, with significantly lower maximum upstream bandwidth than maximum downstream bandwidth. High bandwidth usage in one direction can also negatively affect the available bandwidth in the other direction, which must be taken into account when determining available bandwidth.

The cost of determining maximum or available bandwidth can be high, as some active bandwidth measuring algorithms send out the maximum possible bandwidth for a period of time. Despite this, based on various factors such as network bottlenecks or a shared Internet access, the reliability of these algorithms is generally low.

While the maximum bandwidth only changes, when the user switches his type of Internet access, the available bandwidth can change frequently with large magnitude of changes, due to other applications or systems using the same Internet access. If these factors are present, variance is also high.

4.4.1.4 Metric

While we require these parameters in order to determine the suitability of a peer, we can only determine the CPU power and stability with some reliability and at low cost.

To determine the available CPU resources so, we retrieve information on the current CPU use ($avCpu$) from the operating system and use a history function to deal with the high fluctuation of the result. In addition, we prefer peers with more CPU cores, by weighting the result with the number of CPU cores ($noCores$). We normalize the result to a range of [0-1] using the maximum value for the available bandwidth $maxAvCpu$. The available CPU resources are thus given as:

$$avCpu = \frac{(\alpha * noCores) * \beta * avCpu + (1 - \beta) * avCpu}{maxAvCpu} \tag{4.1}$$

To determine stability, we use a weighted history of a peer's last session length ($sLength$). In addition, we downgrade peers which have not yet exceeded a certain minimum session length. If the minimum session time length has been exceeded, $minLenExceed$ equals 1, otherwise it is 0.5. Again we normalize the result to a range of [0-1]. The stability of the peer is thus given as:

$$stability = minLenExceed * \frac{\gamma * sLength + (1 - \gamma) * sLength}{maxLength} \tag{4.2}$$

While this simple approach has relatively low cost, it takes several sessions for the reliability of the value to become acceptable. Given the usage pattern of MMVEs it can therefore take days before we can get a reliable value for any given peer.

Nevertheless, this provides us with simple solutions for available CPU resources and stability. However, no simple or existing solution for determining available bandwidth satisfies our requirements (see Chapter 3.3). A solution which suits our needs would have to be low-cost, provide results quickly and deal well with high fluctuation. We have therefore developed our own approach to determine $avBw$ (see Chapter 4.5).

To determine overall suitability, we first check if the peer is already a coordinator. If so, it is less suitable to serve as coordinator again, as its available resources should be conserved in case its load increases. If the peer is a coordinator, $notC$ is equal to 0, otherwise it is 1. Second, we check if any of the three values (available CPU, stability and available bandwidth) are below a respective minimum threshold. If the respective thresholds are exceeded, $cThresh$, $sThresh$ and $bThresh$ are 1, otherwise they are 0. A peer must have sufficient available resources in all three categories to be suitable as coordinator. Even if a peer has high stability and high available CPU resources, it is not suitable if the available bandwidth is below a minimum threshold. We can thus use $limitSuit$ to reduce a peers overall suitability by 50%, in case it already serves as coordinator or does not meet the minimum requirements in each category:

$$limitSuit = 0.5 + (notC * cThresh * sThresh * bThresh * 0.5) \tag{4.3}$$

The suitability of a peer is thus given as follows. It should be noted that the thresholds ($cThresh$, $sThresh$, $bThresh$) and weights (α to η) are application-dependant and

should be set by the developer of the specific MMVE. As an example, an MMVE with larger-than-average updates will utilize more bandwidth and should thus have a higher minimum threshold as well as a higher value for weight η. In general we suggest that bandwidth is the most important resource and stability the least important one.

$$suitability = limitSuit * (\epsilon * avCpu) + (\zeta * stability) + (\eta * avBw) \tag{4.4}$$

4.4.2 Coordinator Election Algorithm

We observe that all leaf coordinators receive regular information from their peers in the form of position updates. If we include the suitability score in the position update messages, sending additional information about the suitability of the peers to the coordinator generates no additional message overhead. As the score is a small integer, there is also limited additional bandwidth used by this. Finally, given that the suitability score is also a more stable value than the position and can thus be sent to the coordinator at a much lower frequency, each coordinator can have full knowledge of its peers' suitability with little overhead.

Using this approach it is now possible for each leaf coordinator to select suitable coordinators by selecting the candidates with the highest suitability scores among its peers. This simple solution has several disadvantages however. First, while zone splits occur at leaf coordinators, we may also need to select coordinators at non-leaf coordinators in the overlay. For example, this is the case when a coordinator wants to leave the system and needs to select a replacement. Second, the subset of peers considered by this approach is limited by an individual zone. Repeated coordinator selections (e.g. after repeated splits) may thus occur from a depleted pool of candidates, i.e. one where the most suitable peers have already been chosen. While new peers enter the system regularly, it would be preferable to have all peers in the system as the pool of potential candidates. Finally, selection may be from a very small pool of candidates. If a zone only has a few peers in it, selecting the most suitable among them may not result in any good candidates, even though there are many good ones in the system. On the other hand it is not feasible to collect information about all peers in the system. Due to the potentially large number of peers and coordinators in the system, aggregating suitability information from all coordinators would result in prohibitive overhead.

The basic concept of our coordinator election is thus to sample a subset of peers for each election. To reduce overhead we are using our existing update propagation overlay to perform this sampling. Instead of aggregating suitability information from all zones in the overlay, we are randomly selecting a group of zones. The number of zones in this group determines the size of the candidate pool. The higher it is, the more candidates we can expect to consider, with the downside of higher overhead.

As no peer has a full view of the world, we cannot directly select these zone. We therefore we generate coordinates in the virtual world. Each of these coordinates corresponds to a location in a zone and thus represents the zone responsible for that coordinate. Generating z coordinates thus means that we sample a set of at most z zones. It should be noted that this does not guarantee that we sample exactly z zones, as two or more coordinates can be located in the same zone. In particular this is the case when the virtual world is divided into less than z zones.

Algorithm 8 shows our coordinator election algorithm, which takes four parameters: $numberOfCoordinators$ is the number of coordinator we wish to elect (e.g., four coordinators are needed during a zone split). $suitThreshold$ is the suitability threshold above which a peer is considered suitable as coordinator. This threshold is set by the application developer, depending on the specific application's requirements. $noOfSamples$ is the number of sample coordinates we generate during each election. Finally, $peers$ is a list of peers connected to the coordinator which initiates the election. Note that the coordinator has full suitability information about these peers.

First, we consider the special case that there is only one coordinator in the system (e.g., during system start-up or when very few peers are in the system). This coordinator is thus both the root and a leaf coordinator. In this case all coordinates would be inside this coordinator's zone and an election based on sampling would return the same results as the simple approach detailed above. Thus, a root coordinator without child coordinators never initiates a sampling election and instead only searches for candidates locally. In addition, it also drops any suitability threshold, as the best candidates available to it are automatically the best in the system. This allows basic operation of the system, even when few suitable coordinators are available (e.g., during system startup).

Even in cases where the coordinator is not the only one in the system, we first use this simple solution. Searching locally first allows us to avoid some full elections

Algorithm 8 Coordinator Election Algorithm

```
1: procedure ElectCoordinators(numberOfCoordinators, suitThreshold, noOfSamples, peers)
2:     peers.SortBySuitability()
3:     coordinates = GetPoissonDiskCoordinates(noOfSamples)
4:     if (isLeafCoordinator && parentCoordinator == null) then
5:         for (i = 0 → numberOfCoordinators − 1) do
6:             candidates.Add(peers[i])
7:         end for
8:     else if (isLeafCoordinator) then
9:         for all (Peers p in peers) do
10:            if (p.suitability > suitThreshold + (1 − suitThreshold)/2) then
11:                candidates.Add(p)
12:            end if
13:        end for
14:        if (candidates.Count >= numberOfCoordinators) then
15:            for all (Coordinates c in coordinates) do
16:                request = new ElectionMessage(c, localAddress)
17:                propagate request using Algorithm 10
18:            end for
19:            wait for noOfSamples response messages
20:            candidates.concat(response.candidates)
21:        end if
22:    else
23:        for all (Coordinates c in coordinates) do
24:            request = new ElectionMessage(c, localAddress)
25:            propagate request using Algorithm 10
26:        end for
27:        wait for noOfSamples response messages
28:        candidates.concat(response.candidates)
29:    end if
30:    candidates.SortBySuitability()
31:    for (int i = 0 → numberOfCoordinators − 1) do
32:        result.Add(candidates[i])
33:    end for
34:    return result
35: end procedure
```

(and the associated overhead) when the coordinator has sufficient suitable candidates available in its zone. Thus each coordinator first checks whether there are sufficient suitable peers available in its own zone. If this is the case, it uses these peers as candidates instead of initiating a full election. Note that this could lead to a reduction of the average suitability of selected coordinators. While there may be sufficient suitable coordinators in the zone, a normal election (among a potentially much larger pool of candidates) may provide better results. We therefore use a higher suitability threshold in this initial step. Only leaf coordinators perform this step, as other peers do not have peers connected to them.

If the local search fails or cannot be initiated, we start the standard sampling election by creating $noOfSamples$ sampling coordinates. Instead of a fully random selection we are using poisson-disk sampling[129]. With poisson-disk sampling all samples are randomly placed, but no two points are closer to each other than a minimum distance. This allows us to reduce the number of requests which are sent to the same coordinator in an election. Specifically we use the efficient (linear time) implementation by Bridson [130].

After creating the coordinates, we can build an election request message for each coordinate. This message also contains the address of the requesting coordinator. We then send it to the target coordinator (i.e. the one responsible for the zone which contains the coordinate) using our existing update propagation overlay. This is process is essentially identical to our normal update propagation. The coordinate thus represents an AoE with a diameter equal to zero and the request is propagated via the vertical update propagation overlay until it reaches the responsible leaf coordinator. Note that a point-sized AoE means that it is inside one and only one zone. Algorithm 10 shows the modified propagation algorithm.

In contrast to the normal update propagation, a request message is not delivered to a peer like an update, but handled at the coordinator. Algorithm 9 shows how a request message is handled at this coordinator. It uses the suitability information it has received from its peers to create an election response message. This message contains a list of all suitable peers in its zone (i.e. all peers whose suitability is higher than $suitThreshold$) as well as the address of the requesting coordinator. As suitability is application-dependant, this threshold can be set by the application developer. Note

that in order to avoid sending back an excessively large number of suitable candidates, we can also set a maximum size for this list.

Algorithm 9 Handle Request/Response Message

```
1: procedure HandleElectionRequest(request, suitThreshold, peers)
2:     for all (Peers p in peers) do
3:         if (p.suitability > suitThreshold) then
4:             candidates.Add(p)
5:         end if
6:     end for
7:     response = newResponseMessage(candidates, request.address)
8:     execute Algorithm 10
9: end procedure
10: procedure HandleElectionResponse(response)
11:     for all (Peers p in response.candidates) do
12:         if (!candidates.Contains(p)) then
13:             candidates.Add(p)
14:         end if
15:     end for
16: end procedure
```

The election response message is then propagated back the coordinator which requested it, once again using the vertical update propagation overlay and Algorithm 10. Note however, that the requesting coordinator may not necessarily be a leaf coordinator. Before propagating an election response message, each coordinator in the overlay thus checks the address in the message to see whether it was the requesting coordinator. If it is, it handles the response message (see Algorithm 9).

Once a coordinator has received a response for all election requests, it creates a sorted list of all candidate peers (see Algorithm 9). From this list it can then select the top candidates (e.g., the top four candidates are selected as new leaf coordinators in a zone split). It may also be the case that coordinators which receive an election request message do not have any suitable peers in their zones. To avoid a situation where no candidates are returned, these coordinators always include least the four most suited peers in their zone, even if these peers are under the suitability threshold.

To allow multiple elections at once, we also include a GUID into each request and

response message. This allows the requesting coordinator to accurately assign incoming response messages to its respective election. We can also slightly reduce overhead using the GUID, as a zone which receives multiple request under one GUID can send back empty candidate lists for all but the first request, and avoid sending the same candidate list multiple times. For clarity, we have omitted GUID handling from the algorithm.

Algorithm 10 Coordinator Election Propagation Algorithm

```
1: procedure PropagateElectionMessage(message)
2:     if (message.coordinate ∩ zone) then
3:         if (message.addr == localAddress && message.type == "response")
   then
4:             HandleElectionResponse(message)
5:         else if (isLeaf && message.type = "request") then
6:             HandleElectionRequest(message)
7:         else
8:             for all (coordinators cC in childCoordinators) do
9:                 if (message.coordinate ∩ cC.zone) then
10:                    deliver message to cC
11:                    cC executes Algorithm 10
12:                end if
13:            end for
14:        end if
15:    end if
16:    if (message.coordinate ∩ (world \ zone)) then
17:        deliver update to parent coordinator
18:        parent coordinator executes Algorithm 10
19:    end if
20: end procedure
```

Figure 4.8 shows an example of an election. In this example, Coordinator S starts the election in order to split its zone. It is not the only coordinator in the system, so the first special case does not apply to it. It first attempts to find a coordinator locally. The suitability threshold is set to 0.7, so out of the four peers in its zone, only one is suitable. As it is looking for four new coordinators, this local search fails.

It then generates four coordinates C1 (80,100), C2 (90,210), C3 (90,230), C4 (270,280) and creates an election request message for each coordinate and propagates it through

the vertical update propagation overlay. C1 is inside coordinator E2's zone. E2 thus selects all suitable coordinators from its zone and creates an election response message. However, only one peer has a suitability above the threshold. Thus it also includes the other peer in its zone, even though its suitability is below the threshold. The response message is propagated back to node S, which adds the two candidates with the suitability scores of 0.7 and 0.1 to its list of candidates. Similarly, coordinate C4 lies in coordinator E3's zone. E3 sends back a list of candidates with the suitability scores of 0.9, 0.7 and 0.4.

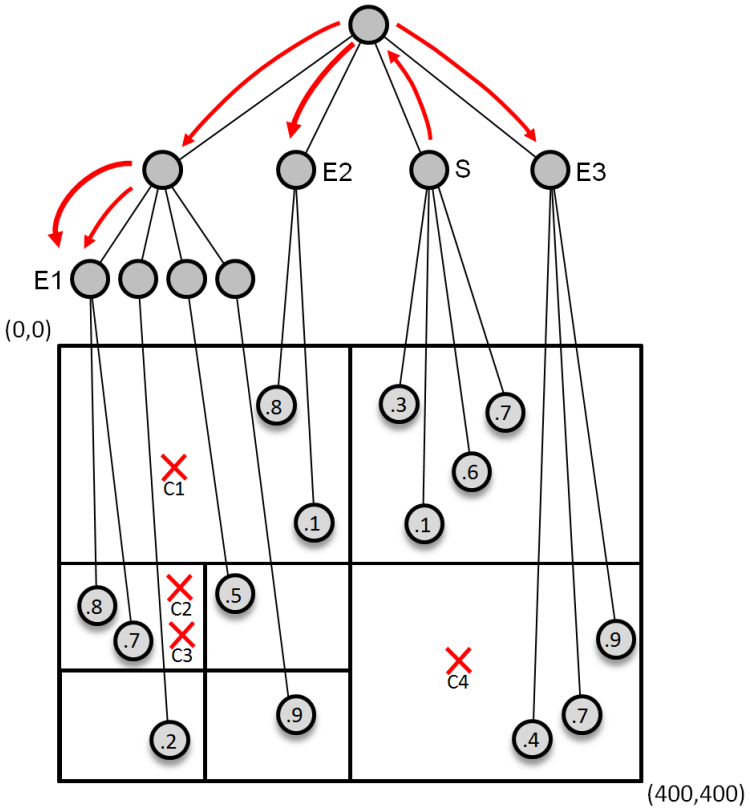

Figure 4.8: Election Example - Node S starts the election and generates four sample coordinates.

Both coordinates C1 and C2 are inside coordinator E1's zone. Thus E1 receives two request messages. It responds to the first one with two candidates (0.8 and 0.7). Using the GUID in the request messages, it can send back an empty list in response to the second one.

Coordinator S now has a list of seven candidates. By sorting the list by suitability and selecting the first four elements of the list, it finds four suitable coordinator candidates and can initiate the zone split.

4.4.3 Bootstrapping

Finally, we must deal with the issue of bootstrapping, i.e., how a peer enters the system and becomes part of the update propagation overlay.

As mentioned in Chapter 2, the peers@play prototype offers bootstrapping services. This bootstrapping system allows a peer which wants to enter the system access to the DHT-based storage system of the peers@play system. In order to participate in the MMVE, the peer must then become part of our update propagation overlay. To facilitate this, we save the Link Layer address of the root coordinator of the hierarchical overlay in the DHT, using the ID (i.e. the name) of the virtual world as a key. Note that this allows a single DHT to support several parallel virtual worlds.

When the peers has received the address of the root coordinator from the peer, it sends a request to enter the virtual world to the root coordinator. This request includes the location in the virtual world where the peer's avatar wants to enter it. Using this location and our update propagation algorithm, the root node then routes this message to the leaf coordinator which is responsible for the zone which contains the location. The leaf coordinator then connects to the peer, which can start using the system.

Note that it is possible to receive outdated root coordinator information when trying to enter the system. There are two possible scenarios for this. First, the root coordinator may have left the system after the peer has read the information from the DHT, but before it has requested to enter the system. Thus, the peer will fail to connect to the coordinator. In this case, the peer will simply try to connect again, after the connection failure or a time-out. Eventually it will succeed in connecting to the root coordinator. In the second scenario, the root coordinator is still part of the system, but no longer serves as coordinator, e.g. if it not longer has sufficient available resources to support the system. This is not a problem however, as the new peer can still connect to the former coordinator, which can forward the request. We thus require all peers in the system to accept connections from new peers and route their connection requests to the responsible coordinator.

A peer can now enter the system and will be connected to the coordinator responsible for its location. This does not work however, when there are no coordinators in the system, i.e. when the peer is the first peer which tries to enter the virtual world. Thus, any peer (regardless of its suitability score) is required to promote itself to the role of coordinator if the DHT does not contain a root coordinator. The first peer thus starts out as coordinator for the entire world. As more peers enter the world it may eventually be replaced by a more suitable peer. As peers leave the system, the last peer remaining will also end up serving as coordinator for the entire virtual world. When it leaves, it is required to remove the coordinator information from the DHT, thus allowing a new start of the system when a peer once again tries to enter it.

It cannot be guaranteed however, that a peer leaves the system gracefully. This includes the last peer. If the last peer in the system, leaves it without removing the information from the DHT (e.g. due to a system crash), then the virtual world could not be restarted. The first new peer to try to enter the world would repeatedly attempt and fail to connect to a peer which is no longer available. To prevent his, the root peer regularly saves his information to the DHT and includes a time-stamp. Thus, when a new peer tries to enter the system and fails to connect, it will check this time-stamp. If it is older than a certain pre-defined period of time, it will assume that it is the first and only peer and the system, promote itself to coordinator and store its own information in the DHT.

4.5 Available Bandwidth Estimation

We have previously assumed that we have knowledge about the available bandwidth of the peers in our system. We use this information to determine the AoPs of individual peers as well as in the utility function during coordinator election. However, this information is a priori unknown, making this assumption unrealistic.

While there is existing work on determining available bandwidth, this work focuses almost exclusively on the bandwidth along specific network paths, e.g. between two individual hosts [131] [132] [133] [134] [135]. However, the communication partners of the peers and the coordinators in our system are a priori unknown and the communication between individual partners is limited. This means these approaches are of limited value to us.

In contrast, we require information on how many communication partners an individual peer or coordinator can handle. This in turn allows us to determine the size of the AoP and select suitable peers as coordinators. In other words, we are interested in the overall bandwidth resources which a peer can contribute to the system.

A common solution to this problem is to let the user set the advertised speed of his internet connection in the application (e.g., [136]). This is insufficient for several reasons. First, many users aren't necessarily aware of the advertised speed of their connection. Even if they are, the common practice of under-provisioning by Internet Service Providers, can result in much lower available bandwidth than the advertised one [137]. Users may also intentionally set incorrect values in an attempt reduce the P2P systems usage of their resources, e.g., in an attempt at *free-riding* [138].

Even if the current available bandwidth has been determined, e.g. by using a server-based, link-level bandwidth test (e.g. [139][140]), this value can not be relied on. Again, under-provisioning can result in fluctuations of the available bandwidth, especially during peak-hours. Other software on the peer can use bandwidth, thus reducing the available bandwidth. Other systems in the same local network may also use up available bandwidth. Mobile peers may also change their internet connection. In short, the available bandwidth fluctuates, which makes a one-time application-level setting unsuitable for our purposes.

In addition, determining the available bandwidth is made challenging by the fact that in a fully distributed P2P system we cannot rely on the availability of a server

infrastructure as used by common bandwidth tests in the internet. Finally, as our system requires significant bandwidth to run, any bandwidth determination must not interfere with the operation of the system.

However, without accurate information on the available bandwidth, our approach will not be able to determine the correct size of the AoPs. This will result in higher load on the coordinators. Unfortunately we also cannot elect peers with sufficient bandwidth to handle this load, in this situation. An suitable bandwidth estimation system is thus critical to our approach.

We have therefore developed such a system [141] and will describe in the following. First we will outline the requirements we have for the system, followed by a detailed description of the estimation algorithm.

4.5.1 Requirements

We have determined three requirements for a bandwidth estimation which provides results which are useful to our system. These requirements are *Accuracy*, *Responsiveness* and *Minimal Interference* [127] [141].

- Accuracy: To be useful, the resulting estimate must be accurate. Note that the specific level of accuracy that is required depends on the application using the P2P system. Therefore, the estimation approach should be configurable for different accuracy levels. Inaccurate estimations can be distinguished into underestimation and overestimation. Underestimations report a total bandwidth that is lower than the actual value. Overestimations report a total bandwidth that is too large. While we want to eliminate both kinds of errors, we argue that overestimations are more severe than underestimations. Avoiding overestimations should therefore be prioritized. Otherwise, a peer might be rated too good and be overloaded with management functions. This could degrade system performance or even lead to system failure. Therefore, we accept a certain amount of underestimation if this reduces the probability of an overestimation.

- Responsiveness: The available bandwidth of a peer is not static. Depending on the peer's context, it can vary frequently, e.g. in case of mobile devices or multiple devices sharing an Internet connection. The bandwidth estimation approach should respond quickly to such variations and always provide a current estimation

of the available bandwidth. Note that – just as for accuracy – the specific level of required responsiveness is application dependent and should be configurable.

- Minimal Interference: While providing accurate and current information about the available bandwidth, the estimation approach should minimize its interference with the actual application. Otherwise, the estimation could e.g. consume a large percentage of the available bandwidth for its measurements. This should not be the case. When the application needs to send or receive data, the estimation approach should use as little bandwidth as possible and provide as much as possible to the application. On the other hand, if the application currently does not need the bandwidth, the estimation approach can take over and use it to perform measurements.

4.5.2 Overview and Design Rationale

There are two basic approaches to estimating available bandwidth, *passive observation* and *active measurement*. Passive observation refers to estimating the available bandwidth based solely on observing existing traffic on the system. In contrast, active measurement uses techniques which actively inject traffic into the system to determine the available bandwidth.

By observing existing traffic on the peer, it is possible to deduce some information about the available bandwidth. This approach has virtually no overhead, as information about the current traffic is a simple low-cost system call on most devices. This results in minimal interference with the P2P system and also allows to fulfil the requirement of Responsiveness, as we can regularly monitor the current traffic. Unfortunately this approach does not provide very high accuracy, in particular during sparse observation. If there is very little traffic to observe, we cannot expect to get high quality results.

Active measurement does allow for more accurate results, as it can generate the necessary traffic to allow good estimates of the available bandwidth. Unfortunately this traffic interferes with the operation of the P2P system by reducing the bandwidth available to it. In the worst case, an active measurement may even use all available bandwidth. This also means, it is difficult to provide responsive results. While a short-term interference may be tolerable, constantly measuring the available bandwidth could

in practice result in a situation where the system permanently reports no available bandwidth, as the measurements use it all up.

Our approach combines low-overhead passive observation of existing traffic with an active peer-based injection of traffic to improve accuracy. To reduce overhead, we only use traffic injection when the peer has idle resources or our confidence in the current estimation is too low. The system is configurable by applications to balance the achieved accuracy with the resulting overhead.

The basic idea of our estimation algorithm is as follows. To determine the available bandwidth, we need two pieces of information: the system's currently used bandwidth uBW and the system's current capacity C (i.e. the maximum available bandwidth). Given these values we can calculate the available bandwidth aBW as $aBW = C - uBW$. We can easily measure uBW, e.g. using system load information provided by the operating system. However, C is a priori unknown and cannot be determined directly. Therefore, the problem of estimating aBW can be reduced to estimating C.

To do so, we continuously determine estimates for a lower bound lB and an upper bound uB of C. The real value of C must be between these two bounds. lB is determined by the current (passive) measurement of uBW. uB is set using (active) traffic injections. To avoid overestimation, we add a decay function to calculate the progression of C. The decay function decreases C over time until a given threshold value is met. Then uB is recalculated using another active traffic injection.

Note that the system may have previous knowledge about C. This knowledge can originate from a user setting in our software, information from hardware, e.g., a DSL modem, or from previous measurements. Our approach can use this knowledge as a starting point for its estimations. However, we argue that these values are not reliable and thus cannot replace continuous estimation. As discussed before, users may simply not be aware of their Internet connection's capacity or even have incentives to misreport it [142]. The capacity of an Internet connection can also change, for example during peak hours due to under-provisioning by ISPs [137]. Finally, certain systems such as mobile devices change their type of internet connection regularly.

In the remainder of this chapter, we describe this approach in more detail. First, we present the basic capacity estimation algorithm. Then we describe how the threshold and the decay rate are computed. After that we show some enhancements to the basic

approach and discuss how traffic injections are performed. Finally combine all of these aspects and estimate the available bandwidth.

4.5.3 Capacity Estimation

In the following we explain our basic capacity estimation algorithm in more detail (see Algorithm 11). As described before, the algorithm depends on a decay function and a threshold value. The details of their computation (together with the parameters T and L) are described in the next chapters and omitted here.

Algorithm 11 Basic Capacity Estimation Algorithm

```
1:  procedure EstimateCapacity(init, T, L)
2:      uB = C = init; t = 1
3:      loop
4:          lB = usedBandwidth()
5:          if lB > uB then
6:              uB = lB
7:          end if
8:          thr = threshold(lB, uB, T)
9:          C = uBd−decay(t, L, uB, thr)
10:         t = t + 1
11:         if C ≤ thr then
12:             uB = injectTraffic()
13:             t = 1
14:         end if
15:     end loop
16: end procedure
```

The algorithm starts by initializing the upper bound uB and the estimated capacity C with the parameter *init*. This parameter is passed to the algorithm by the application and contains all previous knowledge about the network capacity. If no such knowledge is available, *init* can be set to zero. After that, a variable t is initialized to one. It is used to compute the correct decay rate later. The higher t, the larger the decay. Thus, the initial decay is minimal.

After the initialization, the algorithm loops over the following instructions. First, it sets the lower bound lB to the current network load of the system. Then, it checks

whether the current load is higher than the expected upper bound uB. If it is, uB is updated. Now, it computes the current threshold value thr. After that it recomputes the estimated capacity C. C is set to uB minus the current decay function value (decay(..)). Then, t is incremented to increase the decay in the next round. If the resulting capacity is equal or below the threshold, the algorithm measures uB again by initiating a traffic injection and setting uB as the highest traffic observed during the traffic injection. In addition, the algorithm resets the decay rate by setting t to zero. After that, the loop starts again.

This algorithm realizes the basic approach presented in the previous section. It continuously estimates C using passive traffic observation. In order to provide an accurate estimation for C the upper bound has to be determined by traffic injection. Right after an injection, uB is very accurate and we can set $C = uB$. The further our last traffic injection is in the past, the more likely it is that C might have changed, reducing our trust in its estimation. Thus, if we keep C at the upper bound, the likelihood of an inaccurate estimation keeps increasing.

The naïve solution for this is to constantly inject traffic and redetermine uB continuously. Clearly, this would produce prohibitive overhead, violating requirement R3. Thus we use our decay function and the threshold to model the ageing of C over time.

4.5.4 Threshold

The threshold thr is the value towards which C falls before a new active traffic injection is initiated. Clearly, thr must be between our two bounds lB and uB. However, choosing the correct value for thr is not trivial, as it influences the accuracy we can expect to get. If we select a high thr, i.e., one close to uB, we increase the risk of overestimation, as the reported C will be higher. On the other hand, if we select a low thr, i.e., one close to lB, we increase the risk of underestimation, as the reported C will generally be lower. The correct choice ultimately depends on the application. The less tolerable an overestimation is for a given application, the lower th should be. We therefore introduce a parameter T with a range from 0 to 1 and define the threshold (and thus the threshold(..) function) as follows:

$$thr = lB + (uB - lB) * T \tag{4.5}$$

Using T we can select where the threshold will be set between the two bounds. With $T = 0.5$, the threshold is exactly in the middle between lB and uB. With $T = 0$, the threshold is equal to lB. Note that our approach is designed to prefer underestimation even when the threshold is set very high, as the upper bound is selected in such a way that it can be an underestimation even at the moment of selection.

Note that thr is calculated dynamically for each point of time, using the most recent lB and uB. As lB is continuously remeasured, th can change constantly, getting larger and smaller with the rise and fall of lB.

4.5.5 Decay Function

After looking at the threshold, we now discuss how the decay function (decay(..)) is computed in more detail. This function determines the behaviour of the bandwidth estimation mechanism and is designed to perform two functions. First, it decreases the estimated capacity over time. This pessimistic approach reduces overestimations. Second, it determines when a traffic injection is initiated.

The first question when designing the decay function is what basic function to use for it. We have experimented with a number of functions. Fundamentally, we could choose between linear decay, decay which decreases over time (getting flatter) and decay which increases over time (getting steeper).

We have experimented with all three types of functions and found that a function which decreases over time leads to significant underestimation as the capacity quickly nears the threshold. This in turn lead to very low reported available bandwidth. While we consider overestimations to be more serious than underestimation, we considered this trade-off too large. Both functions with linear decay and increasing delay showed suitable results, with the linear one again resulting in a more pessimistic approach with lower reported available bandwidth.

In the end we have chosen to use an increasing decay. This type of function models a higher confidence in the value of the upper bound for the near future after a traffic injection followed by decreasing confidence over time.

The second question is the gradient of the decay, i.e., how fast the decay value rises and thus C falls. Modifying the gradient changes how frequently traffic injections are performed. In stable environments, a low gradient can be used, reducing overhead. To be more responsive (see Requirement R2) in very dynamic situations, a high gradient

should be chosen. To control the gradient, we introduce a new parameter L which determines how long it takes the decay function to reach the threshold, i.e., how many rounds the loop in Algorithm 11 is executed before a new injection occurs. For now, L is set by the application. We discuss a more sophisticated approach in the next section. The resulting decay function is:

$$decay(t, L, uB, thr) = \log_L(t) * (uB - thr) \tag{4.6}$$

Note that as the threshold is computed dynamically and can increase if the current lB is high (i.e. there is a lot of traffic), the result value of the decay function can actually decrease between two consecutive calls, even if t increases. This results in an increase of the estimated capacity C.

4.5.6 Dynamic Decay Rate Adaptation

The approach we presented so far relies on a fixed value for L. This introduces two problems. First, what if the application does not know the overall stability of the system and thus cannot set L correctly? Second, what if the system stability varies over time, such that L should be adapted?

To counter these problems, we add an algorithm to let our estimation approach choose L automatically, depending on the current stability. To do so, we use a history of deltas between successive uB measurements. Using this history we scale between a pre-set maximum and a minimum L. If uB remains stable over time, we choose a large L to reduce unnecessary traffic injections. If we detect large changes to uB, we choose a small L to increase the rate of decay, thus improving the accuracy. The history δ is given as:

$$\delta = \alpha * 1 - \frac{min(uB_{old}, uB)}{max(uB_{old}, uB)} + (1 - \alpha) * \delta \tag{4.7}$$

It computes a weighted average of past deviations between two consecutive uB measurements. The parameter uB_{old} contains the previous value of uB, α is a smoothing factor between 0 and 1 that can be used to control how quick the history adapts to new values. If α is set to a low value, the history reacts slowly. A large α results in quick history adaptations. For our evaluations we used $\alpha = 0.2$. Using this equation we can now specify our new L (and thus the length(..) function) as:

$$L = L_{min} + (1 - \delta) * (L_{max} - L_{min}) \tag{4.8}$$

The equation sets L between a minimal (L_{min}) and maximum (L_{max}) value by adding an amount to L_{min} that depends on the current stability, given by our history δ.

4.5.7 Decay Rate Reset

With our new dynamically computed L, we can adapt the traffic injection rate depending on the stability of uB. But there is another possibility to further reduce the number of injections. Our decay function models our decreasing confidence in a previous measurement of uB. A traffic injection is initiated when this confidence is too low. However, there are situations, in which we can increase our confidence in a previous measurement again, specifically with respect to possible overestimations.

The basic idea is to take the amount of current traffic (represented by the lower bound lB) and its distance to the current uB into account. If there is a lot of traffic, i.e. lB gets close to uB, our confidence in uB increases, because it is less likely that we did overestimate uB. As a consequence, we are resetting the rate of decay in this situation. Thus, in case of higher traffic in the system, we are further delaying the next traffic injection, as we have higher confidence in our upper bound.

This has another advantage. We are less likely to initiate a traffic injection in situations where the system is using significant amounts of bandwidth. This reduces the probability to interfere with the operation of the system (see Requirement R3). On the other hand we perform more regular traffic injections in situations where the system has sufficient bandwidth resources available. This dynamic adaptation thus not only reduces overall overhead, but moves the generation of overhead to situations where its effects are less noticeable.

Note that this optimization should not be applied if underestimations are considered very severe. The likelihood of underestimations is not reduced by high traffic. Therefore, using decay rate resets potentially results in more underestimations.

To realize the described behaviour we reset the rate of decay in two cases. First, we perform a reset, if the new estimation of the capacity C is higher than the last estimation of C. This can happen, if lB has increased so much that the result of the

decay function decreases, as described in Section 4.5.5. In addition we also reset the rate of decay each time we change the upper bound, either because of a traffic injection or because the current traffic was found to be higher than uB. In both cases we have a reliable new value for uB, which means we can set C equal to uB and restart the decay process.

In some circumstances it may be possible that the rate of decay is constantly reset. In this case it can take very long for C to reach the threshold. While this is intended behaviour, we want to put an upper limit on the duration for which we can delay the next traffic injection. To do so, we also define a maximum interval M for traffic injections, i.e., an interval at which a traffic injection is always performed regardless of whether the decay function has reached the threshold. For our evaluation we have set this as three times the maximum interval for parameter L.

4.5.8 Extended Capacity Estimation

To add the additional optimizations described in the last two sections, we have to extend our basic capacity estimation algorithm (see Algorithm 11). Our extended capacity estimation algorithm is shown in Algorithm 12.

It changes the original algorithm only slightly. First, we remove the parameter L from the parameter list passed to the algorithm, as L is now computed dynamically, using the length(..) function. We replace L with two new parameters L_{min} and L_{max}, specifying the minimal and maximum length between traffic injections. Second, we add the two cases for resetting the decay rate, $lB > uB$ (Line 7–12) and $C > C_{old}$ (Line 18–20) together with necessary temporary variables to store the old values of uB and C. Third, we add the maximum injection interval M (Line 26–31) and a new counter variable t_m to enforce it.

4.5.9 Traffic Injection

Traffic injections are a key part of our approach. Whenever the estimation wants to re-evaluate the upper bound uB, it initiates an injection. A traffic injection essentially means that the peer requiring the injection sends or receives as fast as possible to make sure that the uplink or downlink capacity is fully used.

In case of a traffic injection on the uplink, i.e., to measure the uplink capacity, the injecting peer must select a receiver for its traffic. A naïve approach would be to simply

Algorithm 12 Extended Capacity Estimation Algorithm

```
1: procedure EstimateCapacity(init, T, L_min, L_max)
2:     uB = C = init;
3:     t = 1; t_m = 1;
4:     L = 0; M = 3 * L_max;
5:     loop
6:         lB = usedBw()
7:         if lB > uB then
8:             uB_old = uB
9:             uB = lB
10:            L = length(uB, uB_old, L_min, L_max)
11:            t = 1
12:        end if
13:        thr = threshold(lB, uB, T)
14:        C_old = C
15:        C = uB−decay(t, L, uB, thr)
16:        t = t + 1
17:        t_m = t_m + 1
18:        if C > C_old then
19:            t = 1
20:        end if
21:        if C ≤ thr then
22:            uB_old = uB
23:            uB = injectTraffic()
24:            L = length(uB, uB_old, L_min, L_max)
25:            t = t_m = 1
26:        else if t_m > M then
27:            uB_old = uB
28:            uB = injectTraffic()
29:            L = length(uB, uB_old, L_min, L_max)
30:            t = t_m = 1
31:        end if
32:    end loop
33: end procedure
```

use one other peer. However, we may select a peer that has a smaller downlink capacity than our uplink capacity. In fact, given that many Internet connections are asymmetric, this is a likely scenario.Thus we would measure the other peer's downlink instead of our uplink. In addition, we may induce a lot of overhead to the other peer. Therefore, we select a group of peers, which then simultaneously receive data from the injecting peer. We choose the peers in the peer group based on their own available bandwidth, i.e., we prefer peers with high available bandwidth. The details of this election process, how many peers to choose and how to perform the coordination between the peers are beyond the scope of this paper. So far, we rely on a special coordinator peer to control the process. More sophisticated coordination protocols are subject to our future work. For traffic injections on the downlink, i.e., to measure the downlink capacity, we follow essentially the same approach. The only difference is that instead of sending data, the peer initiating the traffic injection is receiving data from the peer group.

Note that there can be a dependency between uplink and downlink traffic injections. High upload speed can negatively influence download speed and vice versa, particularly on certain types of connection (e.g. DSL). This would result in less accurate estimations. Therefore, we delay a traffic injection when another one is currently running.

4.5.10 Available Bandwidth

Now that we have determined C, we can simply calculate the available bandwidth aBW by subtracting the current bandwidth from the estimated capacity. However, this value can fluctuate. While one of our requirements is responsiveness (R2), many applications may prefer a somewhat more stable value. By using a simple history-based approach we can reduce the impact of temporary spikes on the reported available bandwidth. The resulting equation for aBW is as follows:

$$aBW := \beta * (C - lB) + (1 - \beta) * aBW \tag{4.9}$$

Just as for the history used in the computation of L (see Equation 4.7), we introduce a smoothing factor β between 0 and 1 to control how fast aBW adapts to new values. In case of a small β value, temporary spikes are smoothed out and aBW reacts slowly to changes. If β is set near 1, the history is mostly ignored, and aBW reacts faster.

Note that in order to determine the available upload and download bandwidth separately, we can simply run this approach once for upload and once for download. One exception to this is the traffic injection. As high upload speed can negatively influence download speed and vice versa, particularly on certain types of connection (e.g. DSL), we want to avoid simultaneous traffic injections for the upper upload and download bounds. To avoid this, we can simply delay a traffic injection when another one is currently running.

Chapter 5

Evaluation

In this chapter we will evaluate the contributions of this thesis. We will examine our propagation overlay in Section 5.1 and show that it is able to deliver updates quickly (i.e. with a small number of hops in the overlay) while being scalable. Second, we will look at our coordinator election algorithm in Section 5.2 and show that it can elect suitable coordinators with limited overhead in the vertical update propagation overlay. Finally, we will review our bandwidth estimation algorithm in Section 5.3 and show that it provides realistic estimates while generating limited bandwidth overhead.

5.1 Update Propagation Overlay

In this section we evaluate our update propagation overlay with a number of experiments. We have implemented and integrated our approach into the peers@play prototype (see Section 2.2). Our evaluation setup requires a large number of peers in the system as well as the ability to get accurate overlay hop counts and realistic delay measurements. Our experimental setup was designed to satisfy those criteria. To achieve a large number of peers we ran up to 350 instances of our prototype on an IBM BladeCenter with 6 Blades (each with 2 Intel Xeon QuadCore CPUs with 2.33 GHz and 6 GB RAM). While this enables us to get accurate hop counts, the delay between these instances is unrealistically low, as they all run on the same local network or even system. Thus, to perform realistic delay measurements, we set up two additional peers at two separate off-site locations. The first peer was connected to the Internet using a standard home DSL Internet connection. This peer served as our measuring node. The

second peer was connected to the Internet using a standard home cable Internet connection. This peer serves a special coordinator (see Section 5.1.2). Figure 5.1 depicts this setup.

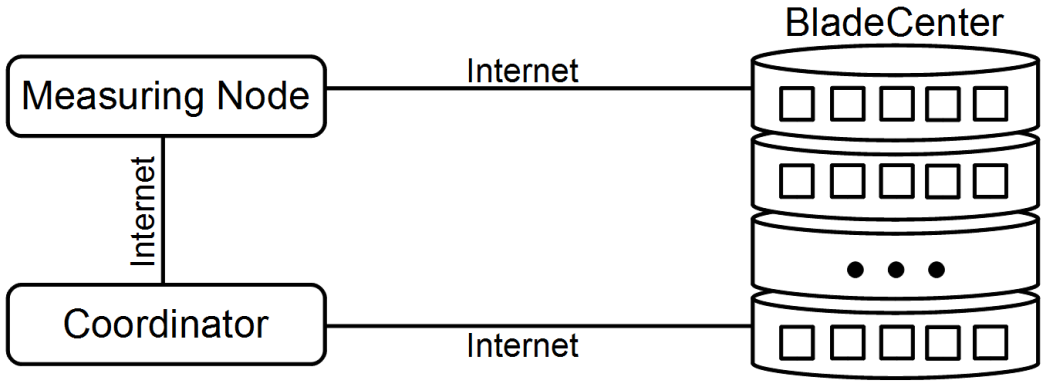

Figure 5.1: Evaluation Setup - A BladeCenter runs a larger number of peers@play instances, while two off-site nodes are set up to enable delay measurements

We have set the following default parameters for our experiments. The size of the virtual world ($worldSize$) is 400m. The number of peers in this world (P) is 350 (based on the number of instance of the peers@play prototype which the BladeCenter is capable of running in parallel). Each peer has an AoI of 20m and each event has an AoE of 20m. Each zone can hold a maximum of 30 peers ($maxPeersPerZone$), before a zone split is initiated. As most peers are in the same local network, or even the same machine, we cannot use realistic bandwidth values and thus manually set a default AoP size of 6 peers. Finally, $eventDistance$ is the average distance of an even location (and the center of its corresponding AoE) from the generating peer.

These default values are depicted in Figure 5.2 and apply to all experiments unless otherwise noted. The peers in our system move along random paths and regularly generate events resulting in their corresponding updates. For each experiment, all P peers were added to the system, which then ran for 10 minutes. During this time, all incoming events at the measuring node were recorded.

peers in system P	350
zone split limit $maxPeersPerZone$	30 peers
AoP size AoP	6 peers
size of world $worldSize$	400m
AoI size AoI	20m
AoE size AoE	20m
average event distance $eventDistance$	10m

Figure 5.2: Evaluation Parameters: Update Propagation Overlay

5.1.1 Overlay Hops

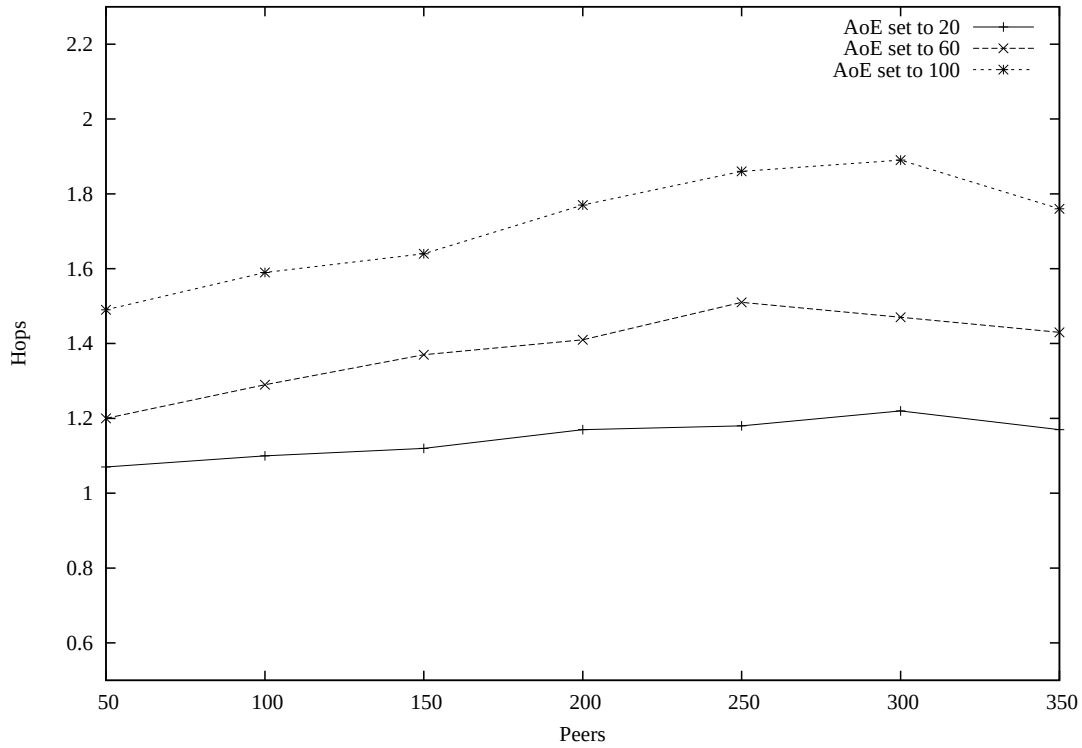

Figure 5.3: Hop Count - Average hop count in the update propagation overlay, showing the influence of AoE size

As we have previously discussed, the overlay connections of a peer are determined by its location in the virtual world. It is therefore not possible for a peer to freely choose

which peers in the system to connect to. This in turn means we cannot optimize our overlay for network delay. The primary metric for the interactivity of our system is therefore the number of hops a message makes in the overlay before reaching its target. We thus look at the average number of hops each message has traversed before reaching our measuring node. To do so we have run our experiment repeatedly, increasing P by 50 each time. Figure 5.3 shows the results of these experiments.

In a traditional client/server approach, all messages take two hops to be delivered, as they are sent to the target client via the server. In our system, all updates delivered via the horizontal update propagation overlay only require one hop. However, when the vertical overlay is utilized, the hop count is two or higher (depending on the location of the target and the depth of the quadtree). As can be seen, the combined average hop count stays below two hops per update in our approach.

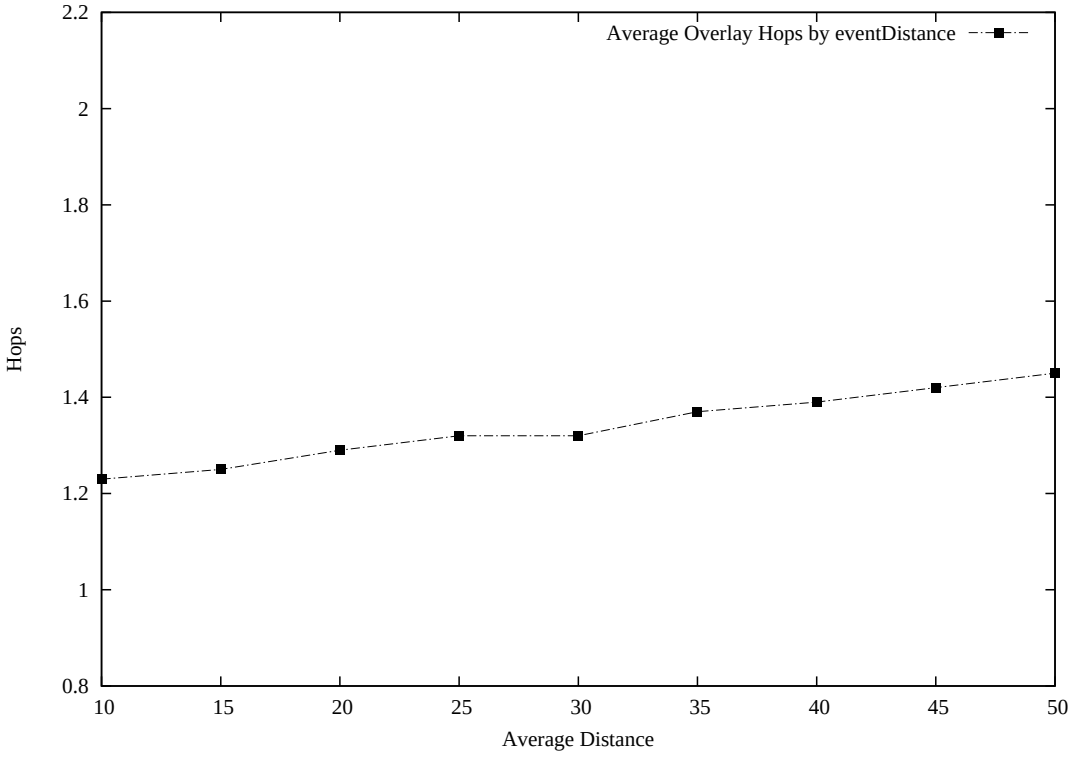

Figure 5.4: Influence of Event Location on Hop Count - Average hop count in the update propagation overlay, showing influence of *eventDistance*

It should be noted that the value of AoI and AoE can have a significant impact on the average hop count. This is due to the fact that larger values for these parameters

increase the number of peers an update must be delivered to, both inside and outside the current zone. When this occurs, more updates need to be delivered via the vertical overlay, thus increasing the hop count. We therefore repeated our experiments for varying sizes of AoE (note that increasing AoI is analogous to increasing AoE, so we did not perform a separate experiment for AoI). As expected, increasing values for AoE also increased the average hop count. However, even with AoEs which cover 25% of the virtual world, the hop count stays below that of a client/server approach.

Note that the drop in the average hop count around the 250/300 peer marks can be attributed to an increase in the depth of the vertical overlay quadtree. At this point a significant number of zone splits have occurred and the average number of peers in each zone is decreased. This in turn means that each peer has a larger percentage of its zone's peers in its AoP, thus reducing the load on the coordinator and therefore the average hop count.

As larger AoIs/AoEs increase the average hop count, we expect the same results with increasing distance of the AoEs from the originating peer. The further events are away from their originating peer, the less likely it is that they can be propagated using the horizontal update propagation overlay. Figure 5.4 shows the effect $eventDistance$ on the overall hop count. We have run our default experiment several times, with increasing values for $eventDistance$. As can be seen, the hop count increases as expected.

Considering this, there are scenarios in which it is possible for the average hop count to exceed that of a traditional client/server approach. This is the case for very large values for AoI/AoE in combination with a high depth of the quadtree. Similarly, this can occur when updates are regularly generated at a significant distance from the originating peer. However, neither very large AoIs/AoEs, nor distant event locations are realistic assumptions for most MMVEs. In addition the depth of the quadtree grows very slowly with the number of peers ($log_4(P)$).

5.1.2 Delay

Even though our key metric is the average hop count, we also evaluate our approach for delay. Our setup imposes some limitations to this, however. We cannot get realistic delay measurements for the entire overlay, as most peers are in the same network or even on the same local machine. Thus we have included two additional peers which enter the system via separate Internet connections. By making one of these peers the

coordinator of a zone, and placing the measuring node inside it, we can evaluate the update delay within this zone. Any update sent directly to or from the measuring node has to traverse an Internet connection, thus taking a realistic amount of time to reach its destination. The same is true for any message sent to or from the coordinator. Under the assumption that no update is sent from outside the peer's zone, this gives us update delays as experienced by an individual participant in the virtual world. While this assumption in not realistic, it allows us to evaluate the horizontal component of the update propagation overlay with regards to delay.

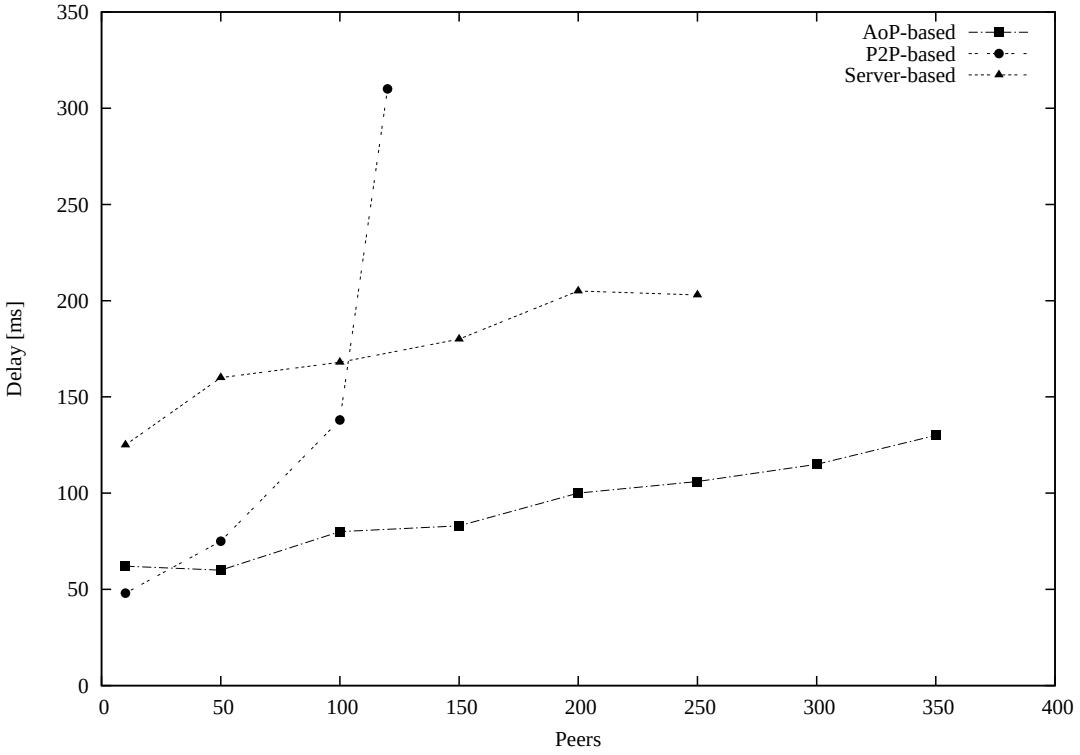

Figure 5.5: Delay - Average update propagation delay for a single zone

We therefore modify our experiment as follows. First we set $maxPeersPerZone$ to ∞. This has two consequences. First, the depth of the quadtree will remain at 1, i.e. no zone splits will be performed. Second, due to this lack of zone splits, the system will ultimately fail, when the single zone's coordinator becomes overwhelmed. Second, instead of using our coordinator election algorithm we pre-determine a single node as coordinator (i.e. the off-site peer). In order to compare our results, we run the exper-

iment again, this time setting AoP to 0. In effect, this creates a client/server system, as all updates have to be sent via the coordinator. Finally, we run the experiment a third time, setting AoP to ∞. This creates a naïve P2P approach where all peers are connected to all other peers. Figure 5.5 shows the results of these experiments.

As we can see, the naïve P2P approach provides the lowest delay for very small numbers of peers in the system. However as peers are added, they quickly become overwhelmed by having to communicate with all other peers and delay increases. With 120 peers in the system, the delay is already above 300ms. For higher numbers, the system fails. While this approach offers low delay in theory, in practice it does not scale. As we have selected a highly suitable system as coordinator, the client/server-like approach scales better, failing only when more than 250 peers were added to the system. As expected delay is about twice that of the naïve P2P approach.

In comparison our approach provides delay below that of the client/server-like approach while providing higher scalability. We achieve delay very close to that of the naïve P2P approach while allowing more peers in the system than the client/server-like approach. Note again that the number of 350 peers is not a limit imposed by our approach but by the resources of the BladeCenter used for evaluation.

Our approach makes use of direct connections between the peers to deliver a majority of update messages with one hop in the overlay (resulting in low delay) while fully utilizing both the resources of the peers and the coordinator (resulting in a scalable system).

Finally, note that we attribute the small increase of delay with increasing peer number to the increasing load on the BladeCenter.

5.2 Coordinator Election

In this section, we evaluate our coordinator election algorithm. As our distributed prototype does not allow us a complete view of the overlay, we have developed a simulation, which enables us to fully evaluate elections in the system. The simulation starts with a single peer and continually adds up to P peers to the system. This results in regular zone splits, each of which initiates an election to select four new coordinators.

Our simulation uses the following parameters. P is the total number of peers which are added to the system. As a default value, we are adding 50.000 peers. $suitTreshold$ is the suitability threshold above which a peer is considered suitable to serve as coordinator. Our default value for this threshold is 0.7. As shown in Algorithm 8, we set the threshold for a local candidate search at $suitThreshold + (1 - suitThreshold/2)$, which means $localSuitThreshold$ is set to 0.85 per default. Finally, we need to set the maximum number of peers per zone, i.e. the number above which a coordinator initiates a zone split. $maxPeersPerZone$ is set to 50 by default. As a zone split is based on the available resources of the coordinator, a coordinator with a suitability of 1.0 would thus initiate a split when the 51st peer attempts to enter its zone. We also assume that each coordinator can handle at least 10 peers, which means we scale each coordinator's $zoneSplitThreshold$ between 10 and $maxPeersPerZone$, based on its suitability. The suitability of all peers is randomly set between 0.0 and 1.0. $noOfSamples$ represents the number of samples (i.e. coordinates) generated for each election. It is set to 4 by default.

peers added to system P	50000
suitability threshold $suitTreshold$	0.7
local suitability threshold $localSuitThreshold$	0.85
zone split limit $maxPeersPerZone$	50
number of samples per election $noOfSamples$	4

Figure 5.6: Evaluation Parameters: Coordinator Election

Figure 5.6 lists all election parameters and their default values. These values apply to all evaluation results, unless otherwise noted.

5.2.1 Average Suitability of Elected Coordinators

First we will look at the suitability of the peers selected by our approach. Figure 5.7 shows the average suitability of the selected peers. We have run the simulation 50 times, starting with $P = 1000$ and increasing P by 1000 each time. We then repeated this experiment for increasing values of $maxPeersPerZone$. As we can see, the average suitability is consistently above 0.9 in all scenarios. We can also observe that the total number of peers in the system has no notable influence on the results. However, the maximum number of peers in each zone ($maxPeersPerZone$) does. The higher the average number of peers in a zone is, the better the election results. This is expected, as each sample and local search generally reaches a larger pool of candidates, the higher the average number of peers in each zone is.

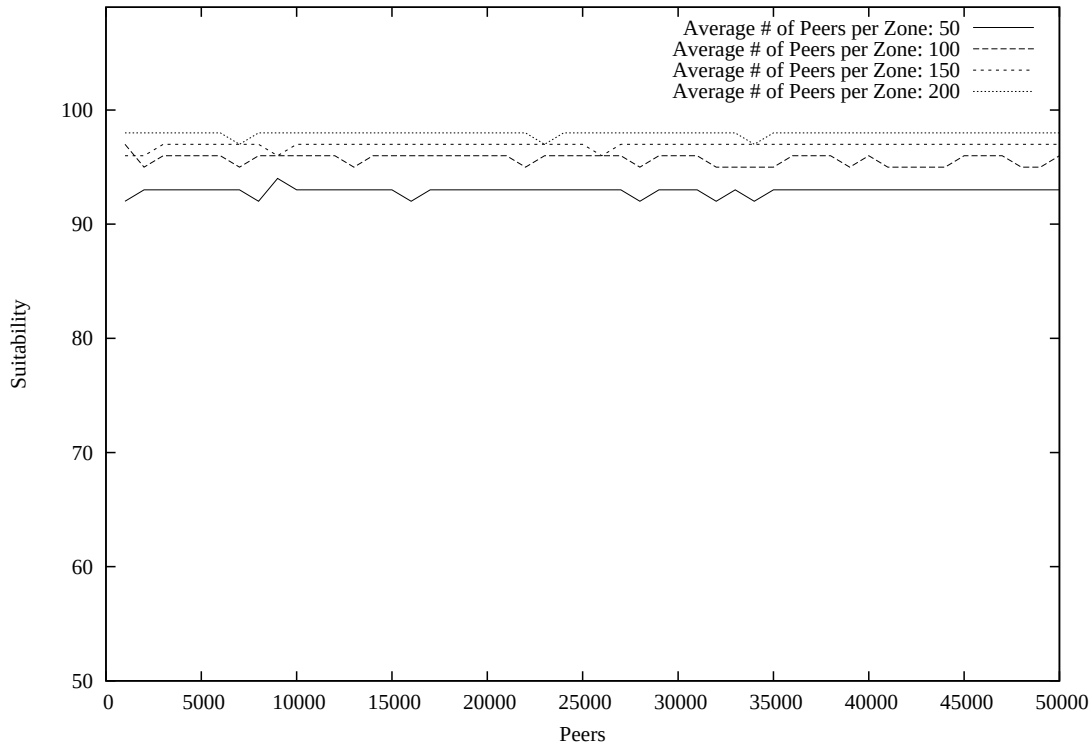

Figure 5.7: Average Peer Suitability - Average suitability of peers, showing the influence of $maxPeersPerZone$ on results

5.2.2 Local Search

We will now consider the effect of the local search component of our algorithm on its results. Figure 5.8 shows the relation of the number of elections which use the sampling component of our algorithm vs. the number of those which find suitable coordinators using a local search only. As we can see, $maxPeersPerZone$ once again influences the result. The lower $maxPeersPerZone$ is, the more often the algorithm uses its sampling component to elect the coordinators. When it is set to 20 (which results in an average of 10 peers per zone with 50000 peers in the system), the local search almost never finds sufficient suitable results. With increasing $maxPeersPerZone$, the percentage of successful local searches increases as well. When it is set to 70 (which results in an average of 48 peers per zone with 50000 peers in the system), the number of unsuccessful local searches is under 5%. When it is set 100 or above, the algorithm can find almost all new coordinators locally.

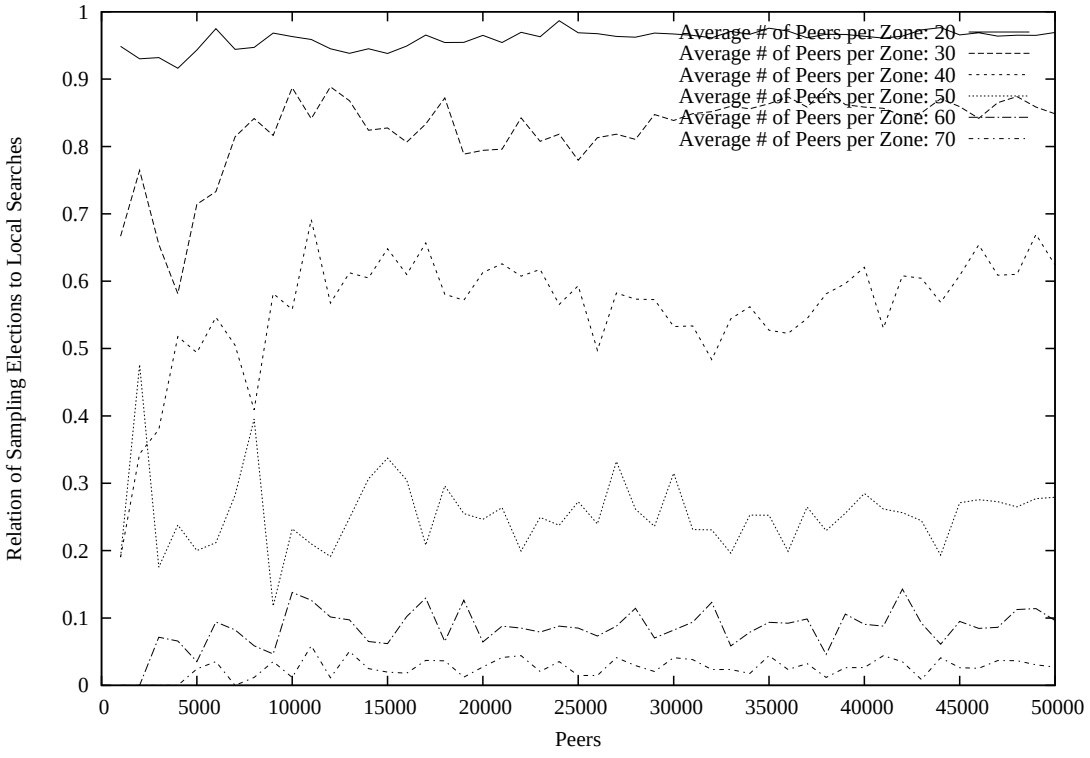

Figure 5.8: Sampling vs. Local Search - Relation of elections which use sampling vs. elections which use local search only, showing the influence of $maxPeersPerZone$

5.2.3 Election in a Resource-constrained System

Our previous results were obtained using an even distribution of suitability. This means that with a *suitTreshold* of 70, 30% of all peers in the system are suitable as coordinators. We will now look at the results of our algorithm when only a small percentage (5%) of all peers are suitable as coordinators. In Figure 5.9 we observe similar results to those in Figure 5.7, with two differences. First, the average suitability is lower in this resource-constrained scenario, though the results remain acceptable to good. Once again, *maxPeersPerZone* has a distinct influence. Second, we note that the results are less consistent, particularly when few peers are in the system. This is due to the fact that there are fewer overall candidates and the random effect of the sampling is more pronounced.

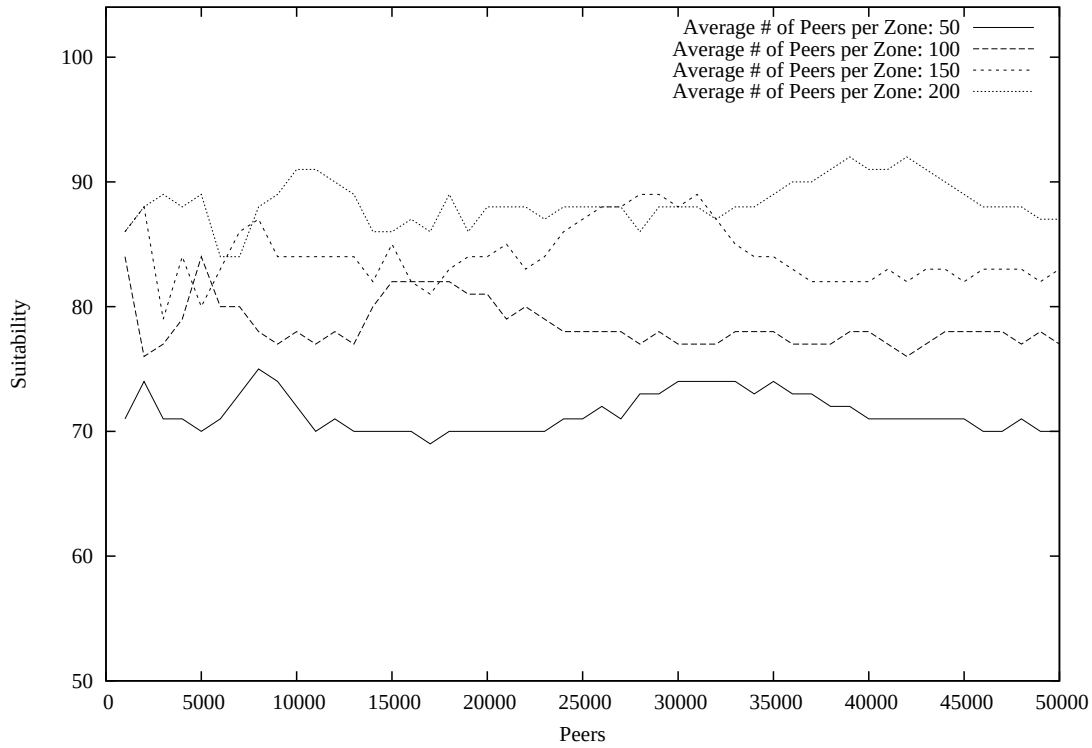

Figure 5.9: Suitability in a Resource-constrained System - Average suitability of peers in a system with only 5% suitable peers

Even though the average suitability of the elected coordinators is lower than in the 30% scenario, the relative quality of the results remains high. Figure 5.10 shows that

for 50000 peers, the elected coordinators are respectively in the 94th (30% scenario) and the 96th (5% scenario) percentile of all peers. In other words, while the average suitability of the selected peers can be as low as 70 (which means that roughly half of the selected peers are below the suitability threshold), the algorithm has still managed to select most of the top peers in the system. This means that our algorithm can elect a high quality set of coordinators even in a resource-constrained environment.

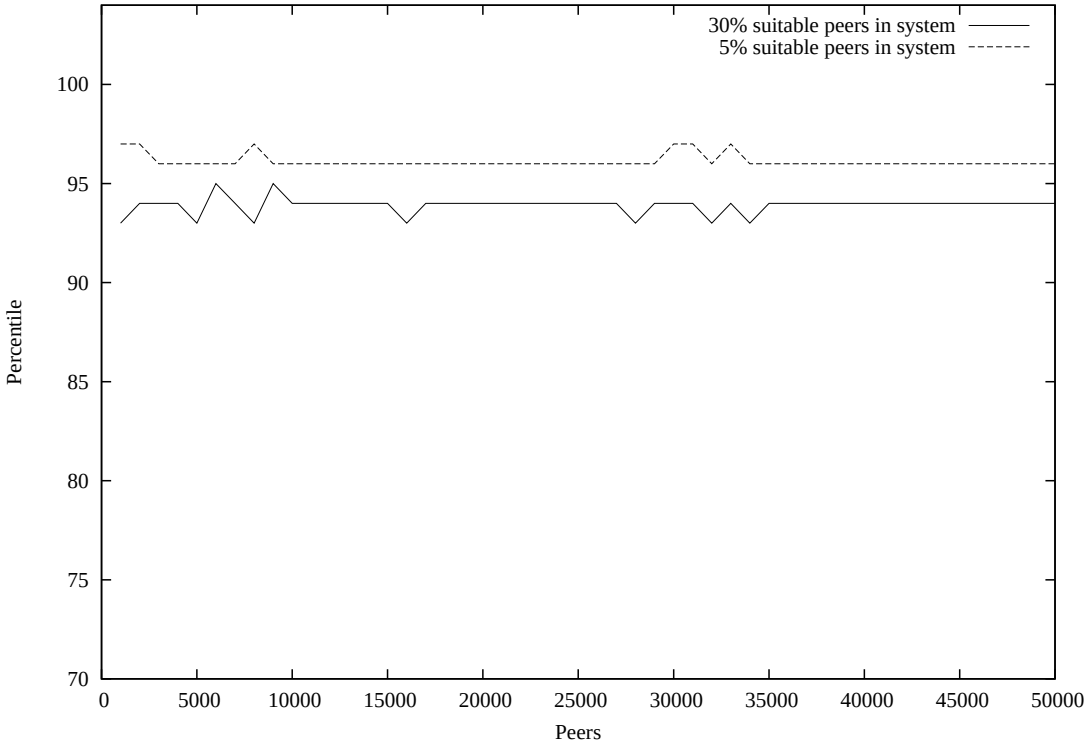

Figure 5.10: Percentile of Suitability - Percentile of elected coordinators among all peers regarding suitability

5.2.4 Influence of Samples

We will now look at the influence the number of samples has on the election results. Figure 5.11 shows our experiment with increasing values of $noOfSamples$ and for the 30% and 5% suitability scenarios. As we can see, the average suitability increases only slightly with a higher $noOfSamples$. This fits with our result regarding the high success rate of the local search. In most cases, a single zone provides a sufficiently

large pool of candidates to successfully elect a coordinator. While additional samples increase the pool of candidates, the improvements to average coordinator suitability are too small to justify a large number of samples.

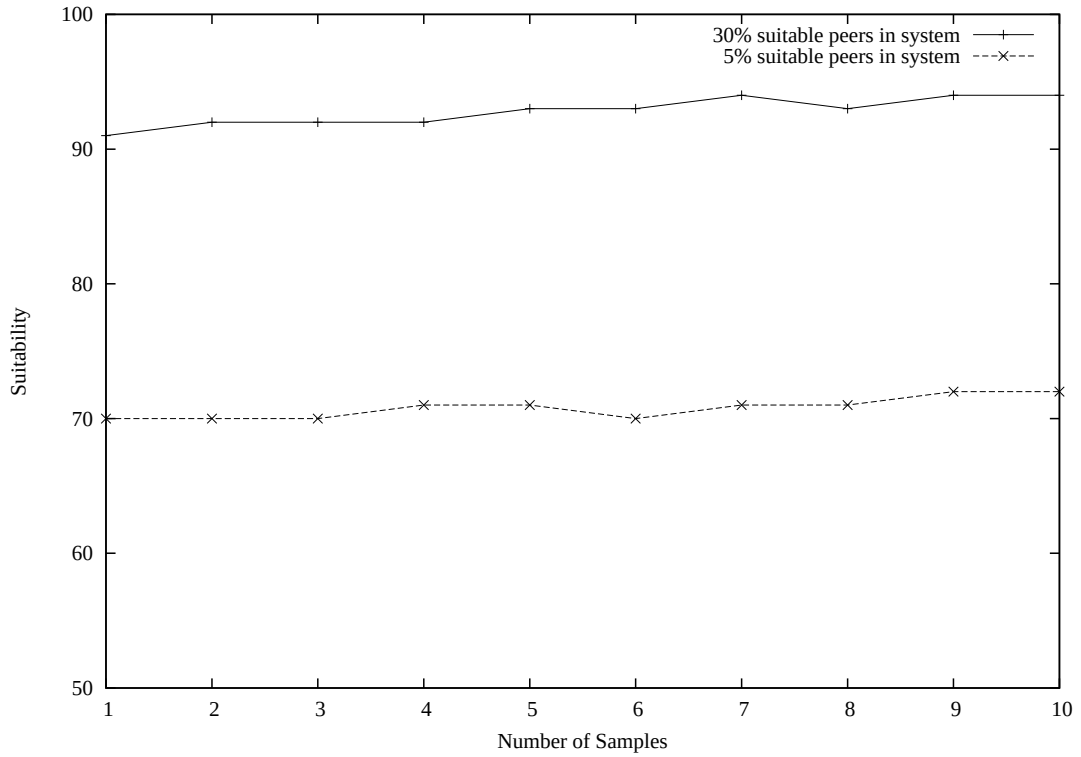

Figure 5.11: Influence of $noOfSamples$ on Suitability - Average suitability of peers, showing the influence of the number of samples used during an election

5.2.5 Election of Unsuitable Coordinators

Even when the average suitability of elected coordinators is high, this does not necessarily mean that every elected coordinator is suitable. In fact, as previously discussed, our algorithm cannot guarantee this. Figure 5.12 shows the number of elected coordinators, with a suitability score below $suitThreshold$. Once again, $maxPeersPerZone$ has significant influence on the results. If $maxPeersPerZone$ is above 40, very few (if any) unsuitable coordinators are elected. Very small values of $maxPeersPerZone$ however, can result a number of unsuitable coordinators. For example, when set to 20, 314 of the elected coordinators were unsuitable. Out of a total of 7141 coordinators in

the system, this is 4.4%. When $maxPeersPerZone$ increases 25, this drops to 0.93%.

Note that even when $maxPeersPerZone$ is not set to a small value, some individual coordinators may still be below $suitThreshold$, particularly among those selected early during system startup. This necessitates a coordinator replacement strategy (see Chapter 6.1). Also note that the sudden increase in unsuitable coordinators at around 18000 and 25000 peers in the system can be attributed to the fact that zone splits are not evenly distributed in time but appear in clusters when multiple zones exceed $maxPeersPerZone$ around the same time. This effect can also be observed in Figure 5.13 and to a lesser degree Figure 5.14.

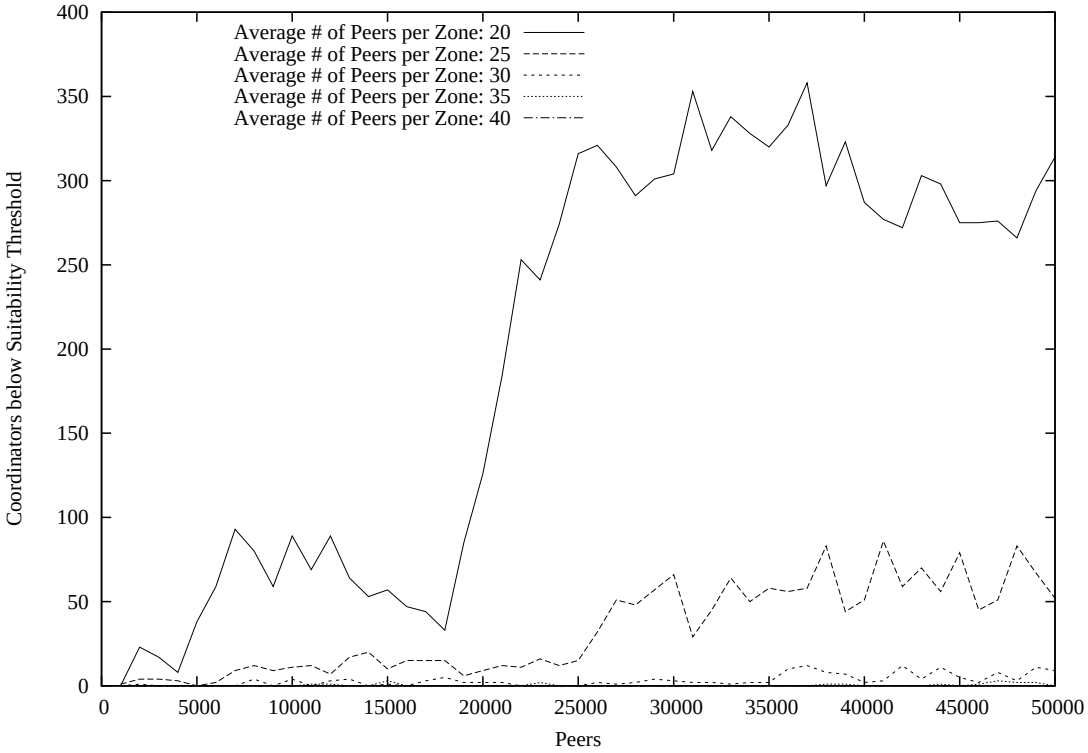

Figure 5.12: Coordinators below Suitability Threshold - The number of elected coordinators with a suitability score below $suitThreshold$, showing the influence of $maxPeersPerZone$

5.2.6 Overhead

To determine the overhead generated by our election algorithm we look at two metrics. First, Figure 5.13 shows the overall number of messages (request and respond) which

are generated during each experiment. After adding 100000 peers to our system, only 3500 election messages are generated. Second, Figure 5.14 shows the number of hops which these messages traverse in the horizontal update propagation overlay. We look at both the hops an individual request message (and its response message) traverse in the overlay, as well as the total number of hops for an election. As can be expected, the number of hops for a request is bounded by the depth of the tree. In our experiment with 100000 peers in the system, the depth of the tree is 7, resulting in a worst case hop count of 14 for a request and its response. On average, each request and response took only 5.1 hops in the overlay, however. Due to the slow increase in the depth of the tree, this number also increases slowly.

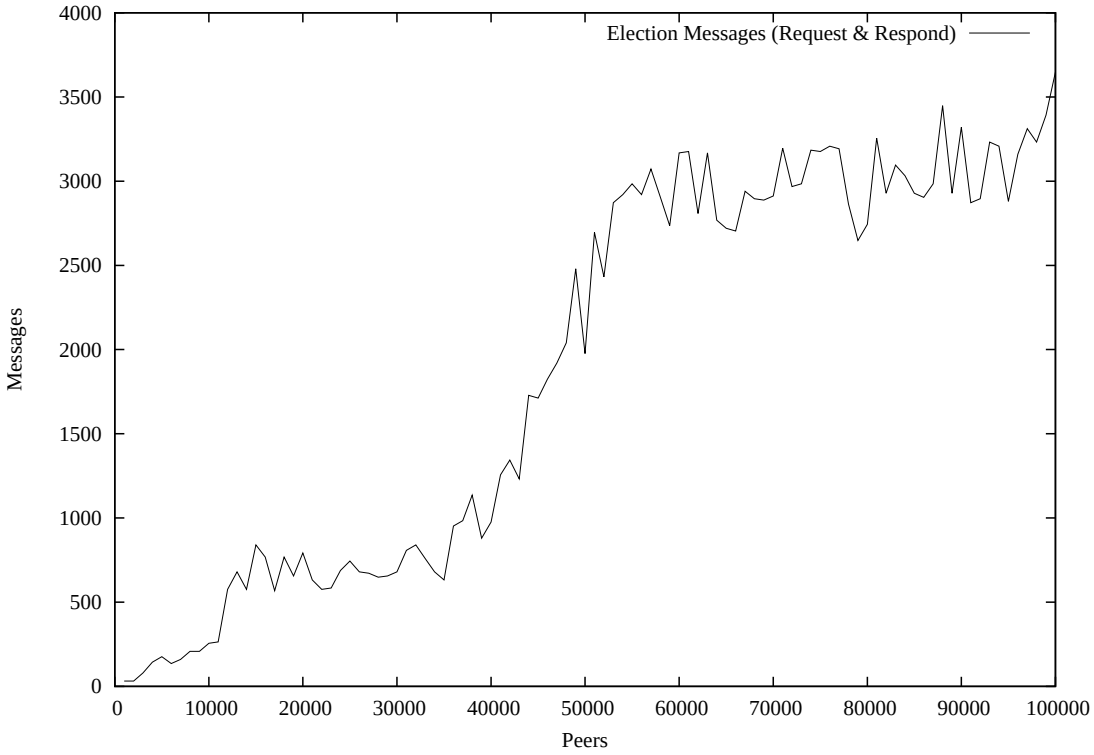

Figure 5.13: Overhead (Messages) - The number of message generated by the election algorithm (per an individual request and per election)

In conclusion, our election algorithm provides very good results with manageable overhead. We note the importance of the local search component and that a low number of samples is sufficient for good results. The algorithm is limited by the average number

of peers in the zones. If this number becomes too small, the quality of the algorithm's results decreases. However this is only the case in systems which are already severely resource constrained.

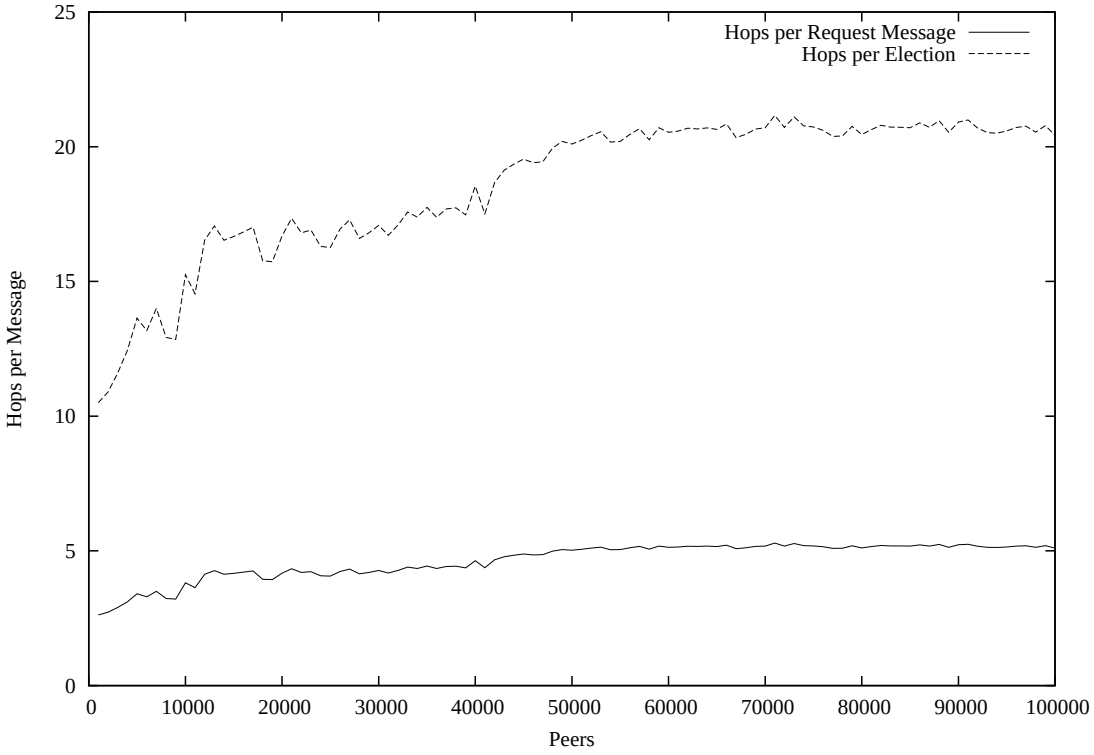

Figure 5.14: Overhead (Overlay Hops) - The number of hops in the overlay by request and response messages

5.3 Bandwidth Estimation

In this section we evaluate our approach for bandwidth estimation with a number of experiments. The experiments are performed on five peers that we deployed on different computers in the Internet. Our evaluation setup is depicted in Figure 5.15. The five peers form an overlay network. Three of them (Peers P_1, P_2, P_3) are executed as micro instances in the Amazon Elastic Computing Cloud (EC2). The other two peers (Peers M and F) run on Intel Core2 Duo (2 GHz, 2GB RAM) computers. They are located in a LAN that is connected to the Internet via DSL. The Internet connection is advertised by the provider with a speed of 16 Mbps. However, in our experiments we found that the actual capacity of the connection is approximately 3.7 Mbps. We have additionally verified this result through several server-based bandwidth tests.

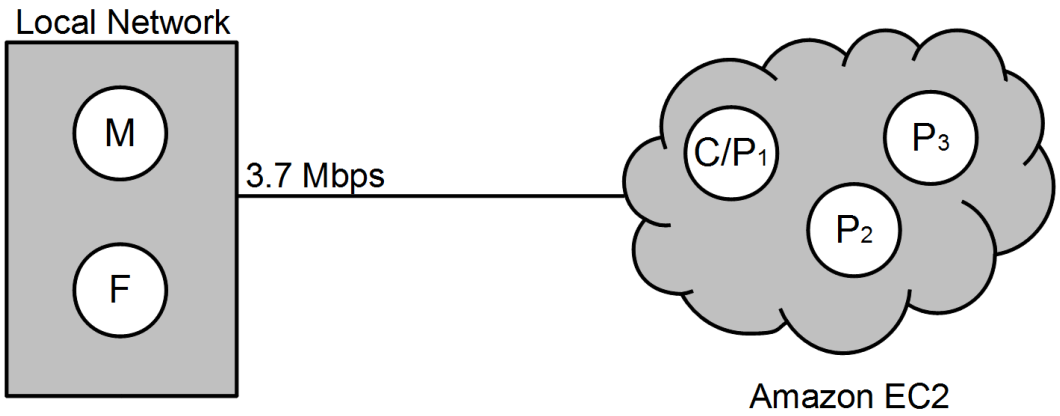

Figure 5.15: Evaluation Setup - Evaluation setup for the bandwidth estimation algorithm

Using this setup, we implement three scenarios. In the first scenario, we measure the available bandwidth of Peer M without any background traffic, i.e. on an otherwise idle system. Peer F is not used in this scenario. The Peers P_1, P_2, P_3 act as the peer group for traffic injections (see Section 4.5.9). Each has a maximum upstream of 2 Mbps. In addition, Peer P_1 also serves as as coordinator to manage the peer group. This is, however, only of marginal interest to this evaluation. In the second scenario, we add background traffic to Peer M to simulate M actually executing an application that uses bandwidth. To do so, M downloads a 175 MB file via BitTorrent. In the third

scenario, we induce background traffic at Peer F, i.e. not at the measuring peer itself, but in the same LAN. Thus, in this scenario M has to share its Internet connection with another active peer. The background traffic is generated – similar to the second scenario – by letting F download a 175 MB file via BitTorrent. The values of additional parameters in our experiments are listed in Figure 5.16:

Threshold parameter T	0.5
minimal injection L_{min}	30 steps
maximum injection L_{max}	120 steps
L history weight α	0.2
aBW history weight β	0.01
Injection size per peer	1MByte
Step length	1s

Figure 5.16: Evaluation Parameters: Bandwidth Estimation Algorithm

With these three scenarios we perform a number of experiments, each focussing on a different aspect of our approach. First, we want to demonstrate the basic functionality of our approach, including the dynamic adaptation of L (see Figure 5.17–5.19). After that, we show the effects of resetting the decay rate under load (see Figure 5.20). Then, we show the influence of the threshold parameter T on the reported available bandwidth (see Figure 5.21) before we discuss how a possible overestimation influences our system and the results (see Figure 5.22). Finally, we show the overhead of our approach, including the effect of the dynamic adaptation of L on it (see Figure 5.23).

5.3.1 No Background Traffic

In our first experiment, we estimate the available bandwidth aBW at Peer M without any background traffic initiated in the local network. Thus, the measured aBW should be close to the capacity of the DSL downlink (3.7 Mbps). Figure 5.17 shows the resulting values for the lower bound lB, the upper bound uB, the estimated capacity C, and aBW.

We can see that our approach reports a constant measured capacity of 3.7 Mbps. The stable measurements lead to an increase in the decay length interval over time and we can see that the reported aBW increases over time with the stability of the

measurements. Note that it stays around 3 MB/s and does not reach the upper bound. This is intended behaviour in order to avoid overestimations. Also note that the lack of background traffic is reflected by the fact that lB is 0 during almost the entire measurement. We can also observe the dynamic adaptation of the parameter L. Initially traffic injections are performed at short intervals, as we have no information about the stability of uB. Since uB is almost completely stable, the interval between injections continually increased until it reaches L_{max}. We can also see the effects of the decay function, which continually reduces the estimated capacity C between traffic injections. After an injection, C is once again set to uB.

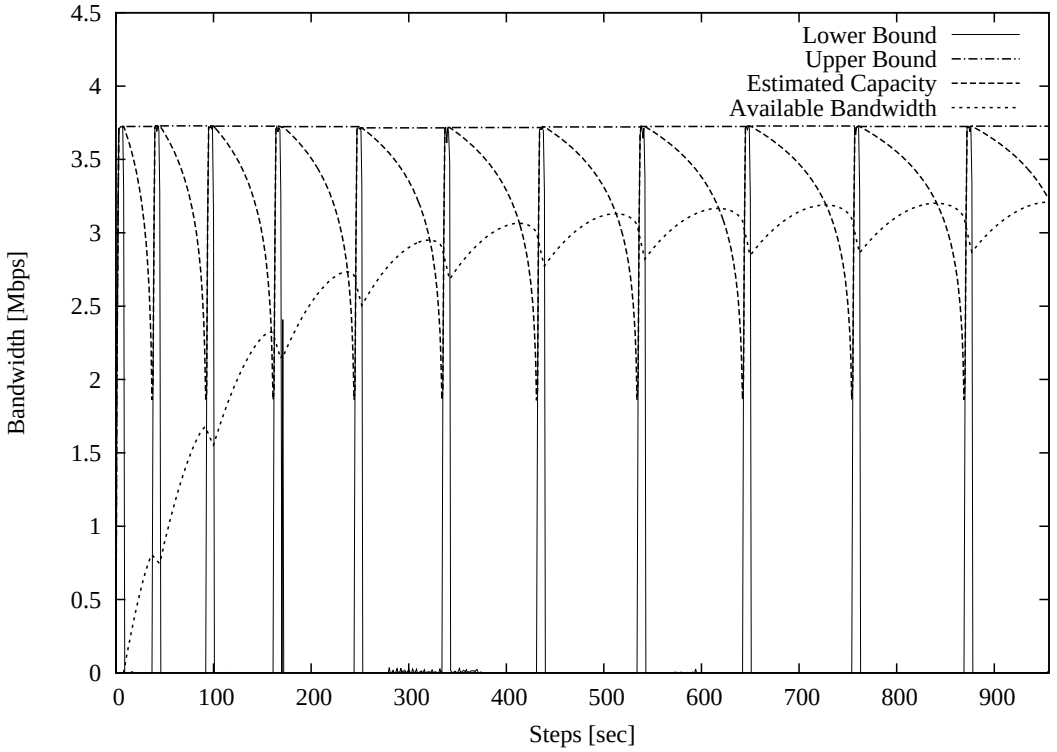

Figure 5.17: No Background Traffic - Bandwidth estimation without background traffic at peer M

5.3.2 Background Traffic at M

In the second experiment we are generating background traffic on Peer M, the same peer on which we are estimating the available bandwidth aBW. The results are depicted

in Figure 5.18. Initially, there is no background traffic and the results are similar to the first experiment. After approximately 480s, the download of a popular 175 MB file is initiated via BitTorrent on M. During the download nearly all of the available bandwidth is used for it.

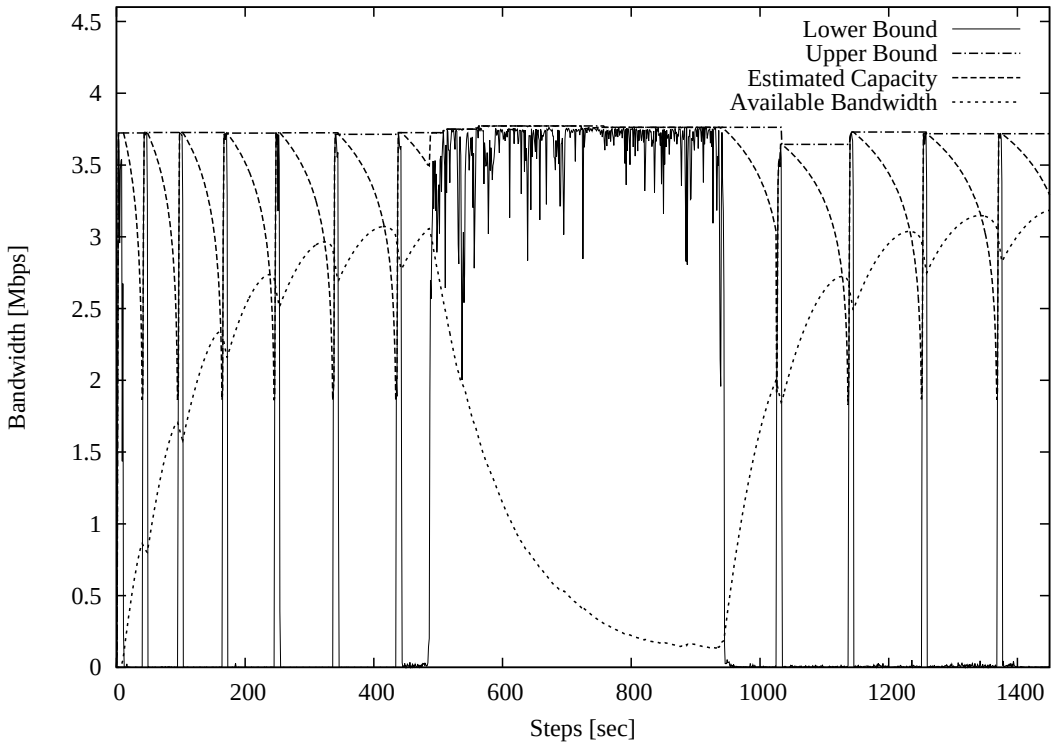

Figure 5.18: Background traffic at *M* - Bandwidth estimation while background traffic is being generated at peer M

Thus, the passive traffic observation of our algorithm should return a good estimate of the available bandwidth. This can be seen by looking at the lower bound lB. During the download it is near the capacity of the DSL connection with sporadic peaks down to 3 Mbps, in rare cases down to 2.5 Mbps. During this time, the reported available bandwidth aBW decreases to approximated 0.2 Mbps. After the download is finished at approximately 940s , lB – i.e. the observed traffic – is 0 again. Now, the bandwidth estimation requires traffic injections in order to determine aBW. One can see a smooth increase of aBW until it reaches approximately 3 MB/s again at about 1300s.

5.3.3 Background Traffic at *F*

In this scenario we are generating background traffic on Peer F, by downloading a 175 MB file via BitTorrent. This is similar to the second scenario, except that – while the traffic is still generated in the same LAN – this is done on a different host. In the first two scenarios, the upper bound uB remains constant. In the first scenario, this was due to a lack of traffic. In the second scenario, this was the case because the traffic was generated at the same host. By generating traffic on F, which Peer M cannot directly measure, we are reducing the amount of traffic that can be sent by M, thus reducing uB.

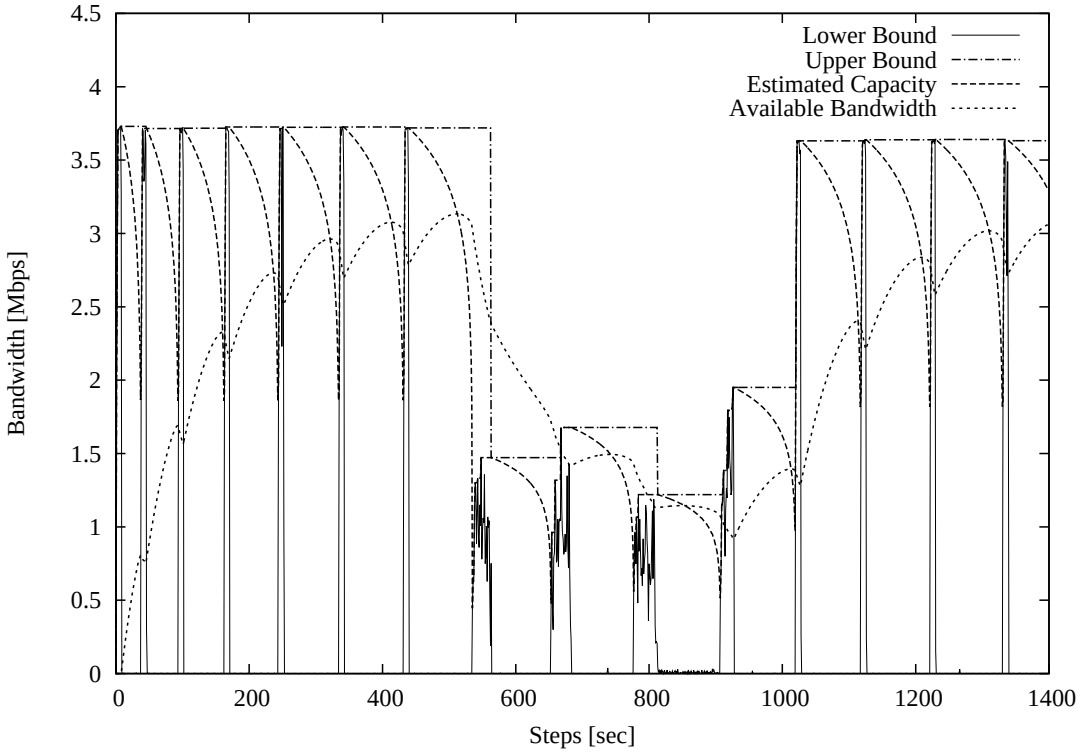

Figure 5.19: Background traffic at *F* - Bandwidth estimation while background traffic is generated at peer F

Figure 5.19 shows the measured results over the experiment's time, as in the previous scenarios. Again the experiment starts and the decay length is adapted and increases while uB is increasing to ca. 3 Mbps and the estimated capacity C is 3.7 Mbps. When the download starts at approximately 500s, the available bandwidth

aBW decreases. Here, uB drops down to ca. 1.4 Mbps. Additional traffic injections are performed when C reaches the threshold. Note the changes of uB, based on the background traffic generated by F during the traffic injections. Also note the decrease of the injection interval, due to the dynamic adaptation of L based on the decreased stability of uB. Interestingly, our approach reports values for aBW above 0 even in situations where Peer F uses up all its available bandwidth. This is correct, as M and F share the same Internet connection, i.e., F is not entitled to use up all bandwidth. Thus, when M initiates a traffic injection, F's download speed decreases, as part of the DSL connection is used up by the traffic injection. After the download is completed, uB is once again adjusted and the reported available bandwidth returns to previous levels.

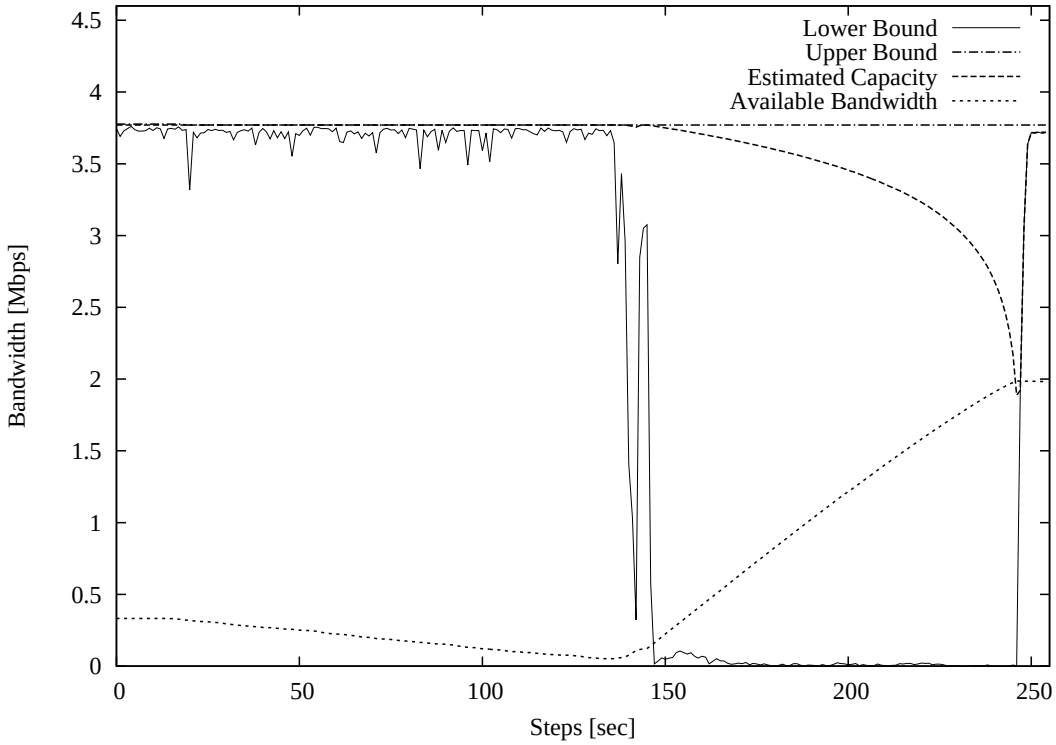

Figure 5.20: Capacity Decay - Behaviour of estimated capacity under high observed traffic

5.3.4 Capacity Decay

Figure 5.20 shows an excerpt of 250 seconds from another experiment in the second scenario, i.e., where background traffic is generated on M. This excerpt shows the behaviour of the capacity decay while significant traffic is generated on the peer. We can see that the estimated capacity C is almost constant and very close to the upper bound uB. This is due to two reasons. First, the high volume of traffic leads to a high lower bound lB, which in turn leads to a threshold which is close to uB. That means that C decreases very little every step. Second, due to the high volume of traffic we regularly reset the decay rate. This means that the high volume of observed traffic allows us to keep C almost constant. This in turns allows us to keep delaying the next traffic injection, reducing overhead under high load (when this overhead can impact system performance). Only when the traffic stops, does C start falling at an increasing rate again, finally leading to a traffic injection as C reaches the threshold.

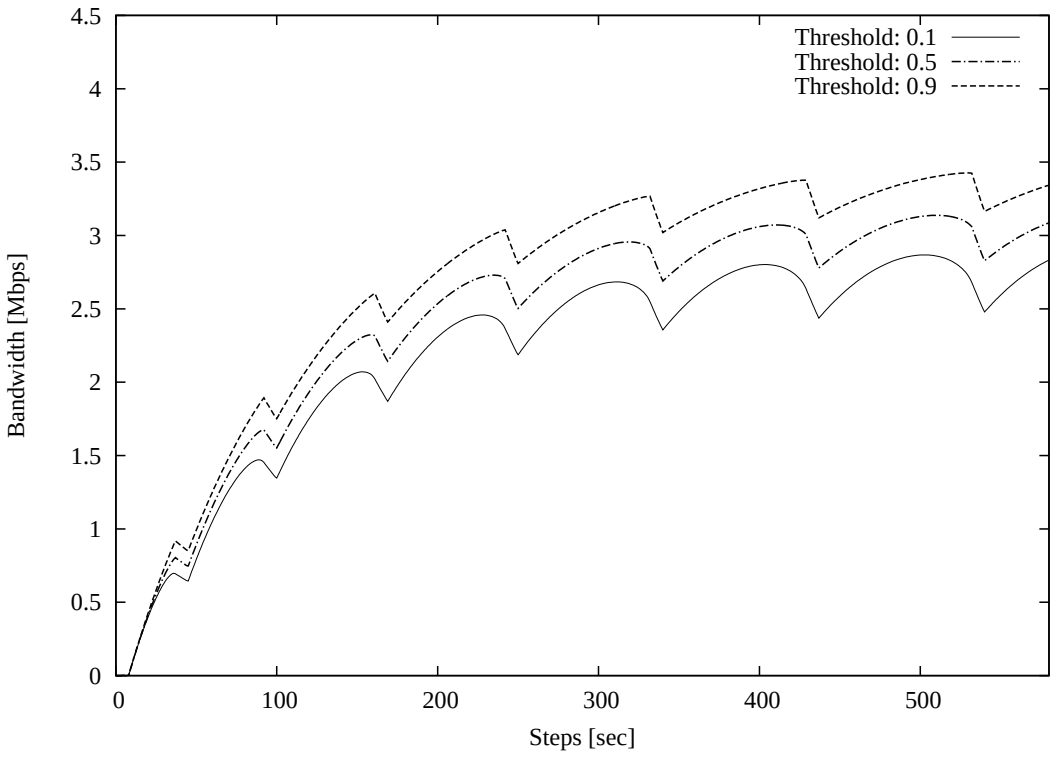

Figure 5.21: Influence of Threshold - Influence of the threshold on reported available bandwidth

5.3.5 Influence of Threshold

Figure 5.21 depicts the influence of the threshold on the reported bandwidth aBW. We have run three experiments in the first scenario, i.e., with no background traffic generated, each with a different setting for the parameter T. T determines the position of the threshold between the upper and lower bounds, which in turns influences the estimated capacity C. With lower values for T we are expecting lower aBWs and vice versa. As we can see, our experiments confirmed this. The higher the threshold, the higher aBW. However, this in turn results in a higher chance of overestimation. This is a trade-off which must be selected by the application. Note that we are reporting aBW above 0 for all three settings of T. This means that even for very low settings of T our approach does not underestimate the available bandwidth to the point where the result cannot be used any more.

5.3.6 Overestimation

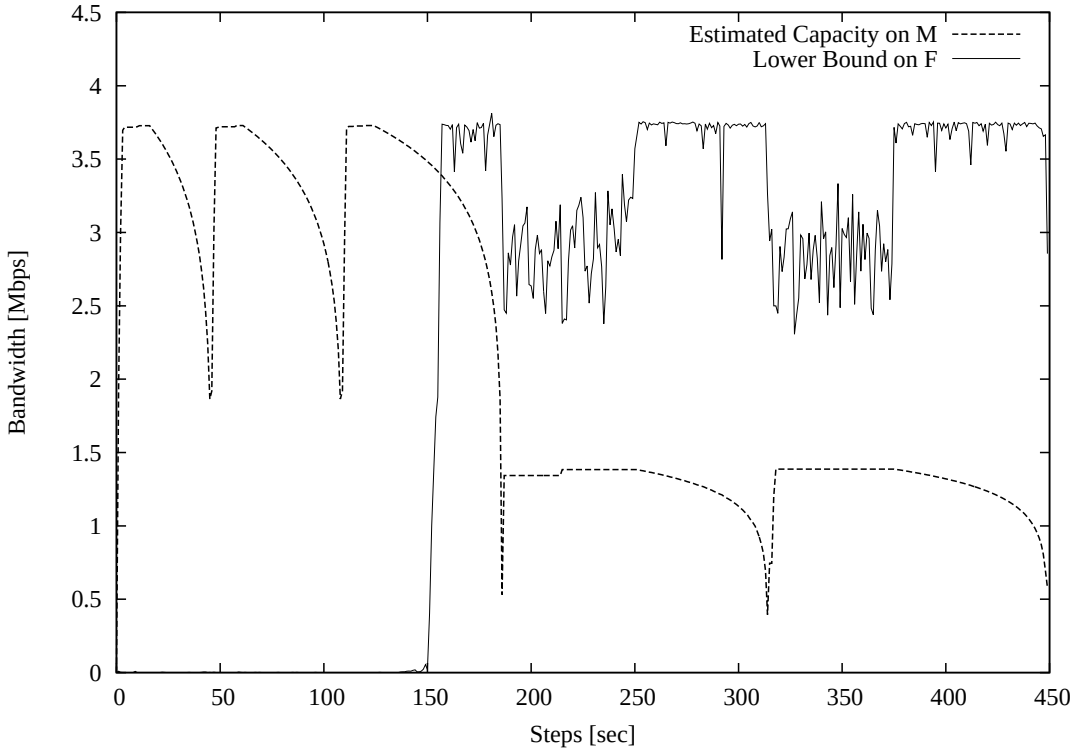

Figure 5.22: Overestimation - Example of overestimation during bandwidth estimation

Our approach cannot completely prevent overestimations. Specifically, overestimation can still occur in between two traffic injections, when the capacity drops below the current estimate. Our decay function is designed to take this into account and thus reduces the estimated capacity C over time. While this always mitigates a possible overestimation, it cannot always completely prevent it. Figure 5.22 shows an example for such a situation. It depicts an excerpt from an experiment in third scenario, i.e., where background traffic is generated on F. The figure shows C on M as well as the traffic generated on F, i.e., the lower bound lB on F. At about 140s, a download is initiated at F. From this point until about 180s we can observe that F is using almost all available bandwidth of the internet connection. However, C on M is initially still estimated at about 3.5 Mbps. During this time, the system suffers from an overestimation. The decay function continuously decreases the amount of overestimation, until C is once again below the actual capacity. Shortly afterwards, another traffic injection is initiated and the upper bound iuB s adjusted, which is reflected by an estimation of C of about 1.4 Mbps. Note that we can also observe the impact that this traffic injection has on the lower bound of F.

Finally, we are looking at the bandwidth overhead generated by our approach. Figure 5.23 shows the total additional bandwidth generated over time. We show the overhead for both the first scenario (no background traffic) and the second scenario (background traffic generated). For the second scenario we have initiated two subsequent downloads of a 175 MB file via BitTorrent.

We can see the reduction in overhead created by the passive observation of the traffic generated in the second scenario. As the decay is reset during this observation, the interval between traffic injections increases, which reduces the overhead.

We also compare our results to a version of the algorithm which uses a static L, set at the middle between L_{min} and L_{max} as $L = \frac{L_{min}+L_{max}}{2}$. Compared to the static approach, the adaptive version initially shows slightly higher overhead, as it starts with a smaller L and thus performs more injections. However, due to stable capacity and high observed traffic both adaptive versions perform better over time as the number of injections is reduced.

Note the result of the static approach does not change when background traffic is introduced, so we omit this result. The result for the adaptive approach in the

third scenario (background traffic initiated on F) is also identical to the result for the adaptive approach in the first scenario, so we omit this as well.

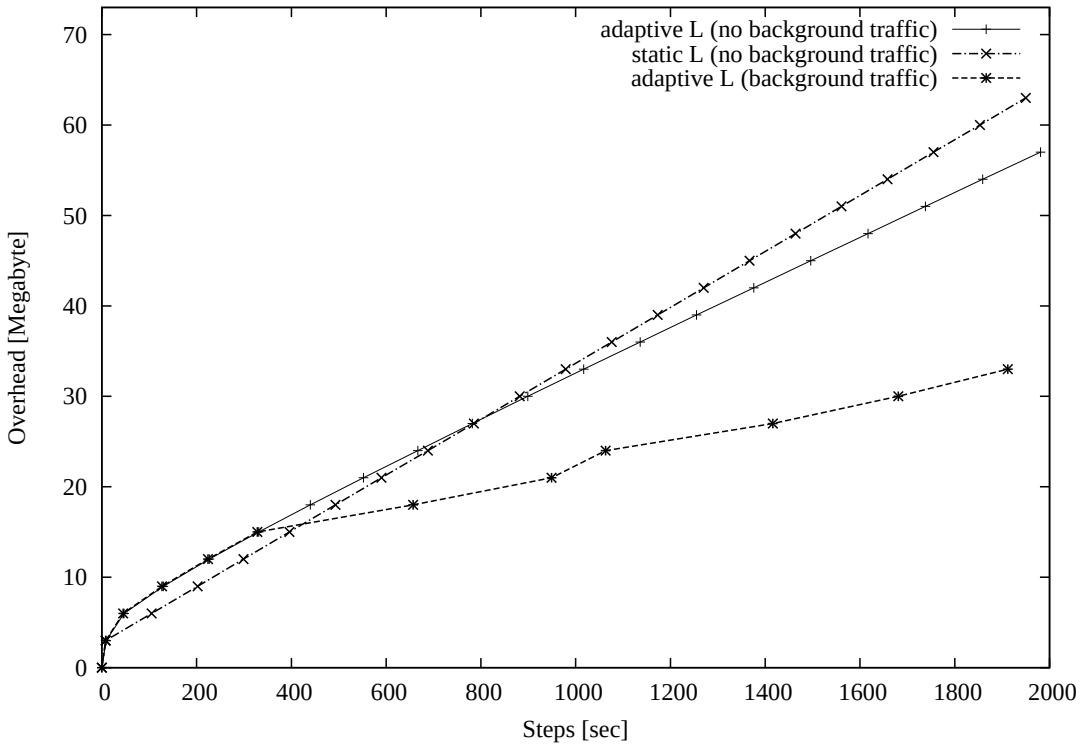

Figure 5.23: Overhead - Overhead generated by the traffic injection

Chapter 6

Conclusion

In this thesis we have developed an update propagation system for peer-to-peer-based massively multi-user virtual environments. The core of this system is a hierarchical update propagation overlay, which consists of a horizontal and a vertical component. The horizontal component uses our concept of an area-of-propagation in order to directly communicate with as many of its neighbour peers as possible. The horizontal component uses special coordinator peers for overlay maintenance and to enable update propagation outside the AoP. To select these coordinators among the massive number of peers in the system, we have developed a coordinator election algorithm, which uses the horizontal component of the overlay to sample a subset of peers in the system. As both creating AoPs and electing coordinators requires knowledge about the available bandwidth of the peers in the system, we have also developed an efficient bandwidth estimation algorithm for P2P systems, which uses a combination of active measurement and passive observation to determine the bandwidth resources each peer can contribute to the system. We will now look at how our update propagation system meets the requirements outlined in Chapter 2 and discuss its remaining limitations.

Our system fulfils the *scalability* requirement through two key mechanisms. First, the horizontal update propagation overlay ensures that individual peers directly communicate with as many neighbour peers as possible, thus making maximum use of their available bandwidth resources. Second, our vertical update propagation overlay allows us to make use of the heterogeneity of the peers' available resources. We can thus assign more resource-intensive tasks to peers with higher available resources. Our quad-tree-based zoning approach also enables us to dynamically assign these tasks to

coordinators based on their available resources and the distribution of the peers in the virtual world.

The scalability of the system is however limited by two factors. First, if the overall system is resource-starved, i.e. most or all peers have very limited available resources, then these techniques are of limited effectiveness. Limited resources on the peers lead to small AoPs, which results in higher load placed on coordinators. As long as there are still enough suitable coordinators available, this is not problematic. However, when no such peers are available, the coordinators cannot handle the additional load and the system may not provide scalability to massive user numbers. Even with sufficient available resources, our system can run into the density problem, i.e. it cannot easily handle a situation where a large number of peers congregate on a single location in the virtual world. This would result in a large number of zone splits and AoPs with very small diameters. This is common to both P2P and server-based approaches and currently an unsolved issue. However, some early research has been done to address this [143], which could potentially be integrated into our system. Despite these limitations, our evaluation shows that our system can potentially scale to massive user numbers.

Our evaluation also shows that our system fulfils the *interactivity* requirement. Most updates can be delivered via the horizontal overlay and thus with one hop in the overlay. This is an improvement over the client/server-approach which always requires two hops for each update. Even when using the vertical overlay, our approach can usually match this, as long as the update is delivered within the same zone where it originates. Higher hop-counts are possible however, when a message is delivered to another zone and thus needs to be forwarded via several coordinators. In practice, given our (realistic) assumption of locality of interest, this is only the case for a small number of updates.

Note that the interactivity of our system is limited by the underlay. If two peers have a one hop connection in the overlay, they may still experience high delay if the underlying network connection is slow. Unlike in other P2P systems (e.g. a file-sharing application), a peer in a P2P-MMVE cannot chose the peers it wants to communicate with. This is due to the fact that its overlay connections are determined by its location in the virtual world. As such we cannot optimize for delay and are limited to providing a low hop count in the overlay. Thus, while we provide good overall interactivity, individual pairs of peers may still experience interactivity problems if their underlay

connection is slow. This is similar to a user experiencing a slow connection to the server in a client/server-based system.

Finally, our system is *self-organizing*, as it dynamically adapts itself to the situation in the virtual world (i.e. the location of the peers and the generated updates) as well as the resources available to the system (i.e. available bandwidth, available CPU and stability). Both the AoPs (the horizontal overlay) and the coordinator quad-tree (the vertical overlay) are constantly adapting themselves to the current situation in order to provide a scalable and interactive system. This adaptation is made possible by our coordinator election algorithm as well as our bandwidth estimation algorithm. The bandwidth estimation algorithm provides the system with the necessary knowledge about the available resources to perform this adaptation, while the election algorithm is required in order to exploit this information for adaptation purposes.

In conclusion, we have developed an update propagation system for peer-to-peer based massively multi-user virtual environments and have shown that it meets the key requirements of scalability, interactivity and self-organization. As an update propagation system is a necessary core component for P2P-based MMVEs, this present a key step towards fully distributed P2P-MMVEs.

6.1 Future Work

While we have developed a fully functioning update propagation system, there are several possible extensions and improvements that could be developed in future work.

First, we have made the assumption that avatars are the only objects which must be managed by the peers. While this is a common simplification in literature, it is not realistic. There are other virtual objects in the world, such as virtual items or computer-controlled characters. These objects can be interacted with by the user, and they must therefore be managed in addition to the avatars. It is however easy to extend our approach to support multiple objects per peer. If a peer is responsible for o objects, we can divide the available bandwidth by o and assign each object an AoP with *availBw = (p.upstream - usedBw) / o*. Further extensions of this approach are also possible. For example we could dynamically assign the available bandwidth (and thus control the size of the AoP) depending on the importance and interaction frequency of the objects. As an example, the peer's avatar may benefit from a larger AoP than

a static object which is rarely interacted with. The placement of objects on the peers also becomes an open problem with this approach. While an avatar object needs to be placed on its user's peer, other objects could be dynamically placed, e.g. on peers close to their location or on those involved in interactions with them. However this must be balanced with the overhead generated by migrating an object.

As mentioned, a key remaining challenge is the so-called density problem. This problem occurs when a large number of avatars congregate in a small area in the virtual world. In this situation, the coordinator of the respective zone experiences high load. Our approach then initiates a zone split. However, as the load was generated by peers near a single location, it will not be evenly distributed among the four new zones. In fact, the zone containing this location will continue to experience high load, leading to continued zone splits. Eventually the load may be balanced, however the respective zones will be very small. This creates large overhead, as even small movements of peers will result in crossing zone boundaries. This is currently still an unsolved problem for P2P-MMVEs, however some initial work into a solution has been conducted (e.g. [143]). Zone transitions can also create notable overhead under other circumstances. Consider a peer which moves in a zig-zag course along a zone border. This means the peer will constantly change its coordinator, thus creating overhead. A potential solution for this problem could be to allow a peer to be part of multiple zones (i.e. be connected to two or more coordinators) when it is near a zone border. This could also reduce the number of multi-hop update propagations when an AoE crosses zone borders.

We have also assumed that coordinators leave the system gracefully. This means they can select a replacement coordinator and hand over all overlay connections to this replacement before leaving the system. This is not a realistic assumption. A simple solution to this issue is to pre-select one or more suitable peers which serve as backup coordinators, maintaining the same information as the primary coordinator (but not doing any calculations or update propagation). If the primary coordinator fails, one of the backup coordinators can take over its duties.

Finally, while we have aimed for a fully distributed P2P system, our approach can also be used in a hybrid approach, as mentioned in Chapter 3. In particular it is possible to use servers as coordinators. In this approach, update propagation would still be largely performed among the peers in the horizontal overlay, thus significantly

reducing server load compared to a fully server-based system. However due to the higher available resources of the servers, zones could be larger and multi-hop event delivery in the vertical overlay could be faster as these servers could be co-located leading to very low delay for each hop. A simple way to implement a hybrid version of our system would be to let the servers enter the system as inactive peers. Due to their high suitability score, they would eventually be selected as coordinators. However, as our election algorithm only samples a subset of all available peers, this could not guarantee that all servers would be selected as coordinator. A better solution would be to replace the election algorithm with a query to a central server, which then returns a suitable server to serve as coordinator.

Finally, while update propagation is a key step towards P2P-MMVEs, several other issues such as consistency, security and persistency need to be solved before such systems can be deployed commercially. Research on these issues is ongoing.

References

[1] KZero Worldswide. **Virtual world registered accounts reach 17bn in Q4 2011**. http://www.kzero.co.uk/blog/virtual-world-registered-accounts-reach-1-7bn-q4-2011/, January 2012. Last retrieved 08.05.2013. 1, 2, 5

[2] ABI Research. **Gaming in the cloud - online gaming for pcs, video game consoles, and connected devices**. Research Report, November 2010. 1

[3] Inc. ArenaNet. **Guild Wars 2**. https://www.guildwars2.com/, August 2012. Last retrieved 08.05.2013. 2, 5

[4] A. Hendaoui, M. Limayem, and C.W. Thompson. **3D social virtual worlds: research issues and challenges**. *IEEE Internet Computing*, **12**(1):88 –92, jan.-feb. 2008. 2

[5] Ned Kock. **E-collaboration and E-commerce in virtual worlds: the potential of second life and world of warcraft**. *International Journal of eCollaboration (IJeC)*, **4**(3):–, 2008. 2

[6] Maged N. Kamel Boulos, Lee Hetherington, and Steve Wheeler. **Second Life: an overview of the potential of 3-D virtual worlds in medical and health education**. *Health Information & Libraries Journal*, **24**(4):233–245, 2007. 2

[7] Blake Ives and Iris Junglas. **Business implications of virtual worlds and serious gaming**. *MIS Quarterly Executive*, **7**(3). 2

[8] William Sims Bainbridge. **The scientific research potential of virtual worlds**. *Science*, **317**(5837):472–476, 2007. 2

[9] Sulake Corporation. **Habbo Hotel**. http://www.habbo.com/. Last retrieved 08.05.2013. 2

[10] Blizzard Entertainment Inc. **World of Warcraft**. http://www.battle.net/wow, 2004. Last retrieved 08.05.2013. 2, 4, 5

[11] Adam Holisky. **World of Warcraft subscriber numbers dip 100,000 to 10.2 million**. http://wow.joystiq.com/2012/02/09/world-of-warcraft-subscriber-numbers/, February 2012. Last retrieved 08.05.2013. 2, 4

[12] Richard Bartle. *Designing Virtual Worlds*. New Riders, 2003. 2

[13] CCP Games Inc. **60,453 pilots: the new eve pcu record**. http://community.eveonline.com/news.asp?a=single&nid=3934&tid=1, 06 2010. Last retrieved 08.05.2013. 3, 16

REFERENCES

[14] Peter Quax, Patrick Monsieurs, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. **Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game**. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games (NetGames04)*, NetGames '04, pages 152–156, New York, NY, USA, 2004. ACM. 4, 15

[15] Chip Morningstar and F. Randall Farmer. *Cyberspace: First Steps*, chapter The Lessons of Lucasfilm's Habitat, pages 273–302. The MI, 1991. 4

[16] Stormfront Studios. **Neverwinter Nights**. http://www.stormfront.com/, 1991. Last retrieved 08.05.2013. 4

[17] Justin Olivetti. **The game archaeologist moves into lucasfilms habitat**. http://www.nextmmorpg.com/the-game-archaeologist-moves-into-lucasfilms-habitat-part-1, January 2012. Last retrieved 08.05.2013. 4

[18] Janelle Brown. **Gamers Claim AOL Is Playing Bait-and-Switch**. http://www.wired.com/culture/lifestyle/news/1997/06/4625, June 1997. Last retrieved 08.05.2013. 4

[19] Holin Lin and Chuen-Tsai Sun. **Cash trade within the magic circle: free-to-play game challenges and massively multiplayer online game player responses**. In Baba Akira, editor, *Situated Play: Proceedings of the 2007 Digital Games Research Association Conference*, pages 335–343, Tokyo, September 2007. The University of Tokyo. 4

[20] Near Death Studios. **Meridian 59**. http://www.meridian59.com/, December 1995. Last retrieved 08.05.2013. 4

[21] Origin Systems. **Ultima Online**. http://www.uoherald.com/, September 1997. Last retrieved 08.05.2013. 4

[22] BioWare. **Star Wars: The Old Republic**. http://www.swtor.com/, December 2011. Last retrieved 08.05.2013. 4, 5

[23] Turbine, Inc. **Lord of the Rings Online**. http://www.lotro.com, April 2007. Last retrieved 08.05.2013. 4, 5

[24] Turbine, Inc. **Dungeons and Dragons Online**. http://www.ddo.com, February 2006. Last retrieved 08.05.2013. 4

[25] Funcom Productions A/S. **Age of Conan: Hyborian Adventures**. http://www.ageofconan.com/, May 2008. Last retrieved 08.05.2013. 4

[26] Kyle Orland. **Turbine: Lord of the Rings Online Revenues Tripled As Free-To-Play Game**. http://www.gamasutra.com/view/news/32322/Turbine_Lord_of_the_Rings_Online_Revenues_Tripled_As_FreeToPlay_Game.php, January 2011. Last retrieved 08.05.2013. 4

[27] Leigh Alexander. **Going Free Boosts Turbine's DDO Revenues 500 Percent**. http://www.gamasutra.com/view/news/27416/Going_Free_Boosts_Turbines_DDO_Revenues_500_Percent.php, February 2010. Last retrieved 08.05.2013. 4

[28] Sony Online Entertainment. **Free Realms**. http://www.freerealms.com/, November 2010. Last retrieved 08.05.2013. 4

[29] Nickelodeon Kids & Family Virtual Worlds Group. **Neopets**. http://www.neopets.com/, November 1999. Last retrieved 08.05.2013. 4

[30] Linden Lab. **Second Life**. http://secondlife.com/, 2003. Last retrieved 08.05.2013. 5

[31] Edward Castronova. **On Virtual Economies**. *SSRN eLibrary*, 2002. 5

[32] James Wagner Au. **Top second life entrepeneur cashing out us 1.7 million yearly; furnishings, events management among top earners**. http://nwn.blogs.com/nwn/2009/03/million.html, March 2009. Last retrieved 08.05.2013. 5

[33] Funcom Productions A/S. **The Secret World**. http://www.thesecretworld.com/, July 2012. Last retrieved 08.05.2013. 5

[34] Raph Koster. **On "Pay To Play" Or, MMORPG Business Models 101**. http://www.raphkoster.com/gaming/busmodels.shtml. Last retrieved 08.05.2013. 5

[35] William Murphy. **The worst MMORPG launches**. http://www.mmorpg.com/showFeature.cfm/loadFeature/4862, January 2011. Last retrieved 08.05.2013. 6

[36] Gregor Schiele, Richard Süselbeck, Arno Wacker, Tonio Triebel, and Christian Becker. **Consistency management for peer-to-peer-based massively multiuser virtual environments**. In *Proceedings of the the 1st International Workshop on Massively Multiuser Virtual Environments (MMVE08)*, 2008. 7, 17

[37] Laura Itzel, Verena Tuttlies, Gregor Schiele, and Christian Becker. **Consistency management for interactive peer-to-peer-based systems**. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SimuTools10)*, SIMUTools '10, pages 1:1–1:8, ICST, Brussels, Belgium, Belgium, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 7, 17

[38] Laura Itzel, Richard Süselbeck, Gregor Schiele, and Christian Becker. **Specifying consistency requirements for massively multi-user virtual environments**. In *Proceedings of the 4th International Workshop on Massively Multiuser Virtual Environments (MMVE11)*, 2011. 7

[39] Sebastian Schuster, Arno Wacker, and Torben Weis. **Fighting cheating in P2P-based MMVEs with disjoint path routing**. *ECEASST*, **17**:1, 2009. 7

[40] Sebastian Schuster and Torben Weis. **Enforcing game rules in untrusted P2P-based mmves**. In *Proceedings of the 2nd Workshop on DIstributed SImulation & Online Gaming (DISIO2011)*, 2011. 7

REFERENCES

[41] J.S. Gilmore and H.A. Engelbrecht. **Pithos: a state persistency architecture for peer-to-peer massively multiuser virtual environments**. In *Proceedings of the 2011 IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE11)*, pages 1 –6, oct. 2011. 7

[42] G. Schiele, R. Süselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis. **Requirements of peer-to-peer-based massively multiplayer online gaming**. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID07)*, pages 773 –782, may 2007. 7, 11, 15

[43] **The peers@play project**. http://www.peers-at-play.org/. Last retrieved 08.05.2013. 13

[44] Sebastian Holzapfel, Arno Wacker, Torben Weis, and Matthäus Wander. **An architecture for complex peer-to-peer systems**. In *Proceedings of the 4th IEEE International Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology (DENVECT12)*, Las Vegas, Nevada, USA, January 2012. USA January 14-17, 2012(to appear). 14, 17

[45] Bryan Ford, Pyda Srisuresh, and Dan Kegel. **Peer-to-peer communication across network address translators**. In *Proceedings of the USENIX Annual Technical Conference (ATC05)*, ATEC '05, pages 13–13, Berkeley, CA, USA, 2005. USENIX Association. 14

[46] S. Holzapfel, M. Wander, A. Wacker, L. Schwittmann, and T. Weis. **A new protocol to determine the nat characteristics of a host**. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW11)*, pages 1651 –1658, may 2011. 14

[47] M. Knoll, A. Wacker, G. Schiele, and T. Weis. **Bootstrapping in peer-to-peer systems**. In *Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS08)*, pages 271 –278, dec. 2008. 14

[48] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. **Chord: A scalable peer-to-peer lookup service for internet applications**. In *Proceedings of the 2001 Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM01)*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM. 14, 24

[49] M. Saternus, T. Weis, S. Holzapfel, and A. Wacker. **Gears4Netet an asynchronous programming model**. In *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW09)*, pages 113 –120, sept. 2009. 15

[50] M. Saternus. *Gears4Net: Ein Programmiermodell für asynchrone Kommunikation in P2P-Systemen*. Südwestdeutscher Verlag für Hochschulschriften, 2011. 15

[51] Lu Fan, Phil Trinder, and Hamish Taylor. **Design issues for peer-to-peer massively multiplayer online games**. *International Journal on Advanced Media Communication*, **4**:108–125, March 2010. 15

[52] Mark Claypool and Kajal Claypool. **Latency and player actions in online games**. *Communications of the ACM*, **49**:40–45, November 2006. 15

[53] G. Armitage. **An experimental estimation of latency sensitivity in multiplayer Quake 3**. In *Proceedings of the 11th IEEE International Conference on Networks (ICON03)*, pages 137 – 141, sept.-1 oct. 2003. 15

[54] Kuan-Ta Chen, Polly Huang, and Chin-Laung Lei. **How sensitive are online gamers to network quality?** *Communications of the ACM*, **49**(11):34–38, November 2006. 15

[55] Tristan Henderson and Saleem Bhatti. **Networked games: a QoS-sensitive application for QoS-insensitive users?** In *Proceedings of the ACM SIGCOMM Workshop on Revisiting IP QoS: What have we learned, why do we care?*, RIPQoS '03, pages 141–147, New York, NY, USA, 2003. ACM. 15

[56] CCP Games Inc. **Eve Online**. http://www.eveonline.com/. Last retrieved 08.05.2013. 16

[57] Brendan Drain. **EVE Evolved: EVE Online's server model**. http://massively.joystiq.com/2008/09/28/eve-evolved-eve-onlines-server-model/, September 2008. Last retrieved 08.05.2013. 16

[58] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. **VON: a scalable peer-to-peer network for virtual environments**. *IEEE Network*, **20**(4):22 –31, july-aug. 2006. 16

[59] J. Keller and G. Simon. **Toward a peer-to-peer shared virtual reality**. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (DCS02)*, pages 695 – 700, 2002. 20

[60] Joaquín Keller and Gwendal Simon. **Solipsis: a massively multi-participant virtual world**. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA03)*, pages 262–268, 2003. 20

[61] Shun-Yun Hu and Guan-Ming Liao. **Scalable peer-to-peer networked virtual environment**. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games (NetGames04)*, NetGames '04, pages 129–133, New York, NY, USA, 2004. ACM. 20, 21

[62] Franz Aurenhammer. **Voronoi diagrams - a survey of a fundamental geometric data structure**. *ACM Computing Surveys*, **23**(3):345–405, September 1991. 20

[63] M. Ohnishi, R. Nishide, and S. Ueshima. **Incremental construction of Delaunay overlaid network for virtual collaborative space**. In *Proceedings of the 3rd International Conference on Creating, Connecting and Collaborating through Computing (C505)*, pages 75 – 82, jan. 2005. 21, 22

[64] Laura Ricci and Andrea Salvadori. **Nomad: virtual environments on P2P voronoi overlays**. In *Proceedings of the 2007 Confederated international Conference on On the Move to Meaningful Internet Systems (OTM07)*, OTM'07, pages 911–920, Berlin, Heidelberg, 2007. Springer-Verlag. 21, 22

REFERENCES

[65] E. Buyukkaya, M. Abdallah, and R. Cavagna. **Vorogame: a hybrid p2p architecture for massively multiplayer games**. In *Proceedings of the 6th IEEE Consumer Communications and Networking Conference (CCNC06)*, pages 1 –5, jan. 2009. 21, 22

[66] L. Genovali and L. Ricci. **AOI-cast strategies for P2P massively multiplayer online games**. In *Proceedings of the 6th IEEE Consumer Communications and Networking Conference (CCNC09)*, pages 1 –5, jan. 2009. 21

[67] Davide Frey, Jrme Royan, Romain Piegay, Anne marie Kermarrec, Fabrice Le Fessant, and Emmanuelle Anceaume. **Solipsis: A decentralized architecture for virtual environments**. In *Proceedings of the 1st International Workshop on Massively Multiuser Virtual Environments (MMVE08)*, 2008. 21, 22

[68] Jui-Fa Chen, Wei-Chuan Lin, Tsu-Han Chen, and Shun-Yun Hu. **A forwarding model for voronoi-based overlay network**. In *Proceedings of the 13th International Conference on Parallel and Distributed Systems (ICPADS07)*, **2**, pages 1 –7, dec. 2007. 21

[69] Jehn-Ruey Jiang, Yu-Li Huang, and Shun-Yun Hu. **Scalable AOI-cast for peer-to-peer networked virtual environments**. In *Proceedings of the 28th International Conference on Distributed Computing Systems Workshops (ICDCS08)*, pages 447 –452, june 2008. 21

[70] H. Kato, T. Eguchi, M. Ohnishi, and S. Ueshima. **Autonomous generation of spherical P2P delaunay network for global internet applications**. In *Proceedings of the 4th International Conference on Creating, Connecting and Collaborating through Computing (C506)*, pages 184 –191, jan. 2006. 21, 22

[71] Anthony (Peiqun) Yu and Son T. Vuong. **MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games**. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV05)*, NOSSDAV '05, pages 99–104, New York, NY, USA, 2005. ACM. 21, 23, 24, 25, 26, 28

[72] M. Ohnishi, S. Tsuboi, M. Hirayama, T. Eguchi, and S. Ueshima. **Distributive generation algorithm of long range contact for remote spatial-data access on P2P delaunay network**. In *Proceedings of the 5th International Conference on Creating, Connecting and Collaborating through Computing (C507)*, pages 145–152, 2007. 22

[73] S. Tsuboi, T. Oku, M. Ohnishi, and S. Ueshima. **Generating skip delaunay network for P2P geocasting**. In *Proceedings of the 6th International Conference on Creating, Connecting and Collaborating through Computing (C508)*, pages 179 –186, jan. 2008. 22

[74] Ryo Nishide, Dai Ito, Masaaki Ohnishi, and Shinichi Ueshima. **Data aggregation method for view range computation on P2P-based VCS**. In *Proceedings of the 1st International Workshop on Massively Multiuser Virtual Environments (MMVE08)*, 2008. 22

[75] A. Schmieg, M. Stieler, S. Jeckel, P. Kabus, B. Kemme, and A. Buchmann. **pSense - Maintaining a dynamic localized peer-to-peer structure for position based multicast in games**. In *Proceedings of the 8th International Conference on Peer-to-Peer Computing (P2P08)*, pages 247 –256, sept. 2008. 22

[76] Helge Backhaus and Stephan Krause. **QuON: a quad-tree-based overlay protocol for distributed virtual worlds**. *International Journal of Advanced Media and Communication*, **4**:126–139, March 2010. 22

[77] Y. Kawahara, H. Morikawa, and T. Aoyama. **A peer-to-peer message exchange scheme for large scale networked virtual environments**. In *Proceedings of the The 8th International Conference on Communication Systems (ICCS03)*, pages 957–961, Washington, DC, USA, 2003. IEEE Computer Society. 23

[78] N. Matsumoto, Y. Kawahara, H. Morikawa, and T. Aoyama. **A scalable and low delay communication scheme for networked virtual environments**. In *Proceedings of the Global Telecommunications Conference Workshops (GLOBECOM04)*, pages 529 – 535, nov.-3 dec. 2004. 23

[79] P. Morillo, W. Moncho, J. M. Ordua, and J. Duato. **Providing full awareness to distributed virtual environments based on peer-to-peer architectures**. In *Lecture Notes on Computer Science*, page 2006, 2006. 23, 28, 43

[80] Shun-Yun Hu, Shao-Chen Chang, and Jehn-Ruey Jiang. **Voronoi state management for peer-to-peer massively multiplayer online games**. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference (CCNC08)*, pages 1134 –1138, jan. 2008. 23

[81] Shun-Yun Hu, Chuan Wu, E. Buyukkaya, Chien-Hao Chien, Tzu-Hao Lin, M. Abdallah, Jehn-Ruey Jiang, and Kuan-Ta Chen. **A spatial publish subscribe overlay for massively multiuser virtual environments**. In *Proceedings of the International Conference on Electronics and Information Engineering (ICEIE2010)*, **2**, pages V2–314 –V2–318, aug. 2010. 23, 28

[82] M. Albano, L. Ricci, and L. Genovali. **Hierarchical P2P overlays for DVE: an additively weighted voronoi based approach**. In *Proceedings of the International Conference on Ultra Modern Telecommunications Workshops (ICUMT09)*, pages 1 –8, oct. 2009. 23

[83] B. Knutsson, Honghui Lu, Wei Xu, and B. Hopkins. **Peer-to-peer support for massively multiplayer games**. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM04)*, **1**, pages 4 vol. (xxxv+2866), march 2004. 24, 28, 43

[84] Antony Rowstron and Peter Druschel. **Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems**. In Rachid Guerraoui, editor, *Middleware 2001*, **2218** of *Lecture Notes in Computer Science*, pages 329–350. Springer Berlin / Heidelberg, 2001. 24

[85] M. Castro, P. Druschel, A.-M. Kermarrec, and A.I.T. Rowstron. **Scribe: a large-scale and decentralized application-level multicast infrastructure**. *IEEE Journal on Selected Areas in Communications*, **20**(8):1489 – 1499, oct 2002. 24

REFERENCES

[86] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. **A scalable content-addressable network**. *SIGCOMM Computer Communication Review*, **31**(4):161–172, August 2001. 24, 26

[87] Ashwin Bharambe, Jeffrey Pang, and Srinivasan Seshan. **Colyseus: a distributed architecture for online multiplayer games**. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation (NSDI06)*, NSDI'06, pages 12–12, Berkeley, CA, USA, 2006. USENIX Association. 24

[88] Scott Douglas, Egemen Tanin, Aaron Harwood, and Shanika Karunasekera. **Enabling massively multi-player online gaming applications on a P2P architecture**. In *Proceedings of the IEEE International Conference on Information and Automation (ICIA05)*, pages 7–12. IEEE, 2005. 24

[89] Egemen Tanin, Aaron Harwood, Hanan Samet, Sarana Nutanong, and Minh Tri Truong. **A serverless 3D world**. In *Proceedings of the 12th annual ACM International Workshop on Geographic Information Systems (GIS04)*, GIS '04, pages 157–165, New York, NY, USA, 2004. ACM. 24

[90] Takuji Iimura, Hiroaki Hazeyama, and Youki Kadobayashi. **Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games**. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and Ssystem Support for Games (NetGames04)*, NetGames '04, pages 116–120, New York, NY, USA, 2004. ACM. 24, 28, 42, 43

[91] Luther Chan, James Yong, Jiaqiang Bai, Ben Leong, and Raymond Tan. **Hydra: a massively-multiplayer peer-to-peer architecture for the game developer**. In *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames07)*, NetGames '07, pages 37–42, New York, NY, USA, 2007. ACM. 25, 42

[92] Shinya Yamamoto, Yoshihiro Murata, Keiichi Yasumoto, and Minoru Ito. **A distributed event delivery method with load balancing for MMORPG**. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames05)*, NetGames '05, pages 1–8, New York, NY, USA, 2005. ACM. 25, 28, 42

[93] D.T. Ahmed, S. Shirmohammadi, and J.C. de Oliveira. **A novel method for supporting massively multi-user virtual environments**. In *Proceedings of the IEEE International Workshop on Haptic Audio Visual Environments and their Applications (HAVE06)*, pages 72 –77, nov. 2006. 25, 28, 42

[94] M. Merabti and A. El Rhalibi. **Peer-to-peer architecture and protocol for a massively multiplayer online game**. In *Proceedings of the 2004 Global Telecommunications Conference Workshops (GLOBECOM04)*, pages 519 – 528, nov.-3 dec. 2004. 26

[95] Abdennour El Rhalibi and Madjid Merabti. **Agents-based modeling for a peer-to-peer MMOG architecture**. *Computer Entertainment*, **3**:3–3, April 2005. 26

[96] A.E. Rhalibi and M. Merabti. **Interest management and scalability issues in P2P MMOG**. In *Proceedings of the 3rd IEEE Consumer Communications and Networking Conference (CCNC06)*, **2**, pages 1188 – 1192, jan. 2006. 26

[97] Hsiu-Hui Lee and Chin-Hua Sun. **Load-balancing for peer-to-peer networked virtual environment**. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames06)*, NetGames '06, New York, NY, USA, 2006. ACM. 26, 28

[98] Alvin Chen and Richard R. Muntz. **Peer clustering: a hybrid approach to distributed virtual environments**. In *Proceedings of 5th ACM SIGCOMM workshop on Network and System Support for Games (NetGames06)*, NetGames '06, New York, NY, USA, 2006. ACM. 26, 28

[99] Kyungchul Kim, Ikjun Yeom, and Joonwon Lee. **HYMS: A Hybrid MMOG Server Architecture**. In *IEICE Transactions on Information and Systems, vol.E87-D, no.12*, 2004. 26, 43

[100] Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Leo Petrak, and Georg Carle. **Peer-to-peer-based infrastructure support for massively multiplayer online games**. In *Proceedings of the 4th IEEE Consumer Communications and Networking Conference (CCNC07)*, pages 763 –767, jan. 2007. 26

[101] Jean Botev, Alexander Hohfeld, Hermann Schloss, Ingo Scholtes, Peter Sturm, and Markus Esch. **The HyperVerse: concepts for a federated and torrent-based '3D Web'**. *International Journal on Advanced Media Communication*, **2**:331–350, December 2008. 26

[102] M. Esch and E. Tobias. **Decentralized scale-free network construction and load balancing in massive multiuser virtual environments**. In *Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom10)*, pages 1 –10, oct. 2010. 26

[103] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996. 27

[104] H. Garcia-Molina. **Elections in a distributed computing system**. *IEEE Transactions on Computers*, **C-31**(1):48 –59, jan. 1982. 27

[105] Ernest Chang and Rosemary Roberts. **An improved algorithm for decentralized extrema-finding in circular configurations of processes**. *Communications of the ACM*, **22**(5):281–283, May 1979. 27

[106] Alon Itai and Michael Rodeh. **Symmetry breaking in distributed networks**. *Information and Computation*, **88**:150–158, 1981. 27

[107] Israel Cidon and Osnat Mokryn. **Propagation and leader election in a multihop broadcast environment**. In *Proceedings of the 12th International Symposium on Distributed Computing (DISC98)*, pages 104–118. Springer-Verlag, 1998. 27

[108] Navneet Malpani, Jennifer L. Welch, and Nitin Waidya. **Leader election algorithms for mobile ad hoc networks**. pages 96–103, 2000. 27

[109] V. Raychoudhury, Jiannong Cao, and Weigang Wu. **Top k-leader election in wireless ad hoc networks**. In *Proceedings of 17th International Conference on Computer Communications and Networks 2008 (ICCCN08)*, pages 1–6, 2008. 27

[110] Sung-Hoon Park, Tae-Gyu Lee, Hyung-Seok Seo, Seok-Jin Kwon, and Jong-Ho Han. **An election protocol in mobile ad hoc distributed systems**. In *Proceedings of the 6th International Conference on Information Technology: New Generations, 2009. (ITNG 09)*, pages 628–633, 2009. 27

[111] S. Vasudevan, J. Kurose, and D. Towsley. **Design and analysis of a leader election algorithm for mobile ad hoc networks**. In *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP04)*, pages 350–360, 2004. 27

[112] Chris GauthierDickey, Virginia Lo, and Daniel Zappala. **Using n-trees for scalable event ordering in peer-to-peer games**. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDV05)*, NOSSDAV '05, pages 87–92, New York, NY, USA, 2005. ACM. 28

[113] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. **Bandwidth estimation: metrics, measurement techniques, and tools**. *IEEE Network*, **17**(6):27 – 35, nov.-dec. 2003. 28

[114] Karthik Lakshminarayanan, Venkata N. Padmanabhan, and Jitendra Padhye. **Bandwidth estimation in broadband access networks**. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC04)*, IMC '04, pages 314–321, New York, NY, USA, 2004. ACM. 29

[115] B. Melander, M. Bjorkman, and P. Gunningberg. **A new end-to-end probing and analysis method for estimating bandwidth bottlenecks**. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM00)*, 2000. 29

[116] Kevin Lai and Mary Baker. **Nettimer: a tool for measuring bottleneck link, bandwidth**. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS01)*, USITS'01, Berkeley, CA, USA, 2001. USENIX Association. 29

[117] M. Jain and C. Dovrolis. **Pathload: a measurement tool for end-to-end available bandwidth**. In *Proceedings of the Passive and Active Measurement Workshop (PAM12)*, March 2002. 29

[118] Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, and Les Cottrell. **Pathchirp: efficient available bandwidth estimation for network paths**. In *Proceedings of the Passive and Active Measurement Workshop (PAM03)*, 2003. 29

[119] Robin Snader and Nikita Borisov. **EigenSpeed: secure peer-to-peer bandwidth evaluation**. In *Proceedings of the 8th International Conference on Peer-to-Peer systems (P2P09)*, IPTPS'09, pages 9–9, Berkeley, CA, USA, 2009. USENIX Association. 29

[120] John R. Douceur, James W. Mickens, Thomas Moscibroda, and Debmalya Panigrahi. **ThunderDome: discovering upload constraints using decentralized bandwidth tournaments**. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 193–204, New York, NY, USA, 2009. ACM. 29

[121] J.R. Douceur, J. Mickens, T. Moscibroda, and D. Panigrahi. **Collaborative Measurements of Upload Speeds in P2P Systems**. In *Proceedings of the 29th Conference on Computer Communications (INFOCOM10)*, pages 1 –9, march 2010. 29

[122] Katherine L. Morse, Lubomir Bic, and Michael Dillencourt. **Interest management in large-scale virtual environments**. *Presence: Teleoperators and Virtual Environments*, **9**(1):52–68, February 2000. 34

[123] Richard Süselbeck, Gregor Schiele, Sebastian Seitz, and Christian Becker. **Adaptive update propagation for low-latency massively multi-user virtual environments**. In *Proceedings of 18th International Conference on Computer Communications and Networks (ICCCN09)*, pages 1 –6, aug. 2009. 34

[124] Florian Heger, Gregor Schiele, Richard Süselbeck, and Christian Becker. **Towards an interest management scheme for peer-based virtual environments**. In *Proceedings of the 1st International Workshop on Concepts of Massively Multiuser Virtual Environments (COMMVE09)*, 2009. 35

[125] S. Rooney, D. Bauer, and R. Deydier. **A federated peer-to-peer network game architecture**. *IEEE Communications Magazine*, **42**(5):114 – 122, may 2004. 43

[126] Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Timo Teifel, and Georg Carle. **Clustering players for load balancing in virtual worlds**. *International Journal on Advanced Media Communication*, **2**:351–363, December 2008. 43

[127] R. Süselbeck, F. Heger, G. Schiele, and C. Becker. **Challenges for selecting optimal coordinators in peer-to-peer-based massively multi-user virtual environments**. In *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN11)*, pages 1 –6, 31 2011-aug. 4 2011. 53, 54, 69

[128] Daniel Pittman and Chris GauthierDickey. **A measurement study of virtual populations in massively multiplayer online games**. In *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames07)*, NetGames '07, pages 25–30, New York, NY, USA, 2007. ACM. 55

[129] Robert L. Cook. **Stochastic sampling in computer graphics**. *ACM Transactions on Graphics*, **5**(1):51–72, January 1986. 62

[130] Robert Bridson. **Fast Poisson disk sampling in arbitrary dimensions**. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM. 62

[131] B. Melander, M. Bjorkman, and P. Gunningberg. **A new end-to-end probing and analysis method for estimating bandwidth bottlenecks**. In *Proceedings of the 19th IEEE Global Telecommunications Conference (GLOBECOM00)*, 2000. 68

[132] Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, and Les Cottrell. **Pathchirp: efficient available bandwidth estimation for network paths**. In *Proceedings of the Passive and Active Measurement Workshop (PAM03)*, 2003. 68

REFERENCES

[133] Karthik Lakshminarayanan, Venkata N. Padmanabhan, and Jitendra Padhye. **Bandwidth estimation in broadband access networks**. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC04)*, pages 314–321, New York, NY, USA, 2004. ACM. 68

[134] M. Jain and C. Dovrolis. **Pathload: a measurement tool for end-to-end available bandwidth**. In *Proceedings of the Passive and Active Measurement Workshop (PAM02)*, 2002. 68

[135] Kevin Lai and Mary Baker. **Nettimer: a tool for measuring bottleneck link, bandwidth**. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS03)*, pages –, Berkeley, CA, USA, 2001. USENIX Association. 68

[136] BitTorrent, Inc. **Torrent 3.0 (32-bit) Stable Release**. http://forum.utorrent.com/viewtopic.php?id=98500, April 2011. Last retrieved 08.05.2013. 68

[137] Ofcom. **UK broadband speeds: The performance of fixed-line broadband delivered to UK residential consumers**. Technical report, Ofcom, 2010. 68, 71

[138] Bernardo A. Huberman Eyta Adar. **Free riding on Gnutella**. *First Monday [Online]*, **Volume 5 Number 10**:1, 2000. 68

[139] Bandwidth Place. **Bandwidth Place**. http://www.bandwidthplace.com/, 2012. Last retrieved 08.05.2013. 68

[140] Ookla. **Speedtest.net**. http://www.speedtest.net/, 2013. Last retrieved 08.05.2013. 68

[141] Richard Süselbeck, Gregor Schiele, Patricius Komarnicki, and Christian Becker. **Efficient bandwidth estimation for peer-to-peer systems**. In *Proceedings of the 11th IEEE International Conference on Peer-to-Peer Computing (P2P11)*, 2011. 69

[142] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. **Measuring and analyzing the characteristics of Napster and Gnutella hosts**. *Multimedia Systems*, **9**:170–184, 2003. 10.1007/s00530-003-0088-1. 71

[143] Ashwin Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, and Xinyu Zhuang. **Donnybrook: enabling large-scale, high-speed, peer-to-peer games**. In *Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM08)*, SIGCOMM '08, pages 389–400, New York, NY, USA, 2008. ACM. 108, 110

Curriculum Vitae

seit 01/2013	**Senior Developer Evangelist** Deutsche Telekom AG
10/2006 - 11/2012	**Akademischer Mitarbeiter** Lehrstuhl für Wirtschaftsinformatik II Prof. Dr. Christan Becker Universität Mannheim
10/1999 - 04/2006	**Studium der Informatik (Diplom)** Technische Universität Dortmund Abschluss: Diplom-Informatiker
07/1997 - 09/1999	**Studium der Wirtschaftsinformatik (Diplom)** Westfälische Wilhelms-Universität Münster
06/1997	**Abitur** Ratsgymnasium Münster