A Comparative Study of Interest Management Schemes in Peer-to-Peer Massively Multiuser Networked Virtual Environments

by

Tielman Francois Septimus Malherbe

Thesis presented for the degree of Masters in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr. H. A. Engelbrecht

December 2016

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety of in part submitted it for obtaining any qualifications.

Date: December 2016 .....

Abstract

Networked virtual environments (NVEs) are virtual worlds where users communicate and interact with others and the virtual world by exchanging messages via a network connection. An NVE user is often only able to interact, or only interested in interacting, with objects and players within a sphere around their position; the area of interest (AOI). Enabling users to properly identify and communicate with objects of interest while limiting the amount of information received to only relevant information is the purpose an interest management (IM) scheme. In this thesis, three existing peer-to-peer (P2P) interest management schemes; SimMud, Vast and MOPAR; and one naive messaging implementation, FullConn, are tested. These schemes are simulated in a virtual game world consisting of varying population densities and movement types, provided by the OverSim network simulation environment. The results show that SimMud performs well in the ideal simulations, supplying users with more correct neighbours and using less bandwidth at the cost of sending more messages and a longer message route. Vast performs better in the simulation representing recorded real-world movements at the cost of more bandwidth. The MOPAR simulations performed sub-optimally due to improper implementation, and FullConn can not scale.

Samevatting

Genetwerkde virtuele omgewings (GVO) is skynwerklike wêrelde waar gebruikers met voorwerpe in die omgewing, met mekaar kommunikeer, en interaksies ervaar deur boodskappe uit te ruil deur middel van 'n netwerk konneksie. 'n GVO gebruiker is dikwels slegs in staat om interaksies te voer met, of stel slegs belang in, 'n klein gedeelte van die wêreld in 'n sirkel rondom hulself; die area van belangstelling (AOB). Die funksie van 'n belangstellingsbestuur (BB) skema is om gebruikers bewus te maak van, en te laat kommunikeer met, ander gebruikers en items in die wêreld binne-in hul AOB, en terselfde tyd die informasie wat ontvang word te perk tot slegs relevante informasie. In hierdie tesis word drie bestaande eweknieë BB skemas; SimMud, Vast en MOPAR; tesame met 'n naïewe boodskap verruilingsimplementasie, FullConn, getoets. Hierdie skemas word gesimuleer in 'n skynwerklike speletjiewêreld wat verskillende bewegingspatrone en bevolkingsdigthede behels, danksy die OverSim netwerk simulasie omgewing. Die resultate toon dat SimMud die beste prestasie toon in die ideale simulasies deur meer gebruikers met die regte bure te koppel en minder bandwydte te gebruik ten koste van meer boodskappe stuur en 'n langer roete vir die boodskap om te volg. Vast vertoon beter resultate in die simulasie wat regte-wêreld bewegingsinformasie naboots ten koste van meer bandwydte gebruik. Die MOPAR simulasies het onderpresteer weens 'n onbetaamlike implementasie, en FullConn kan nie skaleer nie.

Acknowledgements

I would like to express my utmost thanks and appreciation to the following individuals and institutions:

- my supervisor, Dr Herman Engelbrecht, for his continued support and advice, even after a few slow years;
- MIH, for the financial aid towards my research and the opportunity to have worked in the Media Lab;
- my parents, Seppie and Settie, for their constant support and badgering, which ultimately helped me commit when my resolve had faded;
- my flatmates, Jacques and Rikus, for allowing me the peace and quiet to work in an otherwise over-stimulating environment;
- the myriad of friends and colleagues who have given their support to me in my part-time studies;
- and all the MIH Media Lab members; the most passionate, driven, insightful and entertaining group of people with whom I've ever had the pleasure of sharing a work environment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The availability and use of the Internet has seen a tremendous increase in recent years [48]. Soon the application of the Internet had grown to include entertainment as a service. One of these branches of entertainment took the form of interactive fantasy worlds populated by hundreds or thousands of users, participating and socializing in these world. The worlds are known as massive multi-player virtual environments. The architecture that is predominantly used to realise these environments is the client/server architecture. Over the past two decades, studies have been conducted to determine the viability of the use of the peer-to-peer network architecture [49] [43].

1.1 Massive Multi-user Virtual Environments

Massive multi-user virtual environments (MMVEs) are networked applications that serve as a virtual space where hundreds or thousands of users can connect, interact and socialise. The applications of MMVEs include critically important systems, such as air traffic control simulations or virtual war games [47], but their most prevalent iteration comes in the form of massively multi-player online games (MMOGs) [37] [36] [31].

MMOGs are video games in which hundreds or thousands of players play together in the same virtual world. These worlds range from simple text-based descriptions that are navigated and interacted with via typed commands, such as multi-user dungeons (MUDs), to vivid three-dimensional worlds that are steeped in lore and rich in opportunity, such as the popular World of Warcraft [9]. In such games, the world is usually based in some form of Tolkien-inspired fantasy world with elves, dwarves, orcs, trolls, etc.; or a science fiction world based around space exploration and alien encounters. Regardless of the setting, these types of games usually have shared traits:

- **Single-player interactions with the world** As an introduction to the game, the player is let loose into the world to fulfil some kind of quest on their own to learn the basics of gameplay, such as moving around, using abilities and interacting with non-player characters (NPCs). A lot of the in-game quests can be completed without the help of other players. This kind of interaction is referred to as player-versus-environment (PvE) interaction. These kinds of interactions are mostly done to increase the experience of the player's avatar, allowing them to grow and learn new skills and crafts through the game's levelling system.

- **Group-based interactions with the world** Players can choose to fight more powerful enemies by forming a group with other eligible players. The reward for completing such battles is usually in the form of better in-game gear, such as weapons and armour. These are referred to by players as "dungeons" or "raids", as their in-game incarnation usually entails entering into a cave of some description and destroying everything inside. These battles are instanced, meaning that when a group of players enters the dungeon, a separate instance of the dungeon is created on the server for those players. Each instance retains its current state (monsters slain, loot dropped, etc.) until a certain point in time when the instance is reset.

- **Single-player interactions with other players** MMOGs are usually designed with two or more opposing factions that the player can join. These factions can differ in many ways: ideological, cosmetic, race, etc., but their main function is to provide a sense of competition between players in the game world. As much as PvE interactions form a part of the game, player-versus-player (PvP) interactions are just as exciting and rewarding. While moving through a part of the game world, a player might spot another player of an opposing faction. The players can choose to engage in combat. There is usually not much at stake in one-on-one combat, other than a slight disruption to the flow of gameplay.

- **Group-based interactions with other players** A group of faction members can decide to join up and battle a group of opposing faction members in a large PvP "battleground". In some cases, the victorious faction stands to gain something from the encounter, such as a persistent bonus to the experience gained when adventuring in certain areas, or tokens that can be redeemed for gear. Groups of players can also form to storm an opposing faction's main base of operation. This is mostly done for the enjoyment of the experience in itself rather than for any form of story or character progression.

- **User-generated content** In some MMVEs such as Second Life [42], one of the main features of the game world is the ability to create objects that will persist in the game world for others to view or interact with. Second Life is closer to an MMVE than an MMOG, since there is little traditional game play associated with MMOGs, such as PvP combat or cooperative raids, but the same performance metrics apply.

MMOGs are some of the most engaging games on the market, with World of Warcraft players contributing a combined total playtime of roughly 6 million years in 2010 [46]. With the ubiquity of the internet and the growing availability of fast, cheap connections, it is understandable how the two modern media would converge into a form of entertainment that harnesses the connectedness of the world wide web with the social and competitive aspects of games to create a truly unique experience.

Developing MMOGs is a very expensive prospect. It is estimated that developing World of Warcraft cost $63 million over 4 and a half years [19]. Unlike designing a single player game, most of the costs of building an MMOG are generated by purchasing and maintaining the servers after launch. However, as World of Warcraft has shown through the growth of its user base towards its peak of over 12 million users in 2010 [61], it can be very rewarding. Many game development companies have since created their own MMOGs with varying degrees of success.

1.1.1 Client/server architecture

Regardless of their complexity or depth, all current implementations of these games are reliant on a client/server (C/S) network architecture. The C/S architecture entails that all players connect to a central server or server cluster to be able to play the game. The servers are purchased and managed by the game's creators. Figure 1.1 depicts a clustered client/server model used in the UnityPark suite of tools [50], used for making multiplayer games. Note that no interaction takes place between the clients directly. All communication goes through the central server. All client-side game state information is local copies of the authoritative global game state held by the server.

The advantages of using the client/server architecture to host an NVE are as follows [67]:

- **Authoritative game state** A single game state is hosted and managed on a central server or server cluster. All game logic is managed on the server. Game state conflicts are resolved by a single entity and the server is always trusted to have the most up-to-date version of the game state. The server holds the authoritative game state and has the final say on all object states.

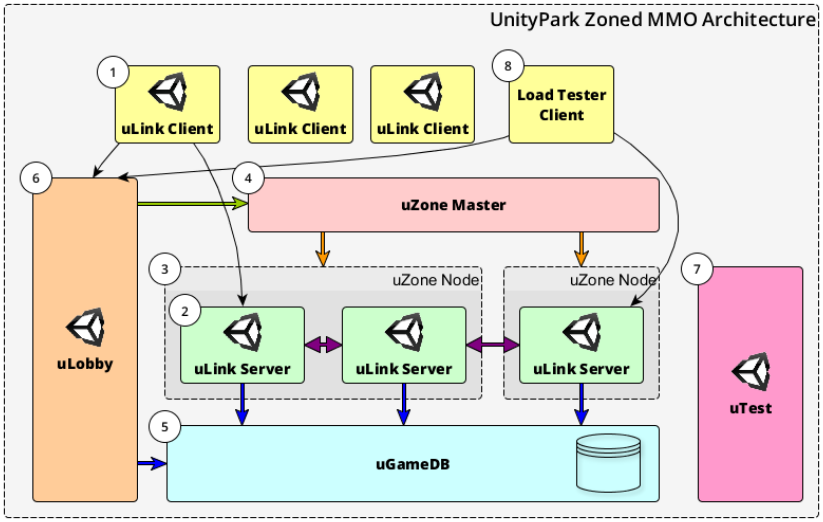

Figure 1.1: A diagram of the clustered client/server architecture used by the UnityPark suite [50].

- **Controlled access to sensitive information** Subscriber details such as personal and credit card information is stored centrally and access to it is closely monitored and controlled. Unauthorised updates to objects are detected by the server and disallowed, leaving the object in a legal, logical state.

- **State persistency managed centrally** The server is configured and maintained to be operational and accessible at all times, so that all game state updates are managed and stored centrally. Even if all players in the game world are disconnected, the game state remains in the server. Many redundancy systems and backup routines are also in place to ensure that the system suffers as little game state loss as possible in the event of a server crash.

- **Message routing is simplified** Since all game logic is performed on the server, the only address the game client has to send messages to is the server, and the server will take care of message routing and event dissemination.

- **Centralised hosting of objects and NPCs** Mutable game world objects, such as NPCs, resource nodes and chests, are hosted centrally on the server. All game logic is applied by the server, and that includes interactions with game world objects.

- **Global connectivity is maintained** Each player is connected to in the game world is connected to the server at all times. This means that the server has a global view of the state of each player avatar. A player's locality in the game world does not affect how a message is routed during, for example, a private

chat with another player. Whether they are right next to each other or they are at the most distant outreaches of the game world, they will always remain in contact with one another through the server.

It can be said that the choice of using the client/server architecture to manage MMVEs is based on the simplicity of implementation. The list above is not comprehensive, but it is a fair indication as to why using this architecture is effective, albeit not without its own inherent flaws.

The use of the client/server architecture does, however, comes with some drawbacks. These include weak scalability, weak robustness, expensive hardware setup, server overloading and a fractured game experience [55].

- **Weak scalability** A centralised server has an upper limit to its computational and communication performance. Adding more processing power to the server or server cluster adds network and computational overhead, ultimately causing a diminishing return in processing power for a considerable monetary cost [37] [32].
- **Weak robustness** A server is a central point of failure in the client/server architecture. If maintenance needs to be performed on the server, the hardware has to be upgraded or the server crashes, the environment becomes unavailable. This is an inherent part of the C/S architecture, and steps are taken to minimise down time, such as backing up game data often, and having more than one game server for the player to connect to. Server downtime has the ability to negatively impact the player experience if the servers are not maintained properly. This problem can be alleviated with the use of cloud servers, as is expanded on in section 1.1.2.
- **Expensive** In order to support more and more players, the server must be made more powerful and have ample bandwidth to communicate with the players. As stated, client/server architectures do not scale linearly, and hosting more people per game world becomes exponentially more expensive. By using a pricing calculator for RackSpace US, a leading cloud hosting platform [54], a hypothetical model for hosting 1000 players was created. This model consists of 4 120GB I/O-optimised servers to host multiple instances in-game (such as battlefields, dungeons, etc.), 2 1TB database servers to store player records and item information, a load balancer rated for 1000 concurrent connections, and an estimated bandwidth of 30,000 GB per month, averaging out to 1GB per day per player. This cloud solution would cost the game provider US$20,155.49 per month; a substantial financial investment for a game that hosts only a thousand concurrent players at its peak.

- **Server overloading** When an MMOG is first developed, the game provider cannot accurately predict how many players will be connecting to the game when it first becomes publicly available. This means that if a developer does not supply sufficient server power and bandwidth and the influx of new players is too high, the servers will be overcrowded and a portion of the player base will not be able to play the game. If a developer supplies too much server power, on the other hand, the funds to obtain and maintain a large server that is not running at optimal capacity is wasted. There is no way to determine beforehand how many players will be playing the game at any given point. The typical approach for the developers of an MMVE is over-provisioning. A similar problem associated with server size is the occurrence of flocking behaviour [16], where a magnitude of players flood to one area of the game world. If the server's processing capabilities for that area are not sufficient, it would hamper the experience of the players.

- **No truly global instance** Since MMOG player subscriptions can number in the millions [61], and the cost of hosting so many users concurrently on a single server would incur a monumental monetary cost, a different approach has to be taken to host so many users. With the design of the EverQuest MMOG, Sony implemented a system where multiple instances of the game world were hosted on different servers. Players would then connect to one of these worlds, and players would have little to no interaction with players on different servers [41]. This system is known as "sharding". While this is an effective way of hosting a large number of players, it does not offer a truly connected experience.

Table 1.1 sums up the differences between client/server and peer-to-peer MMOG architectures.

While the client/server architecture for MMOGs is well-implemented, secure and designed to minimise the impact of its drawbacks, it is not an optimal approach to an application with the intended scale of an MMOG. Optimally, an MMOG should theoretically be able to scale infinitely with the number of players in the game world, while still offering the same level of security, state consistency and low latency needed for satisfactory player experience. The static processing and bandwidth allowance exhibited by client/server architecture could prohibit the player base from growing if the concurrent connection capacity is reached quickly, and suitable hardware provisions have not been made.

Category	Client/server architecture	Peer-to-peer architecture
Cost	High initial cost and upkeep	Low initial cost and upkeep
Scalability	Difficult to scale; limited by server capabilities	Scales by number of concurrent players
Infrastructure needed	Additional server hardware needed to scale	Minimal infrastructure needed as gateway server
NPC and object hosting	Handled by server	Non-trivial object hosting (dependent on individual implementation)
Message routing	One message hop between client and server	Dependent on individual implementation
Security & access	Handled on server	Low security; each player is a potential message forwarder
State persistency	Handled trivially on server	Non-trivial; handled uniquely per implementation

Table 1.1: The differences between client/server and peer-to-peer MMOG architectures

1.1.2 The Role of the Cloud and Cloud Gaming

A possible solution to the problem of server hardware limitations is to host the MMOG on a cluster of cloud servers, on a service such as Amazon's Elastic Compute Cloud (EC2) service [2].

Cloud computing [1] is considered to be the future of providing information and services on both individual as well as organizational scales. It offers web-based services that are scalable, customizable, interchangeable and reusable. These services, and the hardware they run on, are maintained by the service provider. With the aim of keeping their clients' services running, these systems are designed to be reliable and always available. Security is also managed by the cloud service provider.

Given the listed benefits above, hosting an MMOG server on a cloud-based service is an attractive prospect. The benefits for the developers would be that the service is always available, and it could scale to the size that they need on demand. The only real drawback to this is that the game provider would be making use of the services of another company to host their game world. The control offered by a developer-owned server solution must be weighed against the flexibility and low initial investment provided by a cloud-based solution.

Another form of cloud gaming refers to the use of a cloud-based service that outsources the processing involved in rendering the game world and applying game logic to the cloud. This method makes use of a thin client that is not processor intensive. This client accepts user input, streams these inputs to a server or servers where it is processed, and the resulting action is then streamed back to the client. The client also receives pre-processed 3D renders and audio information, as these

are also outsourced to the cloud. This keeps the client application light-weight, and allows for the game to be played on devices that would otherwise not be able to process graphical information as effectively, or without the use of an additional graphics processing unit. A service like Sony's Gaikai [25] game streaming services would, potentially, allow any game to be played on any platform that supports multimedia streaming and has access to a sufficiently fast internet connection.

The 'thin client' cloud gaming solution has the disadvantage of requiring a persistent internet connection and significant bandwidth. If the media stream is interrupted in any way, the game ceases to function, since the information necessary to participate in the game does not reach the user. Similarly, if the information is not conveyed to and from the user at a fast enough rate, the player experiences lag and is therefore not able to effectively participate. Therefore, a well-established internet infrastructure with sufficient bandwidth is needed for a thin-client cloud-based system to be successfully implemented.

1.2 Peer-to-Peer MMOGs

To mitigate the design shortcomings of client/server architectures, research has been done into moving MMVEs from the client/server architecture to using the peer-to-peer (P2P) network architecture.

Peer-to-Peer (P2P) MMOGs has been a topic of research focus for a number of years. P2P MMOGs can be described as a game where the game's state and object storage, event logic processing and event dissemination is managed wholly or largely on the players' computer or console. Modern gaming systems have large processing capabilities. By utilizing some of those resources that would otherwise be left idle while playing, a P2P MMOG can use the available processing cycles and bandwidth to perform tasks that would otherwise be performed by the server in a classic C/S model.

Figure 1.2 depicts how an NVE would be hosted by a peer-to-peer approach. Each peer in the game world hosts its own avatar, as well as any assigned objects. The peers control the states of the objects and avatars and disseminate events to the relevant peers in the network, to ensure that their local copies of the objects and avatars are up to date. The important differences to note between the client/server and peer-to-peer models is that the latter not only allows, but relies on direct communication between nodes in the network; and that no entity in the peer-to-peer network has a complete, up-to-date global view of the game world.

The prospect of utilizing P2P architecture to host MMOGs in a distributed manner holds many advantages over the currently-dominant C/S model, as explained

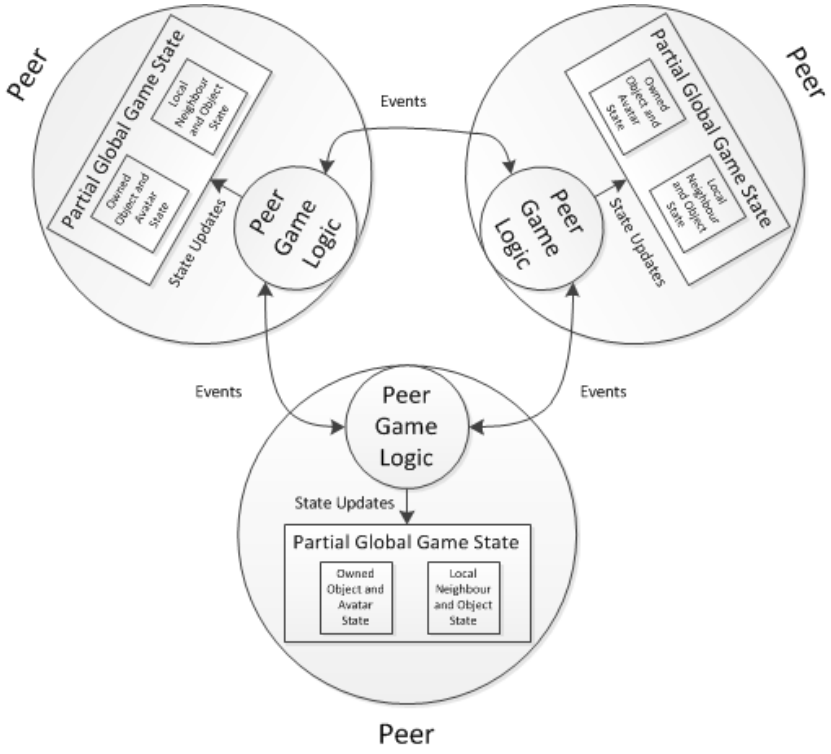

Figure 1.2: A basic representation of the peer-to-peer model as implemented by NVEs. [26]

Component	Minimum requirements	Recommended specifications
Processor	Intel® Core™ 2 Duo E6600 or AMD Phenom™ X3 8750	Intel® Core™ i5 2400, AMD FX™ 4100, or better
Graphics processor	NVIDIA® GeForce® 8800 GT, ATI Radeon™ HD 4850, or Intel® HD Graphics 3000	NVIDIA® GeForce® GTX 470, ATI Radeon™ HD 5870, or better
Memory	2 GB RAM (1 GB Windows® XP)	4 GB RAM
Storage	35 GB available hard drive space	35 GB available hard drive space
Internet connection	Broadband internet connection	Broadband internet connection

Table 1.2: The minimum and recommended computer specifications needed to play Blizzard's World of Warcraft [10]

below.

- **Scalability** In a P2P network, the total processing power and storage space is the sum of the users in the network. Any player joining the game will be bringing their own resources to the game. Given the processing ability of current-generation computers and consoles, as well as increasing bandwidth for residential use, players can easily provide the resources needed to host themselves in the game world. Table 1.2 shows the minimum required and recommended hardware specifications to play Blizzard's popular MMOG World of Warcraft [10]. The global average internet connection speed is 5.1 Mb/s as of the second quarter of 2015 [62]. While the specifications of each player's computer and internet connection speed vary widely, on average each player can be expected to contribute the recommended hardware specifications when joining the P2P network. The processing power and bandwidth afforded by the players' machines add to the overall processing of the game world. This means that the MMOG's processing power increases as players join the game world, at no monetary cost to the game's creators.

- **Cost** As stated above, the network is created by players joining the game. They provide all the bandwidth and processing power needed for processing in-game actions as they enter the game. No large server farms are needed to host the game world, and all communication and processing is handled in a distributed fashion, rather than on a server with a limited processing capacity. This allows developers that do not have a large amount of initial capital to develop MMVEs without investing in server infrastructure.

- **Truly connected world** Since there is theoretically no maximum number of players in a distributed network, and therefore no upper limit to the bandwidth and processing power available to the network, tens or even hundreds of thousands of players can concurrently connect to a single world.

If a more scalable, cost-effective infrastructure for the development of MMVEs can be produced, developers can be given more freedom to produce games that would otherwise be considered too risky to invest in. When investments in the order of hundreds of millions of dollars are considered, such as the $200 million development cost of Star Wars: The Old Republic MMOG [44], investors are reluctant to fund a risky MMOG. This leads to many investors only funding games that are similar in style to World of Warcraft to try and emulate its success and subscriber number [64]. A distributed, P2P infrastructure would nearly eliminate the cost of the servers that need to be purchased and maintained, which would decrease the initial investment

needed for the game, and allow the developers more freedom to produce a more unique game.

One important difference between P2P MMOGs and other P2P applications, such as file sharing programs like the Gnutella framework [56], is the level of dynamism in video games [59]. A file sharing network's layout and the data stored on it changes infrequently. An MMOG generates events and updates multiple times a second per player. Information such as player position, movement vector, inventory and health, as well as any actions performed, must be communicated to nearby players. Players can tolerate a certain amount of latency during game-play. This tolerance is dependant on the type of game [60] [4]. Games where a player assumes direct control of a character, such as a first-person shooter (FPS) game, requires a low latency, as game-play is heavily reliant in reflexes and quick actions. A game where the player is not in direct control, such as controlling an avatar in a role-playing game (RPG) or issuing orders in a strategy game, players are willing to tolerate higher latencies [17].

A way to reduce bandwidth consumption in MMOGs is to limit the amount of information transmitted and received by each player. This spawned the concept of interest management (IM) [49]. IM is the concept that if a player has a limited range to all actions they can perform, they cannot interact with anything beyond a certain range. Since the interaction with objects or players outside this range is restricted, the player does not need to know the state of anything outside this area. This manifests itself in games as the player's Area of Interest (AoI). The AoI extends further than the range of interaction to allow players to see objects of potential interest, such as a loot chest or an enemy player. This helps to minimise the amount of information a player needs at a given moment, since the only information they are given about the world pertains to their immediate surroundings.

It is important for an online game to have as low a latency as possible to allow players to analyse and react to the game world around them. This low latency should not come at the expense of any relevant information that the player may need. By limiting the size of the AoI and thus the range at which players can interact with one another, the latency may be kept lower, but less relevant information is given to the players, thereby affecting the game-play. The chosen architectures will need to be able to give players the maximum relevant information with which to make effective, informed decisions while keeping the bandwidth requirements to a minimum.

1.3 Peer-to-peer interest management schemes

P2P networks pose several challenges for MMOGs [21]. The distributed storage of information makes finding specific player information difficult, since there is no central repository like a server. During the preliminary literature study, different P2P MMOG architectures were investigated, such as VAST [31], SimMud [37], MOPAR [69], Donnybrook [8] as well as the commercially available Badumna architecture [39]. Each of these architectures have their own method of maintaining a consistent view of the world. These mechanisms can be classified into three groups, as listed below. Each of the chosen architectures are discussed in detail in chapter 3.

- **Structured P2P Architectures**
  Structured P2P architectures use existing data structures, such as Distributed Hash Tables (DHTs), to maintain a list of peers in the network and the information that pertains to them. SimMud [37] was chosen to represent the structured P2P architectures.

- **Unstructured P2P Architectures**
  As the name suggests, unstructured P2P architectures maintain player information without the use of structures like DHTs. Most often, they rely on a system known as mutual notification [38] that constantly disseminates information to other players in the network. VAST [31] was chosen to represent unstructured P2P architectures.

- **Hybrid P2P Architectures**
  Hybrid P2P structures make use of both structured systems and unstructured inter-player communication to maintain a consistent view of the world and distribute player information. These architectures are usually hierarchical in nature. MOPAR [69] was chosen to represent hybrid P2P architectures.

Using the OverSim P2P network simulation environment [6], each of the selected IM schemes is either reconstructed from the literature, or the existing example code in the OverSim suite is used to simulate a game-like environment. Each architecture is given the same simulation time, game world dimensions and number of players to simulate. During and after these simulations, data is collected from each player in the world, as well as from the network as a whole using a global observer module in the OverSim network. The metrics for determining the effectiveness of the architectures are described in Chapter 4.

1.4 Research objective

The objective of this work is to compare a selection of interest management schemes for peer-to-peer massively multi-user virtual environment. The aim of the study is to focus on the performance of each of the selected interest management schemes with regards to the latency requirements and bandwidth constraints imposed by the game application. The OverSim network simulation suite is chosen as the implementation platform.

1.5 Research contribution

The research contributions of this thesis are twofold. The first contribution is the design of the OverSim implementation of the MOPAR overlay [69]. This hybrid interest management scheme has no implementation in the OverSim simulation environment, and no empirical measurement of its performance in comparison to other interest management schemes. The second contribution is one of evaluation. This thesis sets forth a framework to use to determine the effectiveness of different interest management schemes for MMVEs. Each of the selected IM schemes is implemented in OverSim, and the data collected from the simulations are used to compare the performance of each scheme, including the newly-implemented MOPAR overlay, in terms of latency requirements and bandwidth consumption. Further research can be performed using the same metrics that are gathered and compared in this thesis.

1.6 Thesis overview

The thesis aims to illuminate the circumstances under which different methods of message routing are better suited, and where the performance of each approach fails to meet the requirements of an NVE.

The rest of the thesis is as follows. Chapter 2 is an overview and analysis of the available literature. Chapter 3 describes concepts that pertain to the structure of interest management schemes. It also describes, in detail, the chosen interest management schemes used in the simulations. Chapter 4 outlines the implementation of the simulation and experimental set-up, as well as the metrics used to analyse the effectiveness of each of the architectures. The results of the simulations are discussed in chapter 5. The thesis is concluded and future work is proposed in chapter 6.

Chapter 2

Literature

The field of peer-to-peer (P2P) networking schemes is a widely researched field, with applications in data storage and retrieval, communication and entertainment [57]. It offers increased scalability and promotes the fair use of resources in large computing systems and networks. P2P systems are more resilient to failure, since there is no single point of failure, and these systems are build to withstand the effects of churn, the constant joining and leaving of nodes. But with these benefits comes several intrinsic issues that must be overcome; issues such as security, balancing the use of resources and data persistency and consistency. This chapter presents information on a small number of interest management schemes that are used in client/server based architectures. It also highlights some of the problems raised with the implementation of a P2P overlay, its application in massively multi-player on-line games (MMOGs) and gives a brief overview of some of the schemes that have been proposed in literature.

2.1 Interest Management

The idea of interest management stems from the idea that in multi-player games, a player's avatar is limited in its movement and sensing capabilities. An avatar can only move a certain distance in a given amount of time, and it can only interact with objects or other players in its vicinity [43]. The result of this is that access to game data show both spatial and temporal locality. Stated otherwise, an avatar only needs to be informed of a subset of the global game state's information that is relevant to their current location at a certain point in time. Moving to a different place in the game world, or simply spending time in it, would result in another set of information becoming relevant or "of interest" to the avatar. A player in the game therefore only receives updates and events that are spatially relevant to players or objects that they are interested in, and the players manage the local copies of said

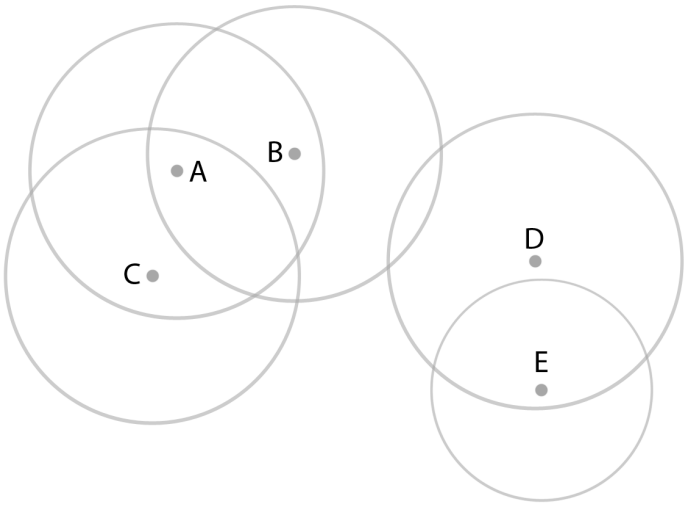

Figure 2.1: A visual representation of the aura-nimbus model.

players and objects, as well as their own game state. While spatial information, such as movement and close-proximity player interactions like combat, is broadcast to interested players based on in-game distance, non-spatial information, such as chat or trade messages, are routed through the network directly to their destination through different methods.

The spatial model, also known as the "aura-nimbus" model [11], of interest management uses the property of space as the basis for determining interest and interaction. The model makes two abstractions: the aura refers to the boundaries of the in-game object's presence. The nimbus refers to the bounds of an object's awareness. The nimbus is often referred to as the area of interest (AOI). In Figure 2.1, the dots represent the auras of avatars in a two-dimensional networked environment, and the rings around them represent the extent of their AOIs. Avatar A is mutually aware of both B and C, because both B and C's AOIs overlap with A's aura. The same can not be said for the inverse, though. C and B are both aware of A, but not of one another, even though their AOIs intersect. The same is true for B and D. D is aware of E because D's AOI overlaps with the aura of E. Because of E's reduced AOI, however, E is not aware of D, so the awareness is not mutual.

The most common approach to interest management is that of zoning, where the game world is divided into smaller partitions known as regions, cells, zones or sections. Zoning mechanisms differ in their partitioning of the game world, and each one approaches interest management differently. The simplest form of this mechanism would be to have the AOI cover the whole game world. This approach

is implemented in one of the chosen interest management schemes and is discussed in section 3.5.4. Other zoning approaches use square regions [37], hexagonal regions [69], Voronoi diagrams [29], triangular tiles [11] or other arbitrary polygons formed by convex hulls [24]. Each approach takes slightly different measures to maintain other aspects of MMVEs such as object hosting, state persistency and global connectivity.

2.2 Interest management in client/server MMVEs

The benefits of using a client/server network architecture to host an MMVE were already discussed in section 1.1.1. This section briefly discusses a study relating to interest management schemes and how they are used in a client/server environment. Please note that this section summarizes and makes extensive references to Boulanger's work [12]; any additional references will be cited as they are used.

Boulanger [11] conducted a study comparing different interest management schemes for client/server-based MMVEs. Using the Mammoth engine [45], a number of interest management schemes were implemented in a virtual world, complete with game objects and walls. The interest management schemes focus on how the world is divided, and how the server approaches the problem of optimally discovering which players and objects are of interest to a particular player.

The simplest implementation of these is the Euclidean distance algorithm, where subscribers are determined by calculating the distance from the center of one node to the center of the neighbouring nodes. If the neighbours fall within the AOI of a node, they are of interest to that node. This method performs optimal interest management and is easy to implement, but it has the disadvantage of having to compute the distance between each pair of subscribers and publishers in the space. As the game world's population increases, this approach does not scale well, since the continued calculation of the distance between each pair of nodes grows exponentially.

Another approach similar to the Euclidean distance algorithm is the ray visibility algorithm. This entails removing segments of the AOI that are hidden from view by non-transparent, immutable in-game objects such as walls and rocks. A line is drawn from the edge of an opaque segment, such as a wall, to the center of the node, creating lines in the AOI which represent the edge of visibility. In this algorithm, only neighbours that are in the AOI and that are visible are of interest to the node. This helps to eliminate the updates from avatars that are "indoors" when the player is in the "outdoor" game world. This, however, falls prey to the same drawbacks as the Euclidean distance algorithm, as well as having to compute the line of visibility relative to the game world for each movement update of the node.

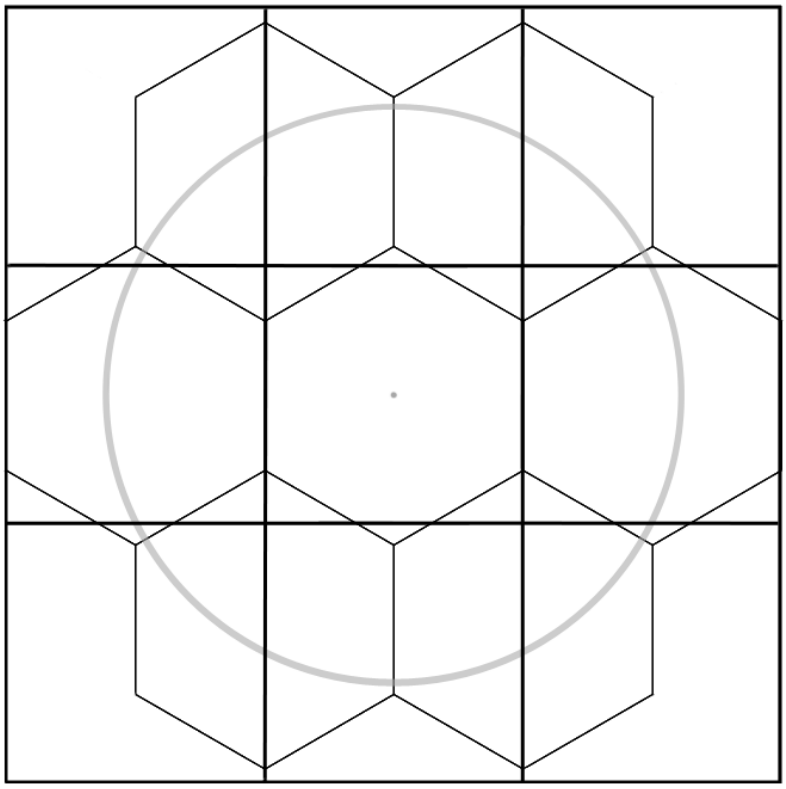

Figure 2.2: The approximation of a circle by use of squares and hexagons.

Region-based interest management divides the game world into sections. Each section could be any arbitrary shape. Dividing the game world into sections allows the interest management to be performed on a group of subscribers based on the section that they occupy. When a player performs an event, for example, all updates are broadcast to the region that it is occupying, and possibly neighbouring regions as well. The complexity of the AOI's computation remains constant, so a region-based approach scales well. The disadvantage is that region-based interest management adds additional information that falls outside the AOI of the node. For example, square regions are a bad approximation of the circular AOI. A grid of hexagons are a slightly better approximation of a circle than a grid of squares, but some information that is not of interest to a node will still be broadcast to it. Figure 2.2 shows that even if the square regions are as long as the width of the hexagonal regions (the shorter of the two dimensions), the squares still registers a larger subscription area than the use of hexagonal regions.

In the paper [11], four different interest management algorithms are based on the use of triangular tiles to partition the game world. The benefit of the triangular tiles is that they can subdivide the world while allowing arbitrary polygonal obstacles in the game world, such as walls, to be excluded from the partitioning of the game

world. This is helpful for interest management that is designed to exclude any nodes that are occluded by immutable game world obstacles. The game world is divided using Delaunay triangulation [7]. Obstacles are stationary and do not change in the paper's implementation. The tiles are stored in a graph. Each vertex represents a tile and neighbouring tiles are connected by an edge. Tiles are considered neighbours if they share a common point that is not on an obstacle.

From this graph of triangular tiles, four interest management algorithms are tested.

- The tile distance algorithm calculates the tiles of interest by finding all the tiles that are connected to the tile that the node is currently occupying within the node's AOI. Tiles that are occluded by obstacles, but that are still reachable through a path formed by the connected neighbouring tiles are included in the calculated AOI.

- The tile visibility algorithm calculated the tiles of interest by finding all the tiles inside current node's AOI that are "visible" from the tile that the node is occupying. A tile is considered visible from another tile if a line can be drawn from any point inside the tile that does not intersect an obstacle. The tile visibility value is static, so it is precomputed and stored to eliminate unnecessary computational overhead for the server.

- The tile neighbour algorithm determines the AOI of a node by traversing the tile graph breadth-first from the node's current tile to a given depth, collecting all the tiles along the way. This algorithm is easy to implement, but since the size of the tiles are not uniform, a preset depth value might not cover the full AOI of a given node.

- The tile path distance algorithm uses the distances between the centers of each tile to determine the AOI of a node. The path distance between two tiles is defined as the sum of the distances from the centers of each triangle along the path connecting the two tiles. The AOI of a node is determined by traversing all the neighbouring nodes until a given distance is reached. This algorithm uses static data, so like the visibility algorithm, these values can be precomputed. As with the neighbour algorithm, however, the non-uniform tile shapes might not cover the entire AOI with a preset distance value.

These interest management schemes were implemented in the Mammoth engine on a client/server based architecture with clients connecting to the server over TCP/IP. Both random movement and real-world traces were used in the evaluation of the effectiveness of the interest management schemes. The size of the triangular

tiles was also a variable, since larger tiles meant more AOI subscriptions, which results in more superfluous updates being sent to nodes. Message filtering is an important part of interest management. Especially in peer-to-peer IM schemes, measures have to be taken to ensure that nodes maintain proper state consistency while minimizing unnecessary transmission of redundant information.

Boulanger's results found that making use of the visibility factor does decrease the amount of messages generated by the server. While this is a pertinent observation, the implementation outlined in chapter 4 does not make use of in-game obstacles, so it is not of relevance to this thesis.

The paper also indicates that the use of hexagonal regions have a marked improvement over square regions. This has been proven by other studies as well [22]. One of the schemes that is compared in this thesis, discussed in chapter 3 makes use of a square grid, and another uses a hexagonal grid. These design choices need to be taken into consideration when looking at the results, since these inherent design choices will affect the outcome of the simulations.

2.3 Peer-to-peer interest management schemes

Interest management in P2P applications is more intrinsically tied with the rest of the components of the MMVE environment. In a C/S environment, interest management is mainly applied server-side, with little help or interaction from the clients. In a P2P environment, however, how interest management is applied to each node is determined by other factors, such as the network overlay and the neighbour discovery algorithm.

2.3.1 Issues facing peer-to-peer MMVEs

In P2P overlay schemes, there are many different design issues that need to be addressed. Amoretti's article, "A Survey of Peer-to-Peer Overlay Schemes: Effectiveness, Efficiency and Security" [3] highlights many of the challenging aspects that a P2P system must overcome. Chief among these issues are information security and integrity in a distributed environment, and the efficiency and effectiveness of the selected overlay. "Design issues for peer-to-peer massively multiplayer online games" [21] also highlights issues facing distributed networked environments, such as interest management, NPC hosting, game event dissemination and cheating mitigation.

A paper by Hu et al. entitled "VON: A Scalable Peer-to-Peer Network for Virtual Environments" [30] lists a group of characteristics of a networked virtual environment (NVE) that can be used as metrics for determining the performance

of IM schemes. These metrics are briefly listed below and will be elaborated on in Chapter 4.

- **Consistency** Each participant must perceive the same states and events. The consistency of the game world is the most crucial requirement of an MMOG; for the game state to be consistent and shared between all players. Consistency can become compromised when information regarding players or objects inside a node's AOI is unreachable (due to either network or hardware failure), high latency between the sender and receiver and unwarranted data manipulation, i.e. players cheating. The hosting of objects and NPCs falls into this category as well, as the state of in-game objects need to be consistent throughout the game as well.

- **Responsiveness** Updates and events must propagate and arrive at their destinations in a timely manner. This goes hand in hand with consistency. If a message has to be routed through too many channels, the player's state remains out of date compared to the shared game state.

- **Scalability** The number of nodes connecting to the network should not be a detriment to the overall performance of the network. On the contrary, the more nodes available in the network, the more resources the network should have for communication and computation. Ideally there should be some sort of load balancing mechanism in place to ensure that interest management schemes that use a hierarchical structure, such as SimMud [37] or MOPAR [69], do not overtax the super peers.

- **Reliability** The propagation of information should not be affected by the abrupt failure of any nodes in the network. Replication messages should be sent often to ensure that if any node leaves the network, the its responsibilities, if any, are resumed by another suitable candidate.

- **Security** In an MMOG, update and event messages, as well as personal information such as account details, must be kept secure and free from tampering or unwanted access. Since each node in the network has the potential to be a forwarder of a message, it would not be difficult for a player to intercept and change unprotected messages.

- **Persistency** Virtual objects and the latest player states must exist persistently, even when no players are present in the game world.

The latter two topics are outside the scope of this thesis. Security is a complex topic in its own right, as is object persistency. Without a central server, there is

no authoritative game state and no user that is "more trusted" than another. A reputation system might help to determine which nodes in the game world are more suited to resolve game logic conflicts. Fan et al. [21] makes a distinction between proactive and reactive cheating mitigation systems. The simulations discussed in chapter 4 do not submit any false or poisonous data into the network, nor attempt to flood the network with requests.

Object and NPC hosting is also problematic in a P2P MMOG. SimMud [37] allows for the hosting of objects and NPCs by assigning a region coordinator to host and manage all the objects in that region. This has the benefit of having the coordinator act as the authority when an update conflict or cheating behaviour emerges. The drawback to any kind of approach that applies object responsibility to a node based on region, is that the mechanism assigning coordinators or super peers are not optimised to pick the most suitable super peer based in their processing capabilities. The other strategy is distance based object hosting, as proposed in [27]. Here, the node that is closest to the object in terms of in-game distance is responsible for the object. While it is likely that the player closest to the object or NPC is most likely to interact with that object, it is not unlikely that other nearby players will also communicate with it. This could incur a high computational cost and bandwidth use on the part of the object host, which results in latency for the other players in the game. Calculating which node is closest to an object is also computationally expensive, and with constant player movement around the object, host switching happens often.

Object and NPC hosting is not implemented in the simulations, for the sake of simplicity. The simulations are written under the assertion that player movement is the most frequent and most abundant form of update message sent [29]. Since the movement of the player counts as an update to an in-game object, the state of the nodes are constantly changing, so state consistency can be monitored.

Another important aspect of P2P MMOG design is incentivising players to part with their resources and bandwidth for the duration of the game. Although participation in any P2P application or game is voluntary, storage space, processing cycles and bandwidth are needed to keep the game functioning. This creates a tension between personal performance versus the collective welfare of the game world. In the game, incentive mechanics are built in to encourage players to offer their fair share of resources to the game world, and to discourage excessive use of the global resource pool. Fan et al. [21] describes the use of accounting and reputation mechanics to incentivise players to share resources and consume carefully.

The main focus of this thesis is to study state consistency and bandwidth consumption based on the communication strategy employed by different interest man-

agement schemes. State consistency refers to the process of keeping the local copies of nodes or objects, housed on any node that is not directly responsible for the state of said node or object, as up to date as possible. This is especially important in an MMOG application, as the player experience can be severely affected by stale data. For example, in a real-time game that features combat, if a player's latest information regarding the location or hit points of an enemy player is not up-to-date (or as close as it can be, given the nature of the network), engaging in battle with the enemy would be difficult. Without sufficient information regarding the enemy player (position, abilities, hit points, etc.), the player has difficulty assessing the combat. If this is coupled with a high latency in update propagation over the network, often referred to as 'lag', something as frantic and dynamic as real-time combat would be a frustrating ordeal for the player with the most lag. For a P2P MMOG to be successful, the consistency of information and the effective and timely propagation of updates have to be addressed as the chief concerns.

By making use of an interest management scheme, opportunities for state inconsistencies can arise. Since an IM scheme effectively limits the view a single node has over the network, slow or faulty neighbour discovery, dynamic AOIs and segmenting game worlds can impact a node's view of the game world. Each IM scheme selected from the literature for comparison has their own methods of dealing with the potential scenarios stated below, and these methods are expanded upon in their respective subsections, starting in section 3.5.

- **Dynamically resizing areas of interest** If an interest management scheme has a mechanic that allows for a node's AOI to grow or shrink, it could introduce state consistency errors. Consider nodes D and E as depicted in Figure 2.1. If an IM scheme required that D be aware of E, but not vice versa, the IM scheme would prevent node E from connecting with node D, and therefore D would have an inconsistent view of the game world.

- **Undiscovered nodes** When a node first joins a P2P MMOG, it must inform other nodes in the topology of its presence. Doing so often requires that a node be registered with some higher-functioning node (such as a supernode, discussed below) and/or broadcast its new location to potentially interested nodes. The information initially given to the joining node could be either incomplete or contain superfluous neighbours, both resulting in a state inconsistency. Most IM schemes tend to have some mechanism in place to allow joining nodes to connect with their relevant neighbours and then inform these neighbours of its arrival.

- **Nodes separated by large distances** When nodes are no longer inside the

AOI of any of their neighbours, they cease to broadcast any information to neighbours and will no longer receive neighbour updates. The node becomes separated from the rest of the game world. All nodes must have a way to rediscover and reconnect with other nodes in the game world after being apart from the rest of the game world, or else the node will not receive the relevant updates and its world view will be inconsistent.

2.3.2 Proposed peer-to-peer interest management schemes

In Krause's paper, "A Case for Mutual Notification" [38], the author makes a distinction between 3 different kinds of P2P protocols.

- **Application layer multicast (ALM) protocol** This kind of protocol uses standard ALM techniques to disseminate events and updates throughout the game world. SimMud [37] is one of these techniques, but with regards to the categories of this thesis, it is regarded as a supernode based protocol, since the system relies on the use of selected nodes as supernodes to each serve a segment of the game world. The method with which the information is disseminated is not being tested, but rather the hierarchy of the peers and how it affects the availability of information. SimMud is described in more detail in a further chapter.

- **Supernode based protocol** The supernode based protocol works by splitting the game world into subsections and granting each of these subspaces a supernode. The supernode acts as a central point of communication for all nodes within the subspace for which it is responsible. This means that a supernode has to maintain an up-to-date list of all the nodes inside its subspace and is responsible for receiving updates and events from these nodes, as well as disseminating this information to other nodes in the subspace. Krause used an architecture built on a publish-subscribe model for his supernode based protocol example, and it is described in [68]. While this PubSubMMOG protocol works in much the same way as SimMud, it differs in that all supernodes only accept messages within the first half of a short timeslot. Just like with SimMud, players register to the supernode currently serving the space that the player occupies.

  Each supernode has a backup node which will take over the responsibilities if the supernode left the region it was responsible for, either through failure or by broadcasting a leave message and exiting the network voluntarily, also known as "graceful leaving". The supernode is responsible for keeping the backup node up to date with the latest positions of the nodes in the cell. When

the supernode leaves the cell it is in control of, the backup node, through a predominantly inactive process, notes that the supernode is no longer present, and then uses the existing information regarding the peers in its current region to serve as a new supernode. It informs all peers, of which it is aware, of the change, and the nodes served by the new supernode send it their latest positions, with which the supernode creates a more recent view of the game world. As stated, the supernode structure is much the same as the ALM based protocol in basic functionality. SimMud was chosen as the supernode based protocol because of its simpler structure and its lack of the timeslot mechanic, which allows the game data to be disseminated in a fashion more akin to the other chosen protocols.

- **Mutual notification based protocol** Vast [31], proposed by Yu et al., was chosen in Krause's paper as the mutual notification based protocol. In Vast, nodes are not place in any hierarchy and do not make use of any formalized structure, such as a multicast tree or a distributed hash table. Vast functions by having each node be aware of the existence and the position of a number of other nodes in its immediate vicinity and using these nodes to form a Voronoi diagram. Nodes are informed of the activity of incoming nodes through the neighbouring nodes, who act as lookouts for potential new connections. This is a dynamic diagram that is recalculated with every movement of neighbouring nodes. The diagram also serves as a method of determining which nodes are nearest to the current node, as well as which nodes are on the boundary of the current node's area of interest (AOI). Each node is required to keep at least enough boundary nodes to enclose the node inside its own Voronoi cell, even if the nodes are located outside the AOI. The boundary nodes serve to notify the central node of the presence of a new node entering the space, and each node sends updates of its movement to each node it is aware of in the game world, hence the concept of mutual notification. Each node builds its own internal Voronoi diagram based on its neighbours' locations. By combining the individual diagrams, a Voronoi diagram of the whole game world can be constructed. Therefore, the system is said to have a complete, distributed view of the game world, if each node's personal Voronoi diagram is taken into account.

  Vast is the closest to a fully distributed P2P interest management system, since it lacks any formal structure and all nodes have equal responsibility.

The research was conducted in simulation using the OverSim [6] simulation suite. Nodes are placed in a two-dimensional space and are allowed to roam free or in

groups, depending on the simulation settings. They join and leave the network by means of OverSim's churn generator, which determines their life span in the game world. Message delays between nodes are based on the nodes' euclidean distance and their access net delay. The sessions were held over an average of 100 simulated minutes and contained an average of 500 live nodes. Player densities were changed by varying the size of the available game world. The most important metrics in the paper were overall message delay and bandwidth consumption.

- **Message propagation delay** Mutual notification proved to be the system with the lowest message delay, irrespective of player density. SimMud suffers greatly in densely populated game worlds. PubSubMMOG shows barely noticeable increases in message delay when player density increases. The size of the node groups as they moved through the world together made little difference in the overall message delay.

- **Bandwidth** In sparsely-populated game worlds, PubSubMMOG consumes less bandwidth than either Vast or SimMud. Although the bandwidth use increased with the size of node movement groups, PubSubMMOG remained less bandwidth-intensive than either of the other protocols. As player density increases, PubSubMMOG's bandwidth use increases as well, in combination with the increase in bandwidth use caused by growing group sizes. SimMud remains mostly unaffected by node group sizes, but shows an increase in bandwidth consumption as player density increases. Vast is the most bandwidth intensive protocol in low density scenarios, but its bandwidth use lowers as node density increases. Group size only affects Vast in high node density scenarios. It is to be noted that, using the given parameters, no protocol exceeded a 10 kByte/s transfer rate.

Based on these findings, mutual notification appeared as the most efficient approach to the propagation of states in a distributed MMOG.

Worth mentioning is an interesting P2P interest management scheme that can be used in first person shooter (FPS) games known as Donnybrook [8]. Relying on the characteristics of FPS games to have a limited field of view (usually between 90 and 135 degrees wide), and the tendency for players to be more focussed on the center of the screen than its periphery, Donnybrook scales the frequency of updates from players or targets. Targets that are more in view and closer to the center of the screen exchange more update messages than those at the edge of the player's view port. Targets outside the player's view port are updated even less frequently. The in-game distance from the player to the target is also taken into account, as closer targets are considered to be of more interest to the player than far away

targets, even if both of them are near the center of the player's view port. While this approach is well suited for FPS MMOGs, it would not be suited to a more traditional role playing MMOG, where the player's field of view is not restricted by camera placement, and often the player has a top-down view.

This thesis aims to include a hybrid scheme, MOPAR [69], into a comparative analysis to determine the characteristics of the hybrid system according to the same metrics as the structured (SimMud) and unstructured (Vast) IM schemes. The MOPAR scheme, as well as the other chosen IM schems, will be elaborated on in chapter 3.

2.4 Summary

In this chapter, the concept of interest management and the motivation for its use is explained. Some examples of interest management schemes used in traditional client/server were outlined and compared, and the main categories for peer-to-peer IM schemes were described. The challenges presented by the design of a distributed MMOG are also explained. In the next chapter, we will elaborate on the peer-to-peer interest management schemes selected from the literature, and how they mean to address each of these challenges.

Chapter 3

Interest Management Schemes

Interest management schemes refer to the mechanisms used to keep nodes in a network updated regarding the status of their peers without utilizing excessive bandwidth. This is of particular importance to MMOGs and other NVEs, where responsiveness is the top priority. In such a system, knowing the status and movements of in-game enemies or receiving important information at the right time is critical to the connected experience. By ensuring that only the most pertinent information is delivered in near real-time, while other, less relevant messages are gathered more passively and with longer delays, game-play becomes more immersive and the overall play experience is increased. This chapter focusses on explaining interest management in greater detail, how it is applied on both server-client and peer-to-peer systems, as well as a detailed description of the candidate IM schemes which will be simulated and compared in this thesis.

In Morse's paper, "Interest Management in Large-Scale Distributed Simulations" [49], a simulation is set up and run that splits the participants into a series of local networks that connect through a wider network, such as the internet, to communicate with one another. The simulation is of a military nature, describing different type of units in the field, each with unique sensing capabilities (infantry only have the ability to see 400 meters ahead, while a tank can see ground-based objects 4 kilometers away). The outermost range of these sensing capabilities are the furthest that the unit can assert an influence, or that the unit needs to be able to perceive activity so as to take an appropriate course of action. In these simulations, each participant broadcasts a so-called 'heartbeat' update periodically to assist newly joined nodes in connecting to the network, as well as to accommodate for lost messages due to network communication errors. These updates generated over 60% of the network's overall traffic, and much of the information is of no use to the receiving entity; sometimes up to 90% of the data transmitted is superfluous. Because a unit can only (and, for effective game-play, must be able to) perceive

other units or events at a finite distance, and can only assert any kind of effect over the same distance or less, there is no need for the abundance of information coming from sources beyond this range. We refer to this range as the Area of Interest (AOI) of a unit.

3.1 Latency requirements

In order for an entity in the network to interact with other entities, it is reasonable to assume that they must communicate with each other. The information communicated among one another is dependant on the type of interaction these entities can have, such as position, velocity and object-specific attributes (for example, the health of another player, or the contents of a container). The most frequent update that is sent in an MMOG is position updates. These updates must be sent at a rate that would allow observing players (the node receiving the updates) to discern the near real-time movements of a player, but at the same time the updates must not be so frequent as to incur excessive bandwidth usage. It has been shown that the tolerance for update latency differs between game types. Games where players take direct control of the in-game avatar, such as first person shooter (FPS) games or racing games, require packets to be delivered as quickly as possible. A latency higher than 180 milliseconds is deemed unacceptable and impacts the game considerably [4]. On the other hand, games where indirect control is assumed, such as commanding units in real-time strategy (RTS) games, can tolerate higher network latency [60], since the focus of the game is not on instantaneous reactions, but rather on strategy and tactics.

3.2 Interest management in client/server MMOGs

In client/server based MMOGs, the currently dominant network architecture for the genre, interest management is applied to each player to minimize the amount of information the server has to communicate to each player. Despite the server's large bandwidth (from 20 MB/s and upwards, depending on the specific application), handling requests from hundreds or thousands of players multiple times a second and sending out response messages to each player can tax the system. If the server does become overloaded, players experience 'lag' in the game caused by a delay in the server response packets being delivered to the player. This negatively impacts the player's experience of the game. Imagine a naive interpretation of an MMOG, where each player is communicated the whereabouts of each other player in the game world. This approach is completely unscalable, since the amount of player updates the server has to communicate for every full set of player requests is $O(n^2)$,

where ɪn is the amount of nodes connected to the server. This does not include any in-game objects that the in-game avatars can interact with, which would increase the amount of messages sent by the server even more. Since the players controlling the avatars can only interact with objects or other players within a certain range of themselves, all the information regarding players and objects that lies significantly outside this range is considered superfluous.

With the inclusion of a server in current MMOG architecture, which acts as a central repository for all object and player information, player discovery is a trivial issue. When a client logs on, its information is loaded from the server and sent to the client application, which is then populated with more information which the server determines is of interest to the recently logged in client. Avatars are discovered in the network merely by sending a connection request to the server with the avatar's in-game location. All other messages to the server contains location information, so the server is always aware of the avatar's most recent location. The main consideration regarding client/server architecture for MMOGs is the computational complexity of the algorithm used to determine which objects or avatars are of interest to a particular player.

3.3 Spatial publish-subscribe

An important concept to keep in mind when an interest management scheme is created is that of spatial publish-subscribe [34]. Each of the chosen IM schemes use spatial publish-subscribe SPS to ensure that they are connected to the proper nodes to receive updates that are relevant to them. All NVEs perform certain fundamental interactions and activities, and according to Yu, these operations can be reduced to spatial publish-subscribe operations.

In a distributed system, nodes are connected to one another in order to receive the information that is generated by the world around them. The receiving nodes are known as subscribers. To let message generators (publishers) know what information is the most pertinent to a given node, the node broadcasts an interest expression message, often in the form of a subscription request. When a node is subscribed to a particular publisher, all events that are published from said publisher are then sent to the subscribed node. In client/server architectures, the server takes the role of a wide-scale message hub, receiving all publications from connected nodes and then relaying them to subscribers who have expressed interest in the content.

The publish-subscribe mechanic can be adapted to suit the needs of an NVE by having the subscription criteria be determined by location in the virtual world, hence the term spatial publish subscribe. In an NVE, users take on a virtual avatar

who is given a position in the virtual world. The avatar expresses an area of interest (AOI), which is represented virtually by the avatar's visibility range. An avatar needs to be aware of its surroundings so the user can make use of this information and react accordingly. By dividing the world into smaller regions and assigning a publisher to each region, a node can simply subscribe to any regions that overlap with its AOI. However, the size of the region must be taken into account when designing the NVE. If the regions are too big, avatars will receive more information than they require as described by their AOI. If the regions are too small, they run the risk of increasing the maintenance needed for each subscription, such as frequently subscribing or unsubscribing to cells when moving.

Using spatial publish-subscribe as a primitive , each of the chosen interest management schemes is able to ensure that updates and events are distributed to the interested participants. On the most basic level, nodes in the network express their interest in events within a certain area, and the organizer of the sections within that area publish messages to those nodes.

3.4 Pastry

The following section gives a brief description of Pastry. It is presented in its own section because two of the chosen interest management schemes are built on top of this overlay, and it warrants further description.

Pastry [57] is described as scalable, decentralized object location and message routing substrate for application in peer-to-peer environments. Pastry is an overlay network that, for our simulations, helps to route messages between nodes in an application.

Each node in the Pastry overlay is assigned a 128-bit node identifier (nodeId). This nodeId is assigned as soon as a node joins the overlay, and it represents the node's position within the 128-bit circular name space of the Pastry overlay. The nodeId is usually generated by performing a collision resistant hashing operation, such as SHA-1, on the node's IP address or public key.

Each Pastry node holds 3 sets of information; a leaf set, a routing table and a neighbourhood set. The leaf set is a list of IP addresses of nodes with L nodes whose nodeIds are numerically closest to itself. $L/2$ of these nodeIds are numerically smaller than the given node, and the other $L/2$ is numerically larger. The neighbourhood set is not used for routing, but it is used to maintain the Pastry overlay structure. The values of the sets are variable, based on 2 configuration parameters, L and b. The parameter b is a configuration variable that defines the number base of the keys in the Pastry key space. L has a typical value of 16 or 32, and b is typically

NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Figure 3.1: The state of a Pastry node with the nodeId 10233102, where $L = 8$ and $b = 2$. [57]

the value 4. The routing table contains a list of IP addresses of nodes whose nodeId shares a common prefix with the given node. Figure 3.1 shows the state of a Pastry node with the nodeId 10233102. All the numbers are in base 4, $L = 8$ and $b = 2$. Note that as the row number increases, the entries in row n of the routing table contain n digits in the nodeId prefix that are the same as the nodeId of the depicted node. This is central to Pastry's routing mechanism.

For a Pastry node A to route a message with key D, A checks its leave nodes first. If the key D falls within range of the nodeIds covered by its leaf nodes, the message is routed to the appropriate leaf node in one hop. If not, the routing table is used to find a node that shares a prefix with D that is at least one digit more than the nodeId of A. If this kind of node is not found, the message is forwarded to a node that shares a prefix with D at least as long as the nodeId of A, but which is numerically closer to D than A's nodeId. It can be shown that the message will be routed to its destination in $\lceil \log_{2^b} N \rceil$ steps, barring significant simultaneous node failure. Eventual message delivery is guaranteed unless $\lfloor |L|/2 \rfloor$ nodes with consecutive nodeIds fail simultaneously.

Two of the selected architectures, SimMud and MOPAR, are build on top of Pastry, and MOPAR makes extensive use of the information maintained by the Pastry node to determine a node's suitability to become a home node. This mechanism is

discussed in detail in section 3.5.3.

3.5 Peer-to-peer interest management categories

This section will provide a more detailed description of the interest management schemes as described in section 1.3. Each subsection describes the interest management scheme chosen to represent each category of architecture.

3.5.1 SimMud - Structured P2P architecture

SimMud [37] is the IM scheme chosen to represent the structured approach to interest management. It makes extensive use of application-layer multicast and DHTs to route messages and to keep the game world consistent and connected.

3.5.1.1 Pastry and Scribe in SimMud

SimMud is built on top of Pastry [57], which was discussed in section 3.4, and Scribe [58], an event notification architecture for topic-based publish-subscribe applications. Scribe is, in turn, built on top of Pastry. Scribe builds a multicast group for each topic in the publish-subscribe model. Each group has an ID and it shares the same circular 128-bit name space as the Pastry overlay.

Scribe leverages the existing Pastry overlay to route messages through the network. For each publish-subscribe group that is required, a topic is created with its own topicId. The topicId is generated by performing a hash function on the topic's textual name hyphenated with the topic creator's name. A coordinator node is the live node in the Pastry overlay with the nodeId closest to the topicId of a given topic. This node acts as a rendezvous point for publishers and subscribers, also serves as the base of the multicast tree - a routing structure detailing the route messages will follow to reach each subscribed node - that describes the topic's subscribers. In the case of SimMud, it is assumed that the topic of each multicast group refers to the cells in the game world. Each cell is a separate topic, run by a coordinator node, whose nodeId is numerically closest to the topicId.

3.5.1.2 SimMud join and move operations

When a node joins the SimMud network, it is given a nodeId by the Pastry overlay. SimMud's game world is statically divided by design, so the position of the node's associated avatar is used to determine which cell the node resides in. A subscribe message is then routed through the Pastry overlay to the live node with the nearest nodeId to the message key. This key is a hashed function of the coordinates of the

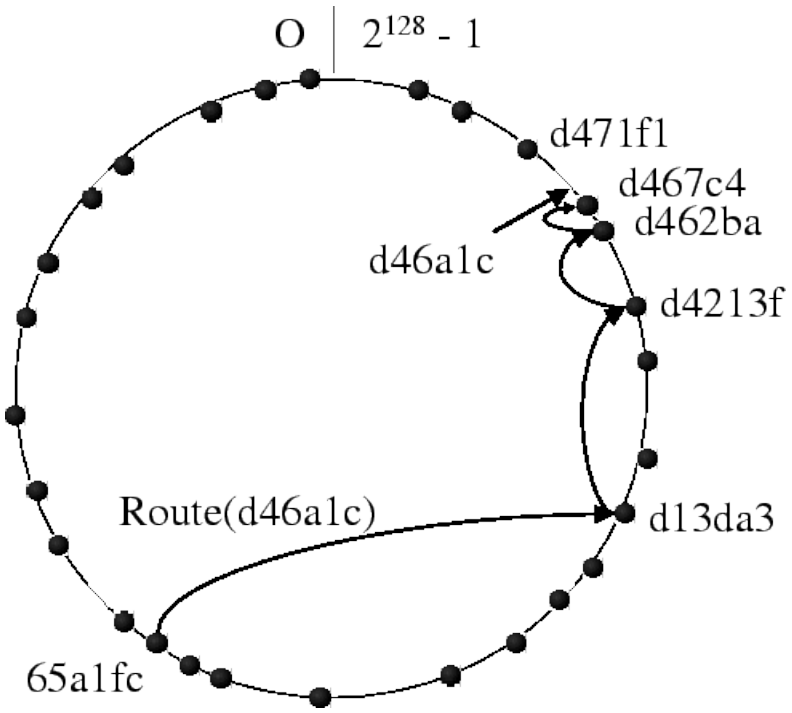

Figure 3.2: Routing of a message with key $d46a1c$ from source node $65a1fc$ through the Pastry. Each dot represents a live node in the circular namespace. [15]

cell that the joining node is residing in. Scribe refers to this key as the topicId. The coordinator is the node on the Pastry overlay whose nodeId is closest to the topicId. Figure 3.2 illustrates how a message is routed from the source, the node with nodeId $65a1fc$, to its destination, nodeId $d467c4$, which is the closest live node to the key $d46a1c$.

A topic's multicast tree is formed by using the union of the route from the topic subscribers to the rendezvous point. Each node along the route from the subscriber to the coordinator forms part of this tree and is called a forwarder for that topic. Forwarders maintain a children table for each topic, which contains the IP address and nodeId of each child in the multicast tree. When a node subscribes to an area, each node along the routing path checks its list of topics to see if it is already a forwarder for the given topic. If it is a forwarder for the given topic, it accepts the node as a child. If not, it creates an entry in its children table for the relevant topic and adds the source node as a child. It then forwards a subscribe message along the original route of the subscribing node towards the rendezvous point.

To minimize the amount of network traffic, SimMud subscribers only inform the coordinator of their movement or actions when they perform something that is not on par with what dead reckoning [53] predicts. The nodes communicate two pieces of information to the coordinator; the initial value of a variable V_1 and the updated

value V_2. If the value of V_1 in the subscriber's message is the same as that of the state that the coordinator manages, the value is changed to V_2 and the change is communicated to all subscribers.

When a node leaves a cell, it unsubscribes locally by defining the topic as no longer being required. If the node is not a message forwarder in the multicast tree, it sends an unsubscribe message up the tree, asking each node to remove it from the path, until it reaches a node that is still a forwarder after removing the leaving node.

3.5.1.3 SimMud design benefits and limitations

Using a highly-structured architecture ensures that message routing will be ensured under moderate failure conditions. Peers in the SimMud network also contribute processing power to the rest of the network. Coordinators manage the shared game state and consistencies of the nodes that they are responsible for. Mapping the coordinators to random nodes on the Pastry overlay instead of by their in-game location produces another benefit; a coordinator only has to accept or hand over responsibility for a cell when it enters or leaves the overlay, respectively.

The structured architecture does suffer from a larger number of network hops when routing a message. This results in a more delayed message propagation, which could raise the overall latency of game state updates and can be detrimental to the user's experience.

The use of predictive algorithms such as dead reckoning can incur additional computational overhead to the nodes in the network. The use of prediction thus becomes a trade-off between more frequent updates with little computational overhead, or less frequent updated to the coordinator with more computational overhead. The coordinators suffer from the effects of increased computation more than subscribers, since multiple nodes' paths must be predicted at once.

Only transient information is stored in the distributed network. Persistent information, such as user account details, is handled by a central server. The benefit of the decentralized system is that bandwidth and process intensive tasks are delegated to the nodes entering the network, instead of a centralized server.

Fault tolerance is an important aspect of any P2P application, since nodes leaving the network, gracefully or otherwise, could play a more active role in the overlay than they would in a client/server architecture. SimMud makes use of replication to ensure that any coordinators that leave are replaced quickly and with little loss of information. Large-scale network outages could cause partitions in the overlay and cause differences in state consistency. SimMud counters this by having a server (the same one that stores persistent data) that "blesses" one of the partitions as having

the authoritative game state.

3.5.2 Vast - Unstructured P2P architecture

Vast [31] is the IM scheme chosen to represent the unstructured P2P architecture. This scheme does not rely on any DHT structure to keep the nodes in the network connected, instead making use of direct message relaying and forwarding.

Vast, like the other chosen schemes, relies on a publish-subscribe mechanism to ensure that network nodes distribute events and updates to interested participants. In contrast to the structured approaches used by SimMud and MOPAR, a Vast node simply sends event messages to all other nodes in its AOI. This is called mutual notification [38]. Using mutual notification minimizes delays in event propagation, resulting from the one-hop message routing path from node to node.

3.5.2.1 Voronoi overlay network

Unlike the other schemes, Vast does not make use of fixed, predefined partitions for the game world. Vast uses a geometric algorithm to allow each node in the network to generate a Voronoi diagram [23]. The diagram is generated by using the node's positional information to dynamically divide the game world into irregularly-shaped cells. The edges of these cells are created by bisecting the line connecting two nodes, as represented by their coordinates in the game world. A single node resides in the center of the cell, and the cell is surrounded by others. The set of cells that each share an edge with cell A are known as the enclosing neighbours of A. The set of cells whose surface area extends beyond the limits of the AOI of cell A's node, is referred to as the boundary nodes of cell A. Note that one cell can be both a boundary node and an enclosing node. Each node builds up its own Voronoi diagram of its known neighbours, and together, they form a network of Voronoi cells. This creates a dynamic partitioning of the game world. Although each node is only aware of its own neighbours and the Voronoi diagram constructed from them, the combined diagrams form an accurate Voronoi diagram of the whole game world. This system is described as a Voronoi overlay network, or VON [30]. Vast relies on this VON for all of its event dissemination and neighbour discovery functionality.

Figure 3.3 offers a visual representation of the different types of nodes in the Voronoi overlay network. The red triangle represents a local node that we will use as the focus of the diagram. The red ring around the node represents the extent of the node's AOI. All nearby (in terms of the game world's geography) nodes whose cells overlap with the AOI of the local node are considered as neighbours. The yellow squares represent boundary nodes. These nodes' areas intersect with the AOI of the local node. The blue circles represent the enclosing nodes, as they each share an

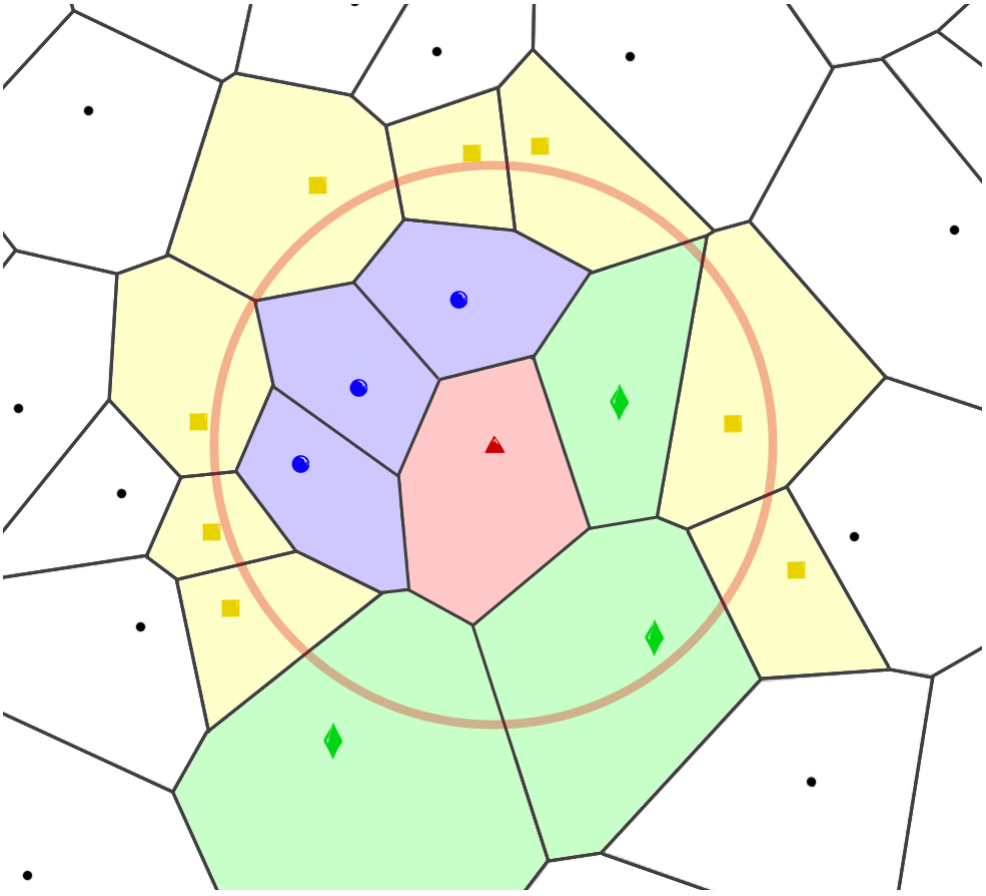

Figure 3.3: A section of a Voronoi diagram as used in the Voronoi overlay.

edge with the local node. The green diamonds represent nodes that are both an enclosing and boundary neighbour, since they meet both criteria. There can also be other nodes in the overlay that do not meet either criteria, but since they are still contained within the AOI of the local node, they still count as neighbours. This is not depicted in Figure 3.3. All other nodes, represented by black dots, are not visible to the local node. They form part of the global Voronoi diagram, but their activity is not of interest to the local node.

3.5.2.2 Vast join and move operations

When a node joins the network, it accesses a bootstrap server to learn the IP of an existing node in the network. The position of this node, known as a gateway node, is very likely not the nearest to the joining node's starting position. This means that the node which the joining node first communicates with will likely not be its intended neighbour. For the joining node to attain the corect neighbour nodes, the gateway node forwards the join message to the node in its neighbour list that has a

closer coordinate to the joining node's position than itself. This is known as greedy forwarding or compass routing. Once the forwarded join request reaches the node whose Voronoi region covers the point of entry for the joining node, this acceptor node returns a list of neighbours to the joining node. This initial list of neighbours allows the joining node to notify the joining nodes of other neighbouring nodes in its AOI that might have been absent from the initial list. Global connectivity is maintained by ensuring that a node is always connected with, at the very least, its nearest enclosing neighbours, even if the neighbouring node itself is located outside its AOI.

Moving is where the idea of mutual notification is most evident. When a node moves, it broadcasts its movement update to all neighbours in its AOI, or all nodes that it is aware of on its local Voronoi diagram. Each of the moving node's boundary nodes then performs a check to see if there is any overlap between its own enclosing neighbours' Voronoi region and the AOI of the moving node. The node receiving the movement update only replies to the update if a new overlap is found, and it replies with the information of the newly-discovered node. The moving node also runs a check on its own neighbour set, using its internal Voronoi diagram to determine if its AOI no longer overlaps with the Voronoi region of a boundary node. If it finds a node that it would no longer subscribe to, it sends a disconnect message to that boundary node.

When a node leaves, it notifies its enclosing neighbours of its intention, and the enclosing neighbours that are affected by the disconnection update their own Voronoi diagrams. If the leaving node is identified as a boundary node, a new boundary node may be assigned. If a boundary node leaves abnormally, it is detected passively by the affected nodes. A request is then sent out to known neighbours to share their enclosing neighbour information, so that the neighbour set can be updated and the topology retains its consistency.

3.5.2.3 Vast design benefits and limitations

The design of the Vast system ensures that network latency is minimised due to the one-hop nature of the network. With no relays in the system for message routing, this approach allows for responsive applications, an important factor for NVEs. When users are near each other in the game world, only the messages broadcast inside the user's AOI is received. This strict adherence to the broadcast of only the most relevant information ensures that ideal interest management is applied to each node in the system.

The Vast approach does have certain limitations and disadvantages. Since all messages are sent from one node directly to another, there is no room for message

aggregation or compression, as would be possible in systems where servers or 'super-peers' are present. This, coupled with the fact that messages have to be duplicated when sent out to all neighbours, incurs additional bandwidth costs, more than those associated with client/server architectures. Another design problem arises when nodes are separated by large distances in the game world. The topology of the game world can only be maintained and remain consistent when all nodes are aware of, at the very least, their enclosing neighbours. This would force an isolated node on the outskirts of the game world to reach outside its AOI to ensure that it maintains its enclosing neighbour set, reducing the granularity of the interest management. On average, though, these neighbours are few, given the characteristics of the Voronoi diagram.

In the literature there is no specific mention of Vast's resistance to large-scale network outage. Single nodes leaving the network are detected passively by their neighbours, due to the lack of update messages being broadcast from it. Given this trait, it is speculated that if a single node loses all of its neighbours simultaneously, the node would be isolated from the rest of the network. In this case, there is no neighbour to notify it of any incoming nodes or any other existing nodes in the area. This would be rare, since the position of the avatars in the game world are not connected to their geographical location.

3.5.3 MOPAR - Hybrid architecture

Mopar [69] is the hybrid interest management scheme chosen for comparison. It is referred to as a hybrid model because it makes use of both structured and unstructured P2P communication models. The structured communication is implemented through a Pastry [57] overlay and routing network for DHTs, as described in section 3.4. When a node first joins the Pastry overlay, it is assigned a key consisting of a hashed value of its IP address. Pastry routes messages via the DHT to the node in the network with the overlay key value that is closest to the key value of the message. This overlay network forms the backbone of the system, ensuring the interconnectedness of the nodes in the system on a broad scale. This also prevents nodes in the network losing connectivity with the overlay as a whole and cause the creation of smaller, unconnected networks within the game [24]. The Pastry overlay is used primarily by MOPAR's home nodes, which will be discussed shortly.

The unstructured communication takes place between slave and master nodes, which will be discussed in the section below. The unstructured approach has nodes communicating with each other asynchronously, sending primary movement updates between one another. This method allows for more fine-grained interest management and connectivity.

The MOPAR game world is divided into a number of hexagons of predetermined size, each with a corresponding 2-dimensional coordinate. The designers of MOPAR chose hexagons as the preferred tile shape because it approximates the circular shape of the AOI better than square tiles. The tile sizes are constant throughout a single iteration of the game world and for its duration, but they can be changed by setting a parameter prior to the launch of the simulation.

3.5.3.1 MOPAR node types

Three types of nodes exist in the MOPAR scheme, each fulfilling a specific role in the interest management scheme. The nodes form a hierarchy of responsibility, with the slave nodes having the least amount of responsibility in the scheme. The different node types are:

- **Home nodes** Each of the hexagonal tiles or 'cells' (the two terms are used interchangeably) in the MOPAR game world is assigned a key value, which is a hash of its x-y coordinate value, for example, a SHA-1 hash of the text value "0:1". The home node of a hexagonal cell is described as the node in the network whose Pastry overlay key is closest to the hashed key value of the cell. This node is said to be the home node for the hexagonal cell [x:y]. Home nodes are responsible for the routing of join requests for new nodes moving into the cell they are responsible for. One node can be the home node of multiple cells. This is often the case when the game world has only a few players in it. One cell, however, can only have one master node. A node can be the home node for a cell even if it does not occupy that cell. Home nodes are self-repairing and self-organizing. They form the primary lookup structure for the MOPAR scheme.

- **Master nodes** Master nodes are primarily responsible for message delivery between nodes in the network. Master nodes are highly connected to neighbouring master nodes and communicate with them directly. A hexagonal cell has only one master node, and any node on the network can only be the master node of one cell at a time. Master nodes are registered on the home node of the cell they are responsible for. This ensures that the home nodes can route any join requests from newly-arrived nodes to the correct master node.

  Master nodes act as a directory for nodes in the cell for which they are responsible. The master node's main purpose is to keep records of all the slave nodes present in its cell. Master nodes also communicate frequently with neighbouring master nodes and they exchange slave node lists. This ensures that the master node knows of all possible nodes that may enter their cell. They also

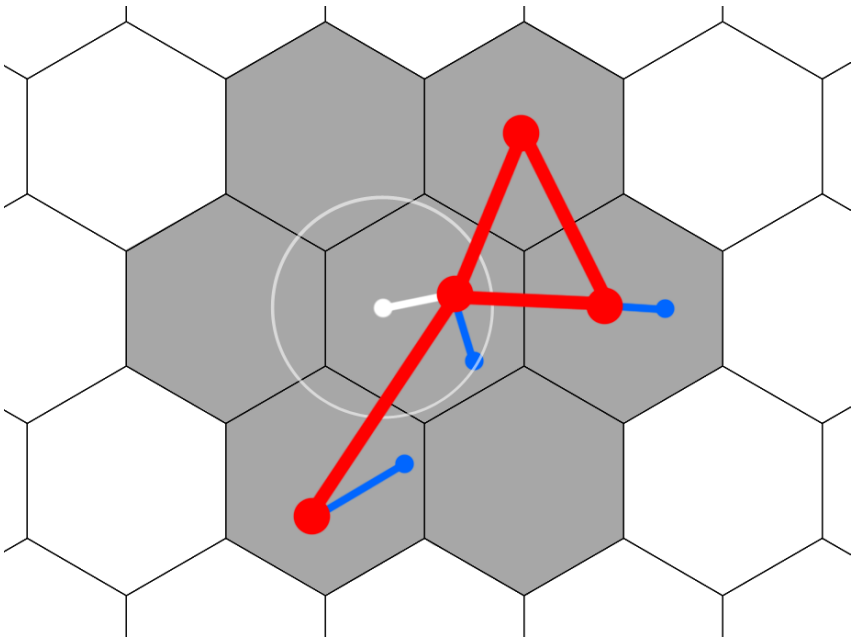

Figure 3.4: A section of the MOPAR network.

act as lookouts, informing slave nodes of any nodes moving into the cell, allowing the slave nodes the opportunity to populate their neighbour lists with the information of the incoming neighbour. Aside from the above-mentioned tasks, master nodes still perform the same functions as slave nodes.

- **Slave nodes** All nodes in the game world that do not perform a master node function are considered slave nodes. Slave nodes only store information that pertain to their direct neighbours in their AOI and their master nodes. Slave nodes can subscribe to multiple master nodes. Since the node's AOI can span several hexagonal cells, the slave node needs information regarding the activity of slave nodes in their neighbouring cells. To do this, slave nodes subscribe to one or more master nodes that are responsible for all the cells that intersect with their AOI. Slave nodes communicate their movement updates directly with one another to ensure that all their neighbouring nodes remain up to date.

Figure 3.4 shows a section of the MOPAR network. The large red dots are the master nodes of each cell. They are connected to other master nodes. The greyed cells represent all the cells that the master node at the centre is "interested in". It keeps the addresses of the master nodes of these cells, as well as a less up-to-date list of the nodes in those cells. The smaller blue dots are slave nodes. The blue lines from them to the red dots depict the master nodes that they are connected to. The white node and the white ring around it represents a node and its AOI. The cells

that the white ring overlaps with represents all the cells that the white node will be interested in.

3.5.3.2 MOPAR join and move operations

When a node first enters the MOPAR system, it is given an overlay key value, indicating its position in the overlay network. This key represents the node's position in the Pastry overlay. Pastry is designed to store the addresses of a small number of its numerically closest nodes in the circular name space that makes up the Pastry routing table [57]. After Pastry has assigned the joining node an overlay key, the node queries its neighbours in the Pastry overlay to determine if the joining node has to take over the home node responsibilities for a certain cell. Since Pastry routes messages that are destined for a certain overlay key value to the node with the nearest key value in the overlay, the home node must always be the node with the closest key value to that of the hexagonal cell. Therefore, if a new node enters the overlay with a key value closer to a cell's overlay key value, the new node assumes the home node responsibility. If the current home node contains a master node registration, the retiring home node passes the master node registration on to the new home node. To ensure that the responsibilities of home nodes are always fulfilled, copies of the master node registrations are stored on the Pastry overlay neighbours of all home nodes and they are informed of all master node changes by the home node. This ensures that if a home node disconnects or fails abruptly, the next-closest node to the cell's overlay key is promoted to home node and is already up-to-date with master node registrations.

The newly joined node is also given a position in the two-dimensional game world, represented by Cartesian X:Y coordinates. This position determines which hexagonal cell the node occupies. The node can determine which cell it is currently occupying with a pre-defined algorithm that determines the location, size and shape of the game world's hexagonal cells. By making a SHA-1 hash of the cell's coordinates, the node can determine the overlay key of the cell it is occupying. A message is then routed through Pastry to that cell, and the cell checks whether a master node is registered to it.

If a master node is already registered to the cell, the master node is informed of the newly-entered node and adds the new node to its list of slave nodes. The master node then informs its existing slave nodes of the new slave node joining the network and sends a join acknowledgement message to the newly joined node, which contains the position and addresses of all the other nodes in the newly joined node's AOI. The newly joined node receives this message and sets the address of the message sender as its master node's address. The remaining information in the join

acknowledgement message is used to populate the list of neighbouring slave nodes. The node is then successfully registered as a slave node.

If a newly joining node contacts the home node of a cell and no master node is present, the entering node is immediately promoted to master node and its address is registered on the home node. The joining node then takes on all master node responsibilities for that cell. Since the joining node is the first node in the cell, no slave node messages are sent out. However, a newly registered master node sends out messages to all the surrounding cells via the Pastry overlay, to notify the home nodes of those cells to inform their respective master nodes (if they have one) of the newly registered master node in their neighbouring cell. The master nodes then respond to the newly registered master node with their address and a list of slave nodes theat they are responsible for.

Since master nodes are subject to a higher degree of network traffic and message handling than slave nodes, slave nodes inform one another of their positions periodically. They do this by frequently exchanging movement updates through direct communication. Slave nodes do not send movement updates in the same manner or at the same frequency as they send to other slave nodes. Instead, they only inform their master nodes when the direction of their movement changes. The master node then uses dead reckoning [53] to determine the new position of the slave nodes. This reduces the amount of messages the master nodes have to handle in a given time period.

When a slave node moves from one cell to another, the slave node polls the master node for the address of the master node of the cell it's moving into. The current master node replies with the new master node's address and updates the leaving node's information to reflect that it is now under the responsibility of another master node. The old cell's master node is still aware of the node by proxy; the neighbouring master node will exchange its slave list with the old cell's master node as part of the standard operating procedure. The leaving node will also communicate with the master node if the slave node's AOI intersects the leaving cell.

3.5.3.3 MOPAR design benefits and limitations

MOPAR draws benefit from the hybrid structure by using the DHT to maintain global connectivity, while using the unstructured P2P communication methods to minimize the network overhead that is associated with the $O(\log n)$ steps required for a message to be routed from source to destination. The DHT is only used to maintain the hierarchical structure of the system. Theoretically, if no new nodes join the overlay and all nodes stay in their respective cells, the DHT will not be used to route any messages.

The hybrid approach makes MOPAR more fault tolerant than a purely unstructured architecture. The use of the DHT allows single node failures to be detected and handled. Thanks to MOPAR's built-in replication mechanism and Pastry's routing, a single node failing will cause the failed node's nearest neighbour to receive all intended messages instead, and it will replace the role of master or home node, depending on the nature of the messages it receives.

The documentation on MOPAR was sparse, and no existing implementation of it existed within the OverSim environment. Using the OMNeT++ development environment [66] and the Pastry API, a MOPAR simulation was created in the OverSim environment. This simulation is not highly optimized, but it serves the purpose of this thesis.

3.5.4 FullConn - Naive baseline unstructured architecture

The fully-connected approach, named FullConn, was created as a naive architecture used as a worst-case baseline. FullConn uses mutual notification to send event messages to all nodes in the game world. When a node first joins, the bootstrap node informs the newly-joined node of the address of an existing node in the overlay, and the existing node simply informs the joining node of each active node in the overlay. There is no AOI and no active interest management. Nodes are only removed from neighbour lists when they finally leave the network. This architecture is basically a very un-optimized version of the existing Vast architecture as described in section 3.5.2. It is used as a baseline with which to compare the effects of interest management on bandwidth usage in the simulations.

3.6 Summary

The above-mentioned interest management approaches were made available in the OverSim simulation environment. Their differences are summarized in table 3.1. In chapter 4, we will discuss the OverSim simulation environment and how it is used to implement the above-mentioned interest management schemes.

	SimMud	Vast	MOPAR	FullConn
Game world partitioning	Static squares	Static hexagons	Dynamic Voronoi cells	None
Communication approach	Super-peer	Mutual notification	Hybrid	Mutual Notification
Interest granularity	Coarse	Fine-grained at high avatar density	Less coarse	Non-existent
Update frequency	Infrequent (prediction)	Per update	Slaves: per update.Masters: on slave exiting or entering cell.	Per update
Message hops	O(log n)	1	O(log n) for master node registration, 1 for all other	1
Fault tolerance	High (Pastry overlay)	Lower than others (lazy leave detection, topology separation)	High (master node replication and Pastry)	Lower (lazy leave detection)

Table 3.1: Interest management schemes compared

Chapter 4

Implementation

Chapter 3 provided descriptions of the different interest management schemes as well as some speculated implementation benefits and drawbacks to each design. This chapter details the OverSim [6] simulation environment and how the different interest management schemes have been implemented in OverSim.

4.1 OverSim

OverSim is a peer-to-peer and overlay network simulation framework. It is built as an extension for OMNeT++ [66], a discrete event simulator for modelling distributed systems. The OverSim simulation environment allows for new P2P communication protocols to be implemented and evaluated. Many well known overlay protocols are already implemented in OverSim, and new applications can be built on top of these existing systems. Figure 4.1 shows a screen capture of an OverSim simulation a few seconds after its commencement.

4.1.1 Motivation for using OverSim

OverSim offers many useful features that make it an ideal environment to evaluate the performance of the chosen P2P interest management schemes.

By using the simulations environment, as opposed to a distributed or cloud based solution, the need for redeployment and setup of multiple devices, or instances of devices in the cloud, is removed. Small changes can be made, executed and the results studied faster than would be possible with a fully distributed system running in real time. Being able to observe the changes in each node's data structures and variables offer the chance to find errors more efficiently, and having access to the local information stored in each node makes finding values such as player drift (discussed in section 5.1) much easier. Global information, such as total number of packets sent

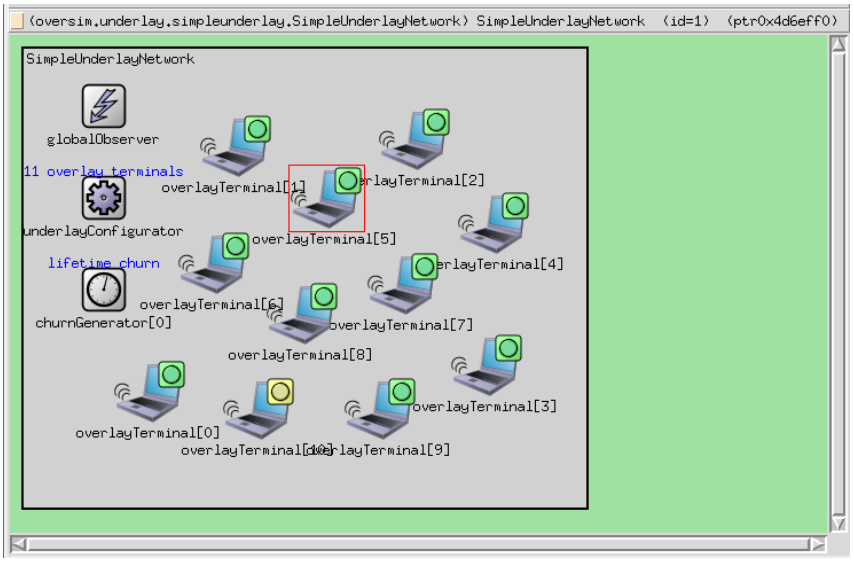

Figure 4.1: A screen capture of an OverSim simulation.

or average message delay, can be gathered with a global observer. Each node in the simulation can also gather statistics and store it to be evaluated later. OverSim also features built-in plotting tools and a graphic user interface of the simulation. This depiction of the network can be paused or viewed at set speeds to visually follow messages as they are routed through the network.

Cloud computing is a viable option for testing the effectiveness of interest management schemes, as is 'piggy-backing' off of existing applications through the use of middleware, such as the Koekepan project [20] designed to use Minecraft as a platform for testing new P2P NVE architecture. The largest factor in the decision to use the OverSim platform was the ease of gathering and plotting data from all avatars in the game world. Being able to iterate through each avatar and extract the necessary measurements allows for much faster access to results.

4.1.2 Simulation structure

The foundation of the OverSim simulation is the underlay network. The underlay network determines the type of nodes that are used in the simulation. For the interest management schemes in question, the "simple" underlay was chosen. This underlay setting calculates the network delay incurred between each node based on coordinates in a two-dimensional Euclidean space. The positions of each node is chosen to match the latencies of the CAIDA/Skitter Internet topology mapping project [18]. To make the simulated network more heterogeneous, nodes are also assigned different jitter parameters and bandwidths.

To simulate network churn, i.e. the effects of nodes entering and leaving the

network, OverSim offers three churn generators; “no churn”, “Pareto churn” and “lifetime churn”. The “no churn” generator adds nodes to the network until it reaches a number specified in the simulation parameters. The nodes in the network remain there indefinitely. The lifetime churn generator adds nodes to the network that have a specified average lifetime, sampled from an exponential distribution. Once the node has reached its determined lifetime, it disconnects from the network and soon the churn generator adds a new node to the network. The Pareto churn generator determines the average lifetime of a node by sampling from a Pareto distribution, and requires that a mean dead time (the duration that a node is present, but inactive) also be specified.

4.1.3 Node structure

An OverSim node is built in a tiered structure. These tiers are used to separate internal components of the node, such as the communication layer, the overlay network and the game logic. The communication between the tiers in each node are instantaneous, while the communication between the nodes is determined by the underlay. The node contains the layers TCP, UDP and Overlay, as well as a series of Tiers numbered from 1 upwards, depending on the complexity of the design.

The TCP and UDP are transport level protocols as described in the OSI protocol stack specifications. They allow for messages to be passed from one node to another in OverSim.

The overlay layer is where information regarding the P2P overlay is located and processed. The overlay could be an experimental application that needs to be evaluated, or a well-known scheme like Pastry [57], Chord [63] or mutual notification [38].

The layers above the overlay are tiered on top of one another. Tier 1 can communicate with the UDP, TCP and Overlay layers, as well as Tier 2. Tier 2 can communicate with UDP, TCP, Tier 1 and Tier 3, and so on. Communication with other nodes can only occur through the TCP and UDP layers. Internal communications occur between tiers, as information filters through the node to be processed at the appropriate layer in the application.

Each interest management scheme applies the Tier layers differently. The common trait is that the top-most tier is a “SimpleGameClient” module. This module takes on the role of the player steering the avatar around the game world.

The game client’s movement can be set to be a random walk or for the nodes to move as a group. In each case, the game client module informs the rest of the layers below of its movement by sending internal messages to the lower tiers. These tiers then process the movement information to determine the appropriate course of

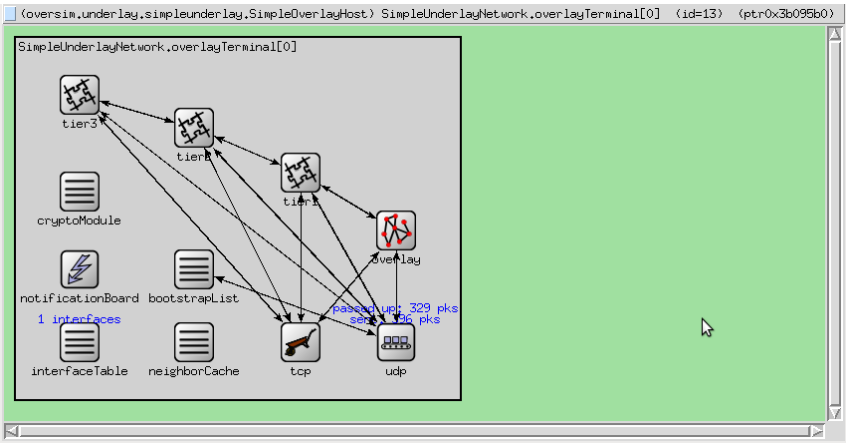

Figure 4.2: The tiered structure of a SimMud node in OverSim.

action. This would include sending update or unsubscribe messages to other nodes in the network, or removing a local instance of a neighbour.

Figure 4.2 shows how the internal structure of a SimMud node looks in the OverSim simulation. The main components are the TCP and UDP modules that handle communication, the overlay module which houses the logic used to route messages and store objects (in the case of SimMud, it would be a Pastry overlay), and the tiers that reside above the overlay. Each tier houses different modules based on the interest management scheme used, but in each scheme, the topmost tier houses the SimpleGameClient module, which controls the movement of the avatar through the game world.

4.2 Simulation constraints

The implementations of the different interest management schemes are quite simplified. Certain aspects of the functionality of an average MMOG have been removed for the sake of simplicity, and to prevent the data from being skewed due to extraneous bandwidth and processing power used to maintain aspects of the MMOG that are not relevant to this study.

- All game states are transient, and apply only for the duration of the simulation. No information is transferred from one simulation to the next.
- Object and NPC hosting are not implemented in the schemes. The different types of object hosting (region-based vs distance-based) adds another variable to the simulations which would alter the results of the schemes. Object hosting is therefore eliminated for the sake of simplicity.

- Only player movement updates are used as events in the simulations. This eliminates player-player and player-object interaction. Since movement updates are the most common messages sent in an NVE and occur more frequently, position state consistency will serve the purpose of evaluating how consistent the state information is between nodes in the network.
- There are no in-game obstacles to prohibit movement or occlude player vision, other than the edge of the map.
- No load-balancing mechanics are implemented. This helps determine the performance of an IM scheme regarding player density.
- No cheating is assumed. None of the nodes attempt to flood the network, intercept packets or spread malicious messages.

4.3 Interest management scheme implementation

Some of the chosen schemes are already implemented in OverSim. These were used as templates for the construction of the different interest management schemes.

In each scheme, only player movements are considered for updates. All game state information is also transient.

This section gives a brief overview of how the OverSim implementations of each of the chosen interest management schemes were achieved. Each of the schemes have different configuration settings, some of them universal and some unique to each scheme. The detailed functionality of each scheme is described in chapter 3.

The following parameters have been set in place as constant throughout each scheme and each simulation.

- The radius of the AOI is 50 in-game units. The size of the AOI contributes to the number of nodes that a single node subscribes to, and therefore directly influences the measurement of state consistency and bandwidth consumption. However, player density, which is a factor of the number of nodes in the game world and the size of the world, contributes to the same factor. It is therefore chosen to keep the AOI constant while changing the player density. With FullConn, this setting does not apply, as there is no interest management applied in the FullConn scheme.
- The size of the game world is a square. The length of the side of the square is fixed at 1000 in-game units.

- The churn generators are set to lifetime churn. Each node randomly selects its expected lifetime from a Weibull distribution with a mean lifetime of 100 seconds.
- The avatar representing each node moves at 5 in-game units per second, and sends movement updates 6 times per second.
- The global statistic module collects information every second. These are stored as either vectors, depicting a value such as connectivity over time, or scalars such as average and maximum bit rate.

The following parameters will be adjusted with each simulation:

- the interest management scheme that is used
- the number of nodes in the overlay
- and the movement type (random walking, group walking or real-world traces).

4.3.1 Movement models

There are three types of movement models used in each of the simulations. They are as follows:

- **Random movement** This model presents players with a set of movement instructions that are generated pseudo-randomly. Each avatar is instructed to move to random position in the game world, and moves independent of any other players.
- **Group-based movement** This model simulates players roaming the game world in groups. The group size is passed as a parameter, and the movement generator sends each avatar movement instructions that keeps the group moving in roughly the same direction.
- **Trace movement** This model mimics the movements exhibited by players on an official server of the popular MMOG World of Warcraft, as recorded by Francois Nolte [52]. The movement updates are broadcast to each avatar at fixed times, and the nodes are forced to those positions. Section 4.3.2 elaborates on this mechanic.

Each of these movement models will test the performance of the interest management schemes when exposed to differing player movement. The different movement models represent different types of player behaviour. Random movement would, for example, be representative of players exploring and interacting in the world on

their own, while group-based movement is exhibited when players quest together to accomplish more difficult objectives. How these interest management schemes perform under different player densities (clustered in groups vs spread apart) is an important factor to take into account during the design of a P2P MMOG.

4.3.2 Trace churn generator

The real-world traces were obtained from a project done by Francois Nolte, a University of Stellenbosch student for the completion of his undergraduate engineering course [52]. These traces are obtained from an official World of Warcraft server, depicting the movement of players near the Stormwind auction house, one of the game's most populated areas. The traces are recorded from the perspective of a single player receiving movement updates from the server.

The trace simulations will run for 878 seconds, and the game world is 350×350 units. The player updates occur between 6 and 8 times per second. Even when players are standing still, movement updates are broadcast. Due to the fact that player traces were recorded from an avatar's perspective, the trace movements are subject to network latency. Many of the trace movements are clumped around a small window in each second. For example, a small section of movement the avatar with the IP of [1.0.0.7] in the simulation experiences is displayed in table 4.1. For the duration of the simulation between 537008 milliseconds and 538008 milliseconds, the bulk of the movement updates happen during the last 100 milliseconds. It would be possible to normalize these movements to occur more smoothly, but that has not been done in these simulations.

The trace churn generator simulates the walking patterns of players as it was captured in-game. The generator reads a file that contains plain text, indicating the index of the player, the action to be taken (JOIN, MOVE, DELETE) and the new position of the node. The times between movements are not completely homogeneous, since packet loss, latency and other real-world network issues affect the speed of the delivery of the movement updates.

At the beginning of the trace simulations, the trace churn generator reads the trace file and builds a time line of join, move and leave messages for all the nodes present in the trace file. This allows the simulations to experience movement behaviour similar to that of actual players, given that trace information can be obtained.

4.3.3 Vast

The Vast implementation in OverSim remained mostly unchanged. In the iteration used in this thesis, only minor changes were made to the global statistics gathering

Time stamp (ms)	X-position update	Y-position update
536924	137.899	151.725
536938	137.899	151.725
536952	137.899	151.725
536966	137.899	151.725
536979	136.983	152.533
536993	134.926	152.655
537008	133.740	152.456
537922	133.740	152.456
537936	133.553	153.352
537951	132.993	155.363
537965	132.298	157.363
537980	131.630	159.141
537994	130.588	161.436
538008	129.399	163.568
538923	127.506	166.081

Table 4.1: A sample of the movement updates produced by a node in the trace movement scheme.

module.

Vast has only one tier above the overlay; the SimpleGameClient tier. Because the overlay is unstructured, the game logic and the overlay are placed in the same module. The Vast module serves to both interpret the movement updates from the tier above and handle incoming and outgoing messages. The average Vast move message is roughly 100 to 120 bytes in size.

Vast does not need any additional configuration for the simulation. The overlay is unstructured, so there is no configuration to be done to it, and the game world is irregularly shaped and dynamic thanks to the use of the Voronoi diagram, so there are no dimensions needed for the construction of the cells.

It should be noted that the implementation of the Vast simulation in the OverSim environment was not created by anyone directly involved with the research and development of the Vast IM scheme. Correspondence with Shun-Yun Hu, co-author of several Vast-related papers [31] [30] [29], brought to light that some suboptimal behaviour could arise because of this third-party implementation. Among the features not implemented is a mechanism for effectively handling nodes leaving the overlay without proper leave notifications.

4.3.4 SimMud

The SimMud implementation was altered to include a global statistics gathering module, similar to Vast. The functionality remains mostly unchanged from the implementation in OverSim.

SimMud has three tiers above the Pastry overlay: tier 1 hosts a Scribe module, tier 2 is the SimMud implementation and tier 3 hosts the SimpleGameClient module. As with the other schemes, the topmost tier sends movement information to the game logic tier, which then interprets the movement data and sends messages to the appropriate tier to be disseminated. SimMud movement updates are between 80 and 85 bytes per message.

The SimMud simulation is configured to divide the game world into a number of square cells.

4.3.5 MOPAR

The MOPAR scheme was absent from the OverSim environment. Given the documentation [69], an OverSim version of MOPAR was implemented.

OverSim's implementation of Pastry was used in the overlay tier, with the TCP and UDP modules remaining the same. Tier 1 contains the MOPAR module, connected to the communication layers and the overlay below, and the SimpleGameClient module in tier 2 above. The configuration parameters are read when a node starts and the MOPAR module uses this information to construct the hexagonal grid that the game world is divided into. The MOPAR tier receives the information about the player movements from tier 2, and then instructs the Pastry overlay to take various actions, such as subscribe, unsubscribe, send updates to neighbouring nodes and so forth. The Pastry overlay routes the necessary messages while the MOPAR module sends update messages directly to the nodes that are interested. MOPAR movement updates between slave nodes are roughly 120 to 130 bytes, however messages between master nodes vary, especially when handing off master node duties or exchanging neighbour lists.

The MOPAR implementation is not optimized. Since the literature was sparse, a perfect or efficient reconstruction, with all the necessary assumptions and optimizations, could not have been made.

MOPAR's configuration settings include the length of the hexagon's side. This is used to create larger or smaller hexagons on the game world's region grid. The configuration also specifies a replication value that denotes how many replicas of a master node are made on its overlay neighbours, as well as a timer denoting how often these replicas are updated.

4.3.6 FullConn

FullConn is a naive baseline implementation of the mutual notification approach to interest management. Each player in the game world is constantly aware of all other

players. This will result in a highly connected network, but will most likely incur large bandwidth costs.

FullConn has a tier structure similar to Vast, with the overlay housing the game logic and tier 1 housing the SimpleGameClient module. Since FullConn uses a similar mutual notification scheme, the movement update message sizes are similar, between 100 and 120 bytes.

4.4 Summary

In this chapter, the experimental setup for each of the four interest management scheme, as well as some of the necessary structure of the OverSim simulation environment are discussed. The constraints set upon the game world and the type of player churn experienced are also stated. Each of the four interest management schemes will be simulated under the same constraints.

Chapter 5

Experimental Results

In this chapter, the results of the different simulations are analysed. Section 5.1 discusses the metrics used to compare the performance of the interest management schemes, and section 5.2 makes predictions as to the outcome of each set of simulations. Section 5.3 describes the results of the tests for the existing OverSim implementations of the SimMud and Vast interest management schemes as compared to the results found in the literature. Section 5.4 outlines the different parameters that are given to each of the interest management schemes and their constituent modules. Section 5.7 compares and explains the results of the tests laid out in table 5.1.

5.1 Metrics

The primary objective of this thesis is to determine the effectiveness of the proposed interest management schemes in keeping a consistent, shared game state between peers in a decentralized MMOG application. The following metrics are chosen to help determine the effectiveness of each scheme; average drift, number of missing nodes, average bit rate, average messages rate and connectedness. Each of these metrics are defined as follows:

- Drift is a measure of how far the local copy of a node's position differs from the authoritative copy. It is the Euclidean distance (in in-game units) between the local copy of an avatar, as kept in the neighbour list of a neighbouring node, and the actual position of the same avatar contained on the node controlling the avatar. The average drift experienced by a node, call it node T is calculated as $D = \frac{1}{L}\sum_{i=1}^{L}|E_A(i) - E_P(i)|$, where L is the total number of neighbours present on node i's neighbour list. E_A and E_P are both arrays of length L. Array E_P contains the two-dimensional coordinates of the local copy of each

neighbour in T's neighbour list, i.e. where T perceives each of its neighbours to be in the game world. Array E_A contains the two-dimensional coordinates of the authoritative copies of each of node T's neighbours, as residing on the nodes in control of each of the neighbouring avatars. The difference in the position between each pair of nodes at the same index is the drift experienced for that neighbour. The average drift per simulation is then determined by averaging all average drifts from each node in the simulation, expressed as $\frac{1}{N}\sum_{j=1}^{N} D(j)$, where N is the total number of nodes in the simulation.

- The number of missing neighbours per node should preferably be zero. This number is determined by checking the local neighbour count of each node against what the global view of the game world determines it should be, given the radius of the AOI and the node's position. The average number of missing neighbours is simply expressed as $\frac{1}{N}\sum_{i=1}^{N}(A_i - P_i)$, where N is the total number of nodes in the simulation, P is the number of neighbours kept on the neighbour list of node i, and A is the number of avatars present in node i's AOI, as determined by the position information stored on each avatar's controlling node. The missing neighbour count is also sometimes expressed as discovery consistency or topology awareness rate, and is measured by taking the number of neighbours that are present in a node's AOI over the number of nodes that should be present. Missing neighbours negatively affect the overall consistency and correctness of a player's view of the game world.

- The number of extra neighbours per node should preferable be zero. This number is determined by comparing the number of nodes in an avatar's interest set that are outside of its AOI. The average number of extra neighbours is expressed as $\frac{1}{N}\sum_{i=1}^{N}(P_i - O_i)$, where N is the total number of nodes in the simulation, P is the number of neighbours kept on the neighbour list of node i, and O is the number of avatars that are both in i's neighbour set and outside of i's AOI. Extra neighbours can add to the player's view of the game world, making him aware of avatars outside their AOI, but they add to the overall bandwidth use.

- Bit rate is determined by the size and frequency of messages sent through the network. The lower the bit rate, the less bandwidth used in the network. The average bit rate is simply expressed as $\frac{B_T}{S}$, where B_T is the total number of bytes sent over the course of the simulation, and S is the simulation time in seconds.

- Message rate is the measure of messages sent through the network. The less messages sent and received, the less bandwidth is consumed. The message

rate is directly related to the bit rate, assuming that all messages are the same size. In the simulations, some of the messages are larger than others, for example Vast's initial neighbour notification messages, but messages with large payloads such as these happen infrequently. The average message rate is expressed as $\frac{M_T}{S}$, where M_T is the total number of bytes sent over the course of the simulation, and S is, again, the simulation time in seconds.

- Connectedness refers to what percentage of nodes can be reached by starting at a single node, then recursively probing the neighbour list of each node and counting the total number of nodes discovered this way. In other words, using only the neighbour list of each node, what percentage of the network can be reached. The connectedness of each simulation can be expressed as $\frac{Max(C_1,C_2,C_3,...,C_N)}{N}$, where N is the total number of nodes, and C_n represents the total number of nodes that are reachable through recursive probing from the nth node in the simulation.

5.2 Expected Results

Given the metrics listed above, and the descriptions of the functionality of each IM scheme in chapter 3, some predictions can be made regarding the performance of the different schemes.

FullConn and Vast should have the highest connectedness rates. Each FullConn node keeps a list of each node in the network, so by definition it should have 100% connectedness at all times. Vast relies on its game world neighbours to act as lookouts for new incoming nodes. Its design forces a node to always keep at least its enclosing neighbours in the neighbour list. These two factors ensures that Vast should also have a connectedness of 100%. SimMud and MOPAR will most likely not have such high connectedness, unless the world is populated to a reasonable degree. Since these schemes rely on the underlying Pastry overlay to find the super peers of their respective networks, pockets of nodes can appear in the game world that do not connect with one another, until a node moves into a region connecting the two network segments. The neighbour lists of the nodes in the connecting region will then be populated with the addresses of the nodes in the separated segments. From there, the connectedness algorithm can find a larger percentage of the game world's population.

Drift is a result of network latency. If a node is not informed of the new position of its neighbours in a timely manner, the local copy's position is out of date. MOPAR will suffer less from this, since it has employed dead reckoning to offset the need for frequent master node update messages.

As depicted in Figure 2.2, the use of square regions is less effective at approximating a circular AOI than hexagonal regions. This could make the node subscribe to a larger area than the AOI instructs. The Voronoi partitioning is less likely to subscribe to a larger area than the AOI in the case of a higher player density. Lower player densities, however, may have the Vast node subscribe to nodes that are far outside its AOI to maintain global connectedness.

Chapter 5 discusses the accuracy of the simulations in comparison to results found in the literature. It also discusses the results of the simulations according to the setup and metrics discussed in this chapter.

5.3 Comparison of existing OverSim implementations

In order to determine if the existing OverSim implementations are correct, these implementations have to be tested according to setup as described in the literature. SimMud and Vast have already been implemented in the OverSim environment.

5.3.1 Existing SimMud implementation

The existing SimMud implementation in OverSim was slightly altered from its original state. A global statistics gathering module was added to the existing model to ensure that the desired results of the simulation could be recorded and compared.

In [37], multiple tests were run, comparing different aspects of the SimMud architecture, such as the effect of message aggregation, avatar population density and object storage on the overall performance of the network. To make the comparisons equal among the different IM schemes, some of the functionality described in the literature are not tested in this paper.

- Message aggregation is not applied. Since not all of the interest management schemes use something analogous to a super-peer, message aggregation is not applicable to all the schemes.
- Object hosting is not supported for the sake of simplicity.
- Inter-player interaction is not supported. Since direct player interaction, such as fighting against another avatar, can be considered as a different kind of state update, we use the most common one, movement updates, as the main interaction mechanic.

In the literature [37], the parameters of the test were the following:

- the game world is 1000×1000 in dimension,

- the game world is divided over 100 regions,
- 1000 nodes are connected to the game world,
- no churn is applied to the network,
- all nodes connect to the network nearly instantly,
- the avatar movement was set to a group-based movement scheme, and the average group size is 10,
- and there is no mention of any interest management outside the currently occupied region.

One aspect of the literature that could not be accurately recreated by the available movement models is the duration of time spent in each cell. The literature describes that each node remains in a region for no more than 40 seconds before moving to another region. With the random movement generator provided with OverSim, it can not be guaranteed that a node will remain in a cell for any particular amount of time. It can be assumed that if a node moves at 5 units per second over a region of 100×100, then the node would travel across the diagonal (with a length of 141.421 units) in just over 28 seconds. In the meantime, the movement generator can also cause the node to change direction at any time, which could further lengthen or shorten a node's residency in any region. On the other hand, crossing into the region over one of the corners might result in the node spending only a single move step inside a region before moving to another region. Using these assumptions, it can be said that there is no way of determining the average time that a node spends in a given cell beforehand.

[37]'s account of the results of the test specified above is as follows:

- The nodes receive between 50 and 120 messages per second, with an average of 82.17 messages per second. This number is regardless of message type.
- The average size of each message is 200 bytes. With 7 update messages sent per second and an average of 10 nodes per region, each node receives an estimated 70 messages per second, at a rate of 14 Kb/s.
- The vast amount of multicast messages (over 80%) arrive at the destination node in under 2 hops. There is, however, a long tail in the graph, with some messages delivered with more than 16 hops. This long tail is attributed to the idiosyncrasies of the Scribe architecture.
- Unicast messages are practically all delivered to their destinations in 6 hops or less.

5.3.2 Comparative results for existing SimMud implementation

The comparisons revealed different results than those shown by the tests in the literature. The parameters were recreated as close to those found in the tests performed in the literature [37], however the results found in the OverSim simulation revealed widely differing results.

- The average rate of messages received and sent on each node both exceeded 300 messages per second. This is substantially more than the average of 82 messages per second depicted by the literature tests. A part of the extra messages can be attributed to the increased frequency of region join and leave messages. These are a result of the random movement generator. As stated in section 5.3.1, it is very difficult to predict beforehand the average length a node stays in any particular region. Another reason is that the simulation's statistics tracking does not make a differentiation between master and slave nodes. The increased traffic experienced by the master nodes will add to the overall number of messages received on average.
- Avatars spend an average of 16 seconds in each region, well below the 40 second mark as indicated in the literature. All but one avatar out of a thousand managed to leave their node in under 40 seconds.
- All the messages are delivered to their destinations in 2 or less hops. This could stem from the fact that a large number of nodes connect in a brief window of time, allowing the Pastry overlay to generate more leaf and neighbour nodes before the majority of the movement updates are sent out. Another possible reason for this improvement over the tests performed in [37] could be due to improvements in the Scribe overlay, or more specifically, an improved implementation of the Scribe overlay in OverSim.
- The average number of bytes that arrive at each node is roughly 17 Kb/s. This number is close to the 14 Kb/s that is estimated in the tests described in the literature. However, nearly the same number of bytes are sent per second by each node. However, in this experiment, no distinction is made between messages sent by slave nodes or master nodes, so the increased traffic experienced by the master nodes will skew the results to indicate a heavier flow of network traffic.

Given the information above, we can conclude that the simulations will produce at least twice the bandwidth of the results found in the literature. The message count is significantly higher than the literature results, so this will have to be factored in when the final results are analysed.

5.3.3 Existing Vast implementation

The existing implementation of Vast in OverSim was altered only in the parameters that are recorded by the global statistics gathering module. Although the initial Vast paper [31] does not contain any actual metrics or results, the paper [29] contains information about the implementation of the Vast overlay on a real-world network. The version of Vast that is described in [29] makes use of relays (super-peers), so most of these results are not of a strictly unstructured P2P architecture. While the aforementioned paper changes the amount of relays in a given simulation, setting the number of relays to its maximum, i.e. all 90 nodes act as relays, the network simulates a fully distributed virtual environment. Therefore, only the furthest data point in each of the graphs in [29], section IV are relevant to this study. But, for the purposes of determining the correctness of the simulation setup, this information will suffice.

A simulation was set up according to the parameters in [29] for comparison with its results, in order to determine how much the implementations differ. The use of super peers may help ease the strain placed on the network by heterogeneous node capabilities (processing power, connection bandwidth, etc.), but these effects would not be as pronounced in a homogeneous system, such as the OverSim simulation environment.

The Vast simulations were set up to represent a SecondLife [42] region on the PlanetLab distributed computing platform [65]. The region is a square that is 256 units long on each side. The AOI of each node is a circular area with a 64 unit radius. There are a total of 90 nodes in the region, forming and moving in groups of 6. Each movement step is 5 units long, and 10 movements are performed per second, for a total of 50 units per second. These conditions and restraints were replicated in an OverSim simulation with the provided Vast code. As stated, this test does not take into account the effect of relay nodes in the network, so only the base results will be compared where the information is available. Each node will only maintain records of the 10 closest nodes, so as to avoid having to store too much information and maintain too many connections. This also helps clients connect to other relay nodes in case of the failure of any other relays.

In [29], the nodes in the network moved in either a clustered pattern or a random pattern. As stated above, only the fully distributed version of the network is relevant to our measurements. To that end, the following information can be obtained from the literature:

- that in the fully distributed version of the Vast experiments, the average latency between nodes is roughly 110 ms,

- the average drift is 5 in-game units per node,
- the discovery consistency of the nodes (the number of perceived neighbours in the node's AOI over the number of nodes that should be present) is 99.8%, a near perfect awareness if each node's neighbours,
- and each node consumes as low as 35 kB/s of bandwidth.

Given how the Vast system is set up, it's not surprising that the discovery consistency is so high. There is a high level of direct interconnectedness between any two nodes in the network. A node informs any of its neighbours whenever a node that would potentially become of interest to them enters the area, and a direct connection between the two newly neighbouring nodes are established.

5.3.4 Comparative results for existing Vast implementation

One fundamental difference between the test done in the literature and the simulations performed in this thesis in the OverSim environment is the size of the messages. Given the information stated in [29], it would appear that a single move message averages about 50 bytes. As stated in section IV part A of the aforementioned paper, an ordinary client has an average upload rate of around 0.5 kB/s. Given that there are 10 movement steps per second, and assuming that a move message is broadcast on each movement update and that the majority of updates sent between nodes, that would translate to just under 0.05 kB per message, or about 50 bytes. Therefore, 0.5 kB/s is the bandwidth that one node in the network would consume sending its movement updates to one other node in its neighbour list. Another difference between the scheme described in [29] and the Vast implementation as described in other papers, such as [33] and [30], is that no mention is made of neighbour list exchanges. This is because a scheme that uses relays or super peers would share information about the nodes that it is hosting or managing to other super peers in the network. Since, in a fully distributed system, only one node is hosted on a single 'super peer', that being the super peer node itself, and its information is sent to other nodes in the overlay, and of those, only to the neighbours in its AOI. In the Vast scheme that our simulation implements, neighbour lists are exchanged between neighbouring nodes. This helps ensure that any neighbours who move into a node's AOI is observed and added to its own neighbour list. This does, however, also increase the number of messages sent per second. This, in turn, directly increases the bandwidth used and the latency experienced. These points must be taken into consideration when judging the performance of the proposed Vast implementation as compared to the data gathered from the literature.

After running the simulations of the Vast network using the same parameters as the tests in the literature, the following information was obtained from the simulation that featured free roaming nodes.

- The average bit rate for the simulation was 515 kB/s. The majority of these updates consist of neighbour list exchange messages, accounting for 407 kB/s of the bandwidth. Move messages make up 32 kB/s of bandwidth, and requests for position information from a node's enclosing neighbours 76 kB/s.
- The average number of neighbours kept by each node is 21.8 nodes.
- The average number of missing neighbours on each node is 0.6 nodes. This equates to 2.75% of each node's neighbours being missing.
- The average drift is 37.4 units, 14% of the length or breadth of the game world.
- The average message latency between nodes is roughly 90 ms.

It is evident that these results are far removed from the results found in the literature. The majority of these disparities can be tracked to the fact that the Vast mechanism as described in [29] does not exchange neighbour lists on regular intervals, and the Vast implementation as provided by OverSim does. As stated in the list above, neighbour list exchanges account for the majority of the traffic in the network, with each node receiving over 400 kB of neighbour list information per second. Given the large packet payload, coupled with the more frequent movement rate and the position of the nodes on the CAIDA/Skitter network latency plot [18], the delay imposed by the simulation often exceeds one movement step, allowing the nodes to move multiple times before an move update is sent. Stated otherwise, node A would receive the move notification from its neighbour, node B, that indicate where B was several movement steps ago. In the meantime, node B has continued to move around, rendering node A's most recent information regarding node B's whereabouts incorrect. In the case of the simulation stated above, the network connection is assumed to be a 10 Mb/s Ethernet line. Because the simulation limits the speed that the UDP module to simulate the data rate, the UDP module builds up an enormous backlog of messages which are not processed quickly enough, further exacerbating the problem of determining the location of neighbouring nodes in a timely manner.

By slowing down the rate at which nodes update, the bottleneck caused by the limited data rate will be alleviated at the cost of a slightly less accurate view of neighbouring nodes. Whether or not this latency in player information is acceptable or not is dependant on the application, as discussed in section 3.1.

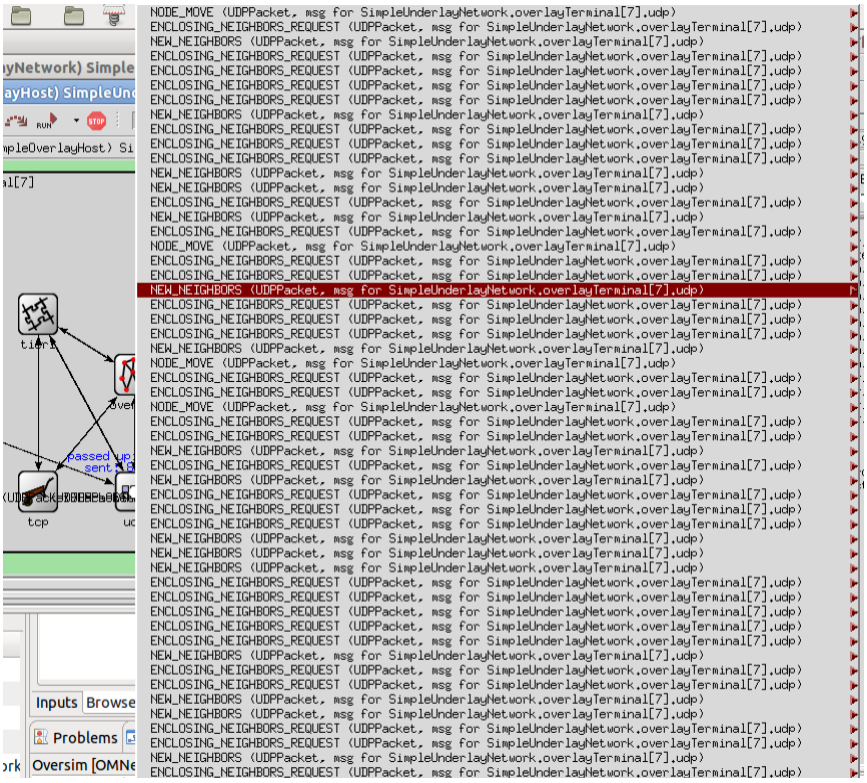

Figure 5.1: The backlog of messages waiting to be processed by the UDP module. This image was during the simulation performed using the parameters laid out in the literature.

5.4 Parameters used in comparative simulations

The aim of this work is to determine how the different interest management schemes; with their different world partitioning, overlay schemes and neighbour discovery methods; are affected by the behaviour associated with NVEs and MMOGs. This section describes the different parameters that are used during the simulations to determine the effectiveness of the IM schemes, and the metrics by which they are compared.

The four interest management schemes described in chapter 3 were implemented in the OverSim environment. Each simulation uses the following parameters.

- **AOI radius** The radius of the area of interest around each node.
- **Area dimension** The size of the game world to be moved in.
- **Node count** The average number of nodes in the game world.
- **Churn generator type** The type of churn that will be applied to the network. In the experimental comparisons in this thesis, only two types of churn will be

Test number	Average node count	Movement generator	Churn generator
1	10	Group roaming	Lifetime churn
2	100	Group roaming	Lifetime churn
3	1000	Group roaming	Lifetime churn
4	10	Random roaming	Lifetime churn
5	100	Random roaming	Lifetime churn
6	1000	Random roaming	Lifetime churn
7	N/A	Trace roaming	Trace churn

Table 5.1: The configuration for the experimental simulations

applied; lifetime churn and 'trace churn', a churn module that adds nodes to the game world as indicated in a specified trace file.

- **Movement generator type** The type of movement that will be applied to the nodes by the SimpleGameClient module. The movement generators used in these experiments will be random roaming (nodes moving in random directions for an arbitrary amount of time), group roaming (much like random roaming, but nodes will tend to flock towards one another in the game world and move randomly as a group), and trace roaming (nodes move according to information located in a specified trace file).

- **Movement frequency** The number of movement updates that are made per second.

- **Simulation time** The time that the simulation will be running.

For our comparisons, we will run the tests in the configuration specified in table 5.1. The table contains the variables that will change with each test run. Each of these seven tests will be applied to each interest management scheme in turn. The non-applicable amount of nodes described in test number seven serves to indicate that the amount of nodes is not controlled by the simulation environment, but by the information gathered from the trace file.

The tests in table 5.1 vary primarily in the number of nodes concurrently in the network, as well as the kind of roaming that the nodes perform. Since the game world's size remains the same size during each test, the node count serves to alter the player density in the game world. The player density in the game world can have a large effect on message latency and the bandwidth consumption in the network, which ultimately leads to outdated player states on each node.

The other variables not depicted in table 5.1 will all remain constant throughout the tests.

- The game world size will be set to 1000×1000.

- The area of interest will be a circle with a radius of 50 units.
- The connection channels between nodes will be capped to 10 MB/s.
- The players will move at a rate of 6 movement updates per second, at a speed of 5 game units per second. This will not be applicable to the trace roaming test, as the movement update rules of the game world, from which the movement traces were taken, will apply.
- The simulations will run for a total of 5 minutes. Again, the following does not apply to the trace test. The trace tests run for the full extent of the recorded data, which is roughly 860 seconds.

Each interest management scheme has its own unique variables that also have to be taken into account. Each of the modules also have their own set of default parameters that are set in the default initialization file. These serve to give parameters values when none are specified in the configuration. The following sections indicate the important scheme specific variables and their values in the simulations.

5.4.1 Variables unique to SimMud

SimMud divides the game world into regions, which it uses to determine the groups of the multicast trees used by the Scribe message dissemination system. Therefore, the amount of regions is an important factor to take into account. Too few regions will increase the area of each region, resulting in too much irrelevant information being sent to each player. Too many nodes will result in nodes changing regions more often, causing a lot of network overhead for the coordinators. Region change transactions rely on the DHT to find the appropriate coordinator, so the network overhead is greater and the message will suffer propagation delays.

In the experiments, 100 regions, arranged as a 10×10 grid, are used. This results in regions of 100×100. This will cause each node in the game world to be subscribed to more than one node at once, since the occurrence of a node in the exact centre of a cell is very scarce.

The Pastry overlay also requires a pair of configuration parameters. L is a configuration parameter that determines the number of leaf nodes that each node holds. Typical values for L are 16 or 32. The other variable, b, is usually set to the value of 4. The keys and nodeIds in the Pastry overlay are sequences of digits in base 2^b. The variable b can potentially increase the routing complexity, since the Pastry overlay guarantees that a message will be routed in $\lceil \log_{2^b} N \rceil$ hops. In the experiments in this thesis, all Pastry overlays use L with a value of 16 and b with a value of 4.

5.4.2 Variables unique to Vast

Apart from the default parameter values specifying, for the most part, the length of time it takes for a node to be declared timed out, Vast has no other parameters that need to be set in the simulation. Since the game world is dynamically partitioned, there is no need to specify any information regarding the size of the game world regions, such as is needed in SimMud or MOPAR. Vast does not use a list exchanging mechanism unless a node joins, so there is no need to configure list exchange timers or node backup timers.

5.4.3 Variables unique to MOPAR

Since the MOPAR scheme is built on top of Pastry, the same parameters apply to this scheme as the Pastry parameters in section 5.4.1. So, for the MOPAR scheme, L is 16 and b is 4.

The MOPAR scheme was built to accept parameters that determine how often the master nodes exchange slave lists, as well as how often the master nodes predict the movement of the slave nodes under their supervision. For these two parameters, a time of 500 milliseconds is chosen. This means that effectively the master nodes predict the movement of each slave node that it is responsible for every 3 steps. The MOPAR module also takes a parameter that determines the size of the hexagons that the game world is divided into. The parameter that is read from the initialization file describes the length of one side of the hexagon. The parameter is chosen as 60 in-game units. This results in the hexagons having a height of 120 units and a width of 103.923 units. This brings the area of the hexagon to 9353 units, or 95.53% of the size of one of the SimMud regions. The final parameter that is given to the MOPAR module is the replication factor. This refers to the number of nodes to either side of a home node that stored a copy of the neighbour lists held by the home node's respective master nodes. This ensures that, if a master node leaves, that an up-to-date backup master node will still receive the updates from slave nodes. The replication factor is set to 4 in the simulations.

5.4.4 Variables unique to FullConn

Since FullConn has no active interest management techniques, or even an AOI, there are no parameters. The game world is not partitioned, nodes remain aware of other nodes throughout their lifetimes. This is the most unoptimized scheme of all those tested. This is purely to establish a baseline for comparison.

5.5 Hardware setup

This section describes the computing system on which the simulations were performed. The hardware setup for each simulation is the same, since they were all run on the same machine at different times.

The simulation software runs on a 64-bit Ubuntu distribution of Linux as a virtual machine on a personal computer running a 64-bit Windows operating system. The computing system has an Intel i7 950 central processing unit, housing 8 logical cores which run at 3.1 GHz. It also has 24 GB (gigabytes) of RAM and 1.5 TB (terabytes) secondary memory. The Ubuntu virtual machine was allowed to use 6 of the 8 cores at 100% processing capacity, 20 GB of the available hard drive space and 18 GB of the available RAM. The reason for the use of the Linux distribution above the Windows operating system is the Windows version of the OverSim simulation software is not compatible with a 64-bit architecture, therefore severely limiting the memory available for the simulation. Using the 64-bit Linux version of OverSim allows the simulations to utilize upwards of 20 GB of primary memory. This allows the simulations to create many more nodes on a single machine, and for the simulations to store more information per node.

5.6 Scheme performance

This section compares the results of the tests shown in Table 5.1. Each of the simulations were run on the same machine as indicated in the previous section. The only exceptions are the 1000-node random movement and 1000-node group movement simulations for the FullConn architecture. The reason for the exclusion is because of the computational intensity of the simulation. The simulations were left to run for days, but multiple technical issues made the full simulation data unobtainable. Between power outages, restarts due to processor overheating, memory failures and other issues, the full course of the simulations could not be run. The machine running the simulations was monitored near-constantly (within reason), but the problems persisted. A rough, back-of-the-envelope calculation estimates the completion time at well over 3 weeks of real-time processing. It is safe to leave out these two datasets, however, because the FullConn architecture was designed as a baseline, indicating the most primitive approach to interest management, which is to say, no interest management at all. It is therefore safe to assume that the FullConn interest management scheme will be out-performed by the other schemes, at least in terms of bandwidth efficiency. It is more important that the information regarding the other interest management schemes are recorded and compared.

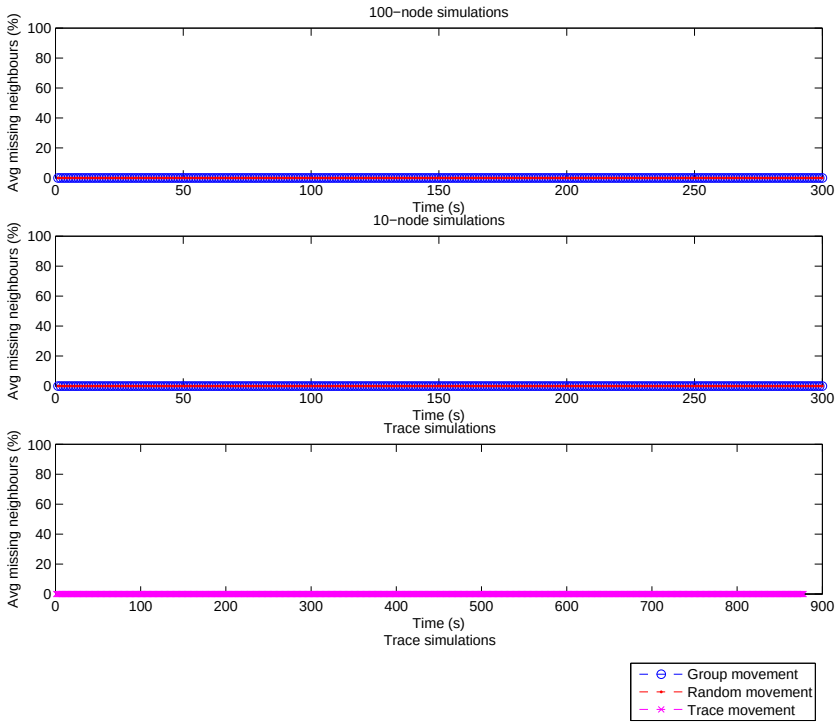

Figure 5.2: Average missing neighbours per node in FullConn simulations, as a percentage of the expected neighbours.

5.6.1 FullConn performance

FullConn exhibits the best possible performance of all the IM schemes, but at the cost of high bandwidth. It was only included in the simulations to act as a naive baseline to compare the other schemes, and to illustrate the inherent value of using interest management schemes to reduce bandwidth while still keeping a coherent world view.

It is evident that the performance of the FullConn IM scheme exhibits the perfect behaviour in terms of state replication across the network. Every node knows exactly where every other node is at all times. As shown in Figure 5.5, all nodes contain the correct, up-to-date information about each of their neighbours. In the case of the FullConn scheme, the number of neighbours allowed by each node is all other nodes in the simulation. The highest-connected node contains 100% of remaining nodes, and there is no drift (both evident in Figure 5.6). The drawback, as expected, is the high number of messages being sent and the relatively high bandwidth usage, as shown in Figure 5.6.

In a realistic network, a fully-connected IM scheme would cause severe network congestion and bandwidth use. The number of messages being sent per player in

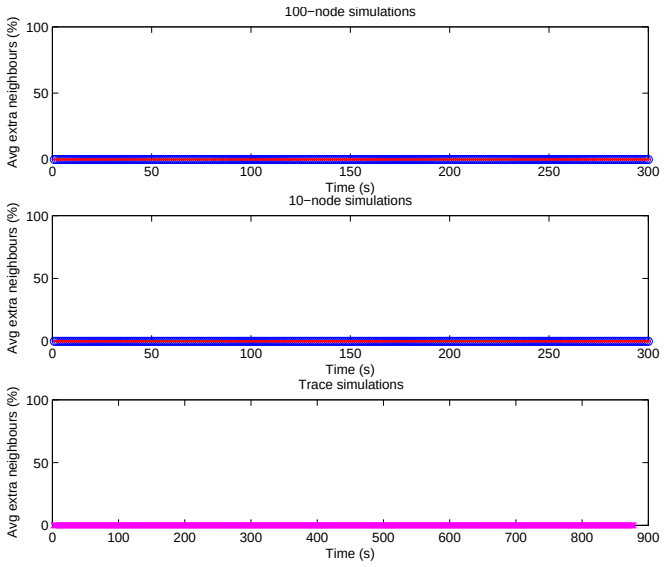

Figure 5.3: Average extra neighbours per node in FullConn simulations, as a percentage of the recorded neighbours.

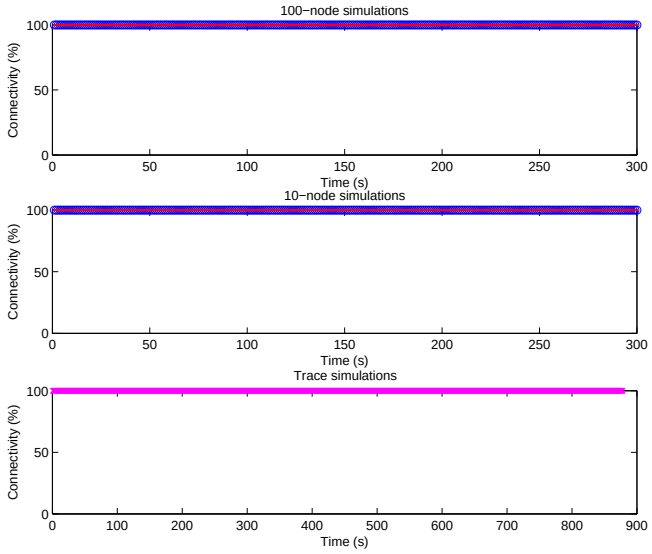

Figure 5.4: Percentage of connectivity throughout FullConn simulations.

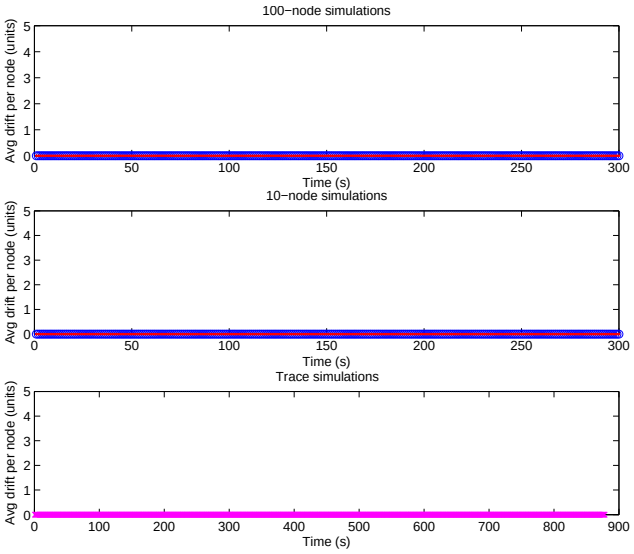

Figure 5.5: Average drift per node in FullConn simulations.

the network far exceed those of the other IM schemes. Note that the bandwidth consumed in these simulations seems low, between 60 and 90 kB/s sending and receiving for the 100-node simulation (Figure 5.6), the contents of each message is very sparse; it contains only routing information, a time stamp, node identification and the new position of the node. Most MMOGs require that much more information be sent to properly convey the necessary information regarding other game world objects and characters to a single player. It should also be noted that the other simulations used only a fraction of that bandwidth at the same population levels.

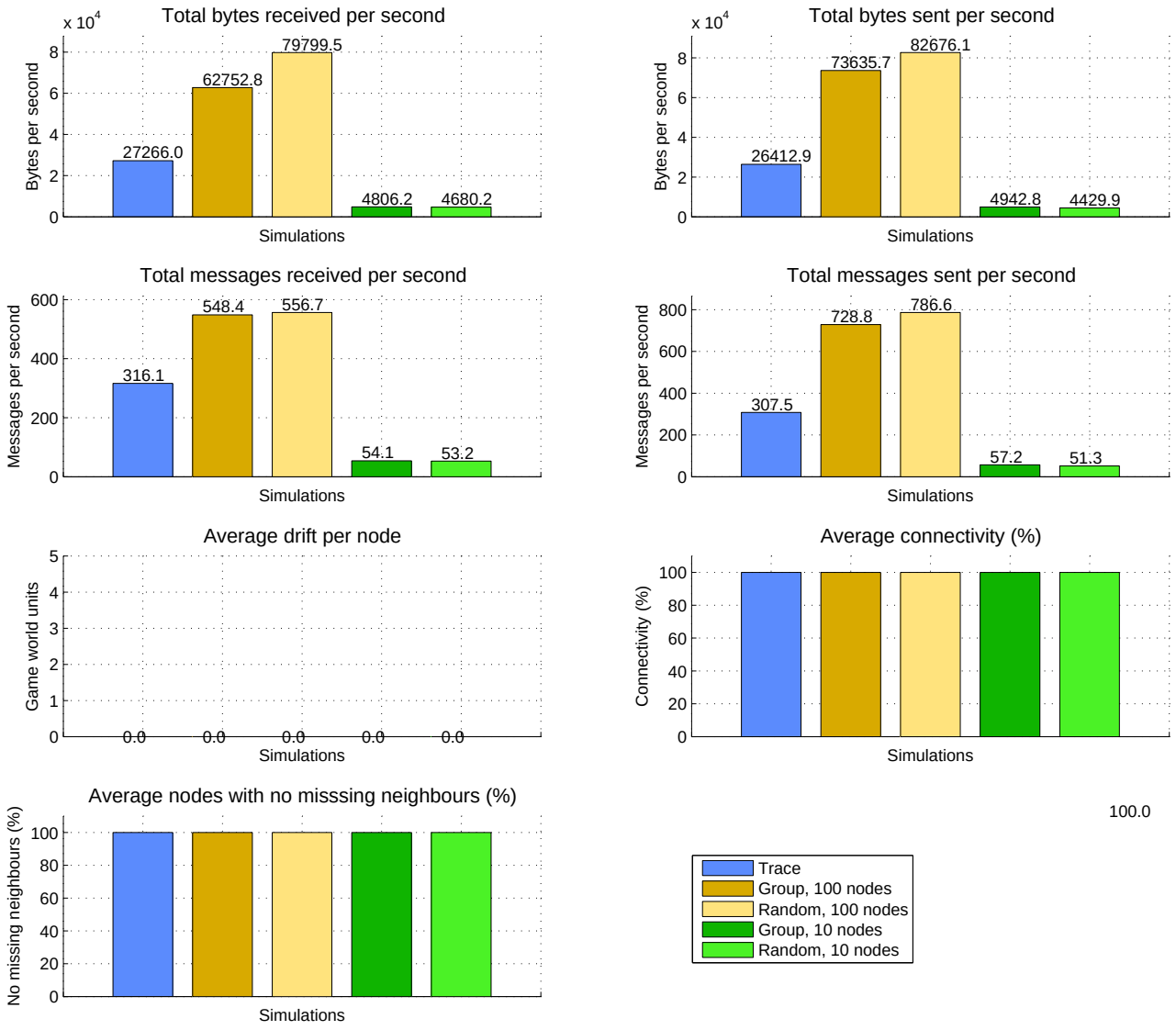

Figure 5.6: Scalar readings from FullConn simulations.

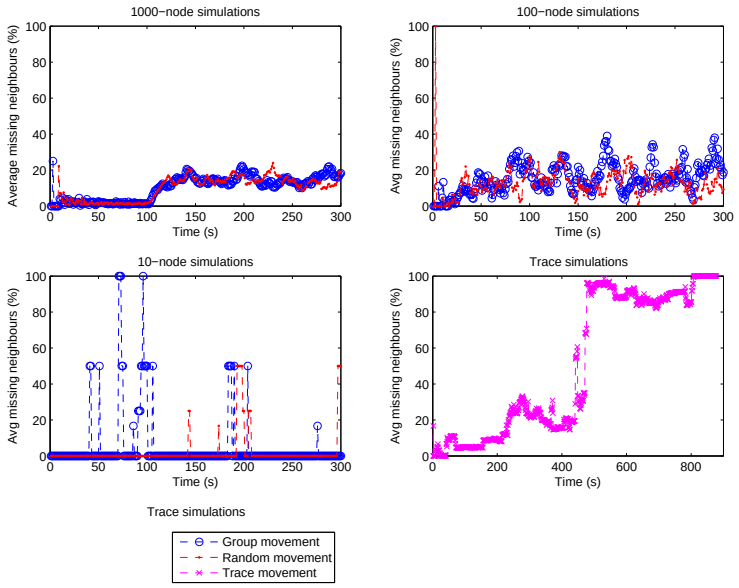

Figure 5.7: Average missing neighbours per node in SimMud simulations, as a percentage of the expected neighbours.

5.6.2 SimMud performance

SimMud, using the Pastry overlay network, is a more structured interest management scheme. This would allow for good performance when users are sparsely distributed around the game world, since the Pastry nodes maintain a list of neighbouring nodes in the Pastry key space, as well as random nodes, creating a message route that is not based on in-game proximity. This prevents the nodes in the game world from creating smaller clusters of peers that function independently and retaining some connectivity to nodes further away in the game world, under moderate failure conditions.

Figure 5.7 shows that the SimMud simulation has the lowest number of missing neighbours at high density, apart from FullConn. This is due to the robust routing scheme provided by Pastry and Scribe, and the self-organizing nature of the scheme helps to ensure routing will take place under nearly all circumstances. As the density decreases, the percentage of missing neighbours rises slightly, but still mostly maintaining a missing neighbour percentage of under 40% of the expected neighbours in its AOI. It is to be noted that grouped movement suffers more from missing neighbours than random movement, which would make sense; moving in groups results in more neighbours, and therefore a higher probability of neighbour updates not reaching the intended recipient. This is especially evident when comparing the missing neighbours of the two 10-node simulations. For the trace simulations, the missing neighbour count seems to be high all around. It is unclear at this point as

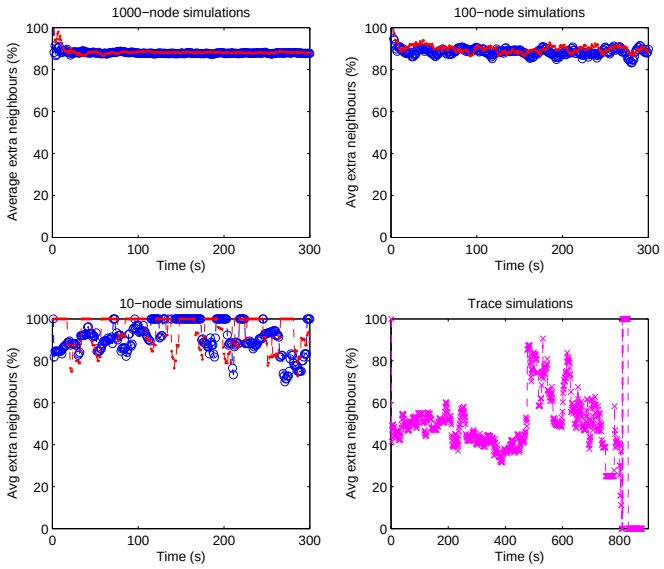

Figure 5.8: Average extra neighbours per node in SimMud simulations, as a percentage of the recorded neighbours.

to why the leaving of one node, again at the 486 second mark, resulted in such an increase in missing neighbours. The structure of the SimMud scheme should be able to repair itself after a coordinator leaves, but in the trace simulation, this seems to not be the case. It is unclear why so many nodes are missing neighbours.

Figure 5.8 shows the amount of neighbours kept in a node's neighbour list that is outside its AOI, as a percentage of the total neighbours recorded. It is evident that throughout the SimMud simulations, roughly 90% of each node's neighbours are nodes that occupy a position outside its AOI. This is to be expected of SimMud, as the use of the static world partitioning offers less granularity than other approaches when it comes to filtering superfluous neighbour messages. During the trace simulation, roughly 45% of each node's neighbour list is outside its AOI, until the same node leave at the 486 second mark, when the average number of extra neighbours rises and falls drastically, culminating in losing all neighbours after the 800 second mark, when multiple nodes leave in rapid succession.

Figure 5.9 shows that SimMud maintains a high level of overall connectedness in the higher-density simulations. When a node subscribes to more than one region, it gains neighbours from all the subscribed regions. This means that, in densely-populated areas, each node is aware of more neighbours, and each of those neighbours are also aware of other, different neighbours. This explains the high connectivity of the SimMud scheme in highly-populated game worlds. As the distance between nodes increases, each region is less likely to be populated, so hopping from one neighbour list to the next and reaching each node becomes more difficult. The drop

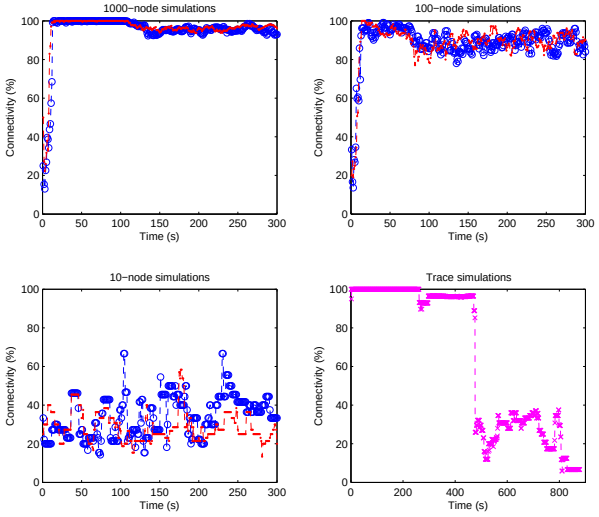

Figure 5.9: Percentage of connectivity throughout SimMud simulations.

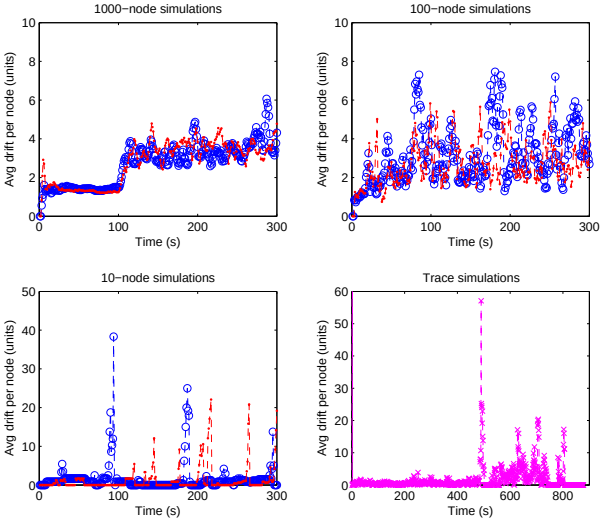

Figure 5.10: Average drift per node in SimMud simulations.

in connectivity that happens during the trace simulation at the 486 second mark corresponds to the rise in missing neighbour nodes and the fluctuation of extra neighbours experienced. The leaving node could have been the connection between two heavily-populated areas, effectively separating the game world into two areas. As more players leave the trace simulation towards the end, the connectivity remains fairly low.

The average drift per node in the SimMud simulation, as seen in Figure 5.10, increases as the density of the nodes decreases. It should be noted that SimMud nodes do not send leave requests to their coordinators if they are message forwarders, and

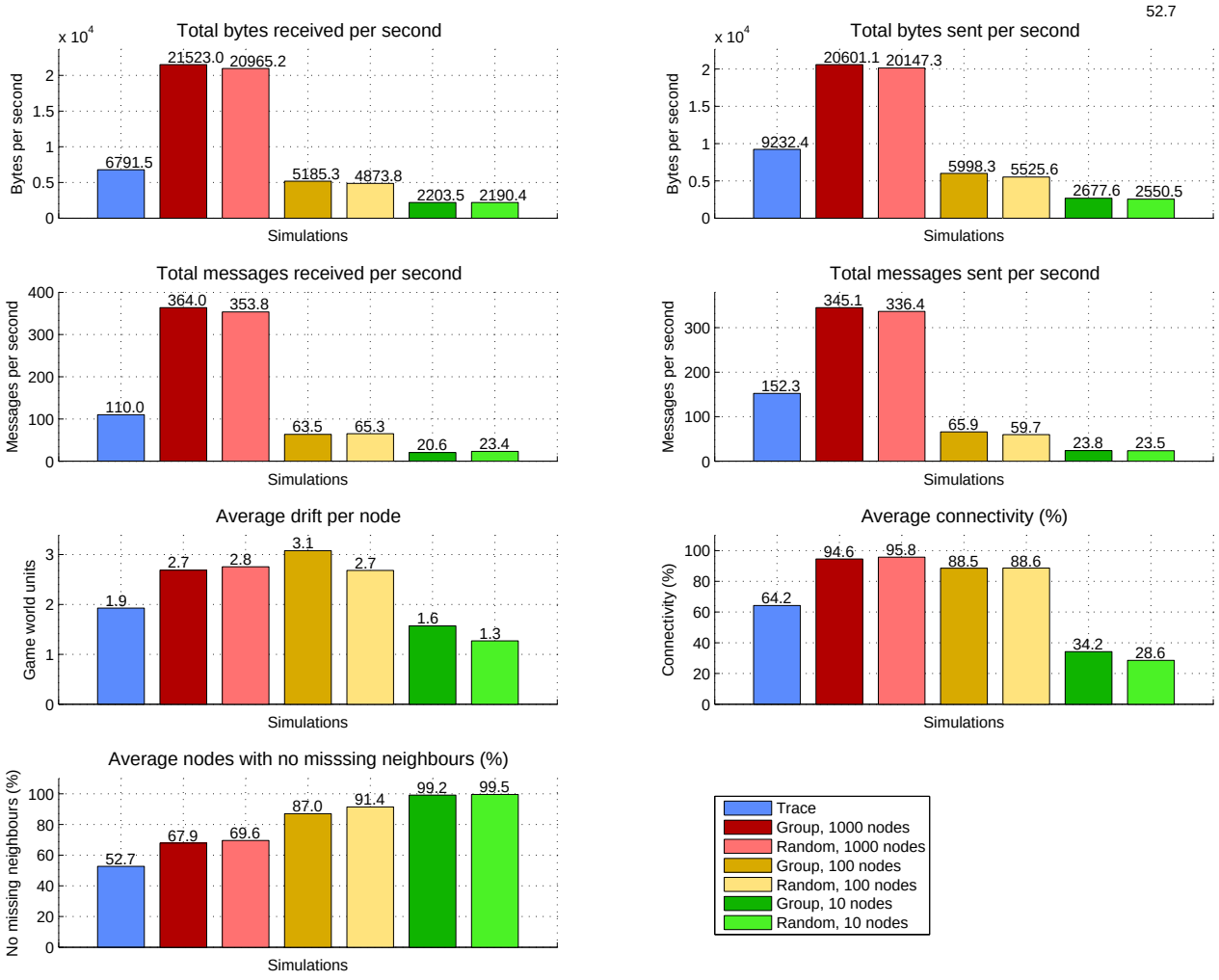

Figure 5.11: Scalar readings from SimMud simulations.

as the population density decreases, a higher percentage of nodes become message forwarders. Therefore, other nodes are not informed of the leaving node, and the updates received from them stagnate. This explains the spikes in drift on the 10-node group movement simulation; a node moves out of the region occupied by a few other nodes, and the neighbours all stop receiving updates from it, leaving it in the list of neighbours but never updating its position. When the node comes back into a region that is subscribed to by its former neighbours, the group-roaming nodes rediscover each other as neighbours and the spike drops down to 0.

As seen in the scalar graphs in Figure 5.11, SimMud has a rather low bandwidth usage; even at the densely-populated 1000-node simulation it only uses between 20 and 22 Kb/s both up and down. The number of messages sent and received in the network is higher than the other practical schemes, since player updates and infor-

mation needs to be routed through the coordinators via the DHT instead of sending it directly to neighbours. The average drift per node stays rather low on average, although it tends to spike in singular areas where players on the multicast forwarding path change regions, as seen in Figure 5.10. The average connectivity is higher in densely-populated simulations, thanks to the high number of superfluous neighbour connections granted by the static world partitioning. The same partitioning makes for a much less connected game world in the lower population simulations. The number of nodes with no missing neighbours for SimMud trends upwards when the population decreases, and although the number of missing neighbours remains low, the average percentage of nodes who are missing neighbours remains high, especially in the trace and 1000-node simulations. This can be attributed to the high amount of messages being sent between nodes and coordinators along the multi-hop path of the DHT, and down the multicast tree, that at the time of the data capture, each node has not been made aware of each possible neighbour.

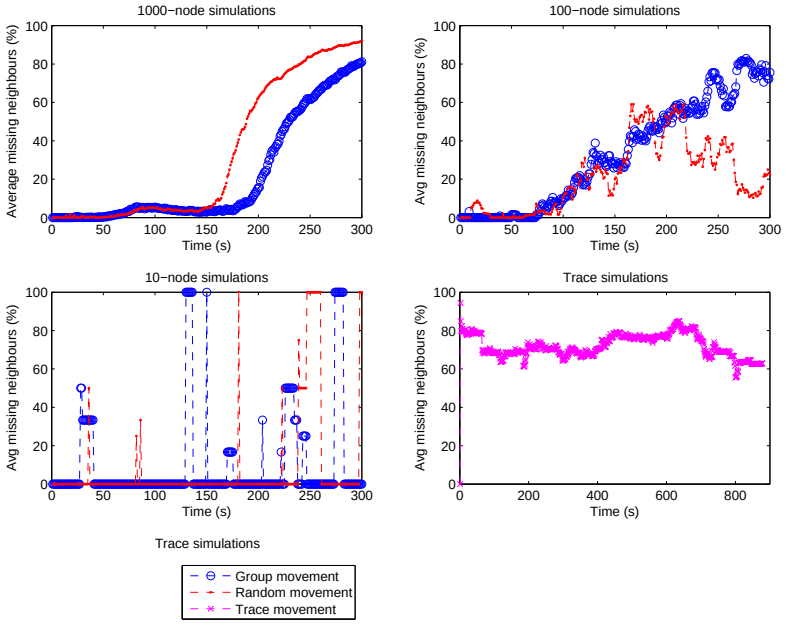

Figure 5.12: Average missing neighbours per node in Vast simulations, as a percentage of the expected neighbours.

5.6.3 Vast performance

Vast is an unstructured, truly peer-to-peer IM scheme. With no underlying routing structure, no hierarchy, and direct communication between neighbouring nodes, Vast strives to use mutual notification to keep all nodes informed of player activity.

The missing neighbour count exhibited in Figure 5.12 indicates that the average Vast node is not properly informed of the nodes surrounding it. This could be as a result of neighbour information not disseminating quickly enough between neighbours, or that the enclosing neighbours that are queried for incoming avatars are unaware of said avatar as well. While the random and group simulations are unaware of neighbours in their AOI, the trace simulations are consistently unaware of between 60% and 80% of their intended neighbours during the majority of the simulation. This runs counter to expectations, but given the delay in message broadcasting introduced by the OverSim environment (to simulate network jitter) and the small window in which the trace generator fires movement messages, it is not guaranteed that the messages would reach the desired neighbour in the correct order. This would result in the neighbour's position being updated to the location stated in the last-received update message, and will result in an incorrect world view. These effects can be mitigated by ordering the messages chronologically, disregarding updates with a time stamp prior to the that of any previously received movement update. It could also be the result of only the necessary enclosing neighbours being

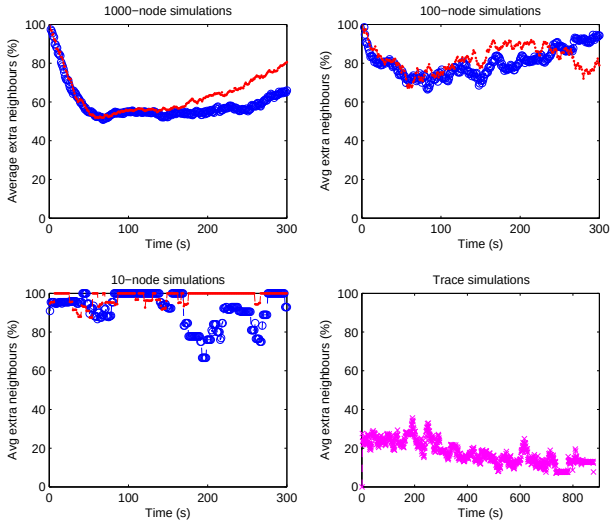

Figure 5.13: Average extra neighbours per node in Vast simulations, as a percentage of the recorded neighbours.

kept on the neighbour list, and no further nodes being included as neighbours. The denser population would result in nodes not needing to be aware of nodes that are still in its AOI, but do not form the enclosing neighbour set, although this is not in line with the Vast design. It is unclear whether the third-party Vast implementation incorporates this feature.

A large portion of the Vast simulations' neighbours residing on each node's neighbour list is extra nodes, as is evidenced by Figure 5.13. The number of neighbours outside a node's AOI as a portion of the total neighbours decreases as the population density decreasess. In some cases of the 10-node simulations, 100% of neighbours on the average node's neighbour list is not inside its AOI. This is to be expected, since Vast strives to keep at least a set of enclosing neighbours in its neighbour list, even if those neighbours are outside its AOI. Mutual notification derives its functionality from the prerequisite that each node should at least know of a few neighbours around it, and in Vast's case, enough to have the surrounding nodes form an enclosed Voronoi cell around it. The trace simulation contains less extra neighbours, as the slightly denser population keeps the nodes from having to reach far outside its AOI to find enclosing neighbours.

The connectivity of the Vast simulation, seen in Figure 5.14, is seen to drop as the non-trace simulations carry on. The lack of proper node leave handling means that nodes hold onto 'dead' neighbours; nodes that have already left, but that have not yet been removed from neighbour lists. This has a cumulative effect as the simulations continue. Nodes keep a slightly fault neighbour list, passing move updates to nodes that are no longer there, and possibly not accepting new nodes

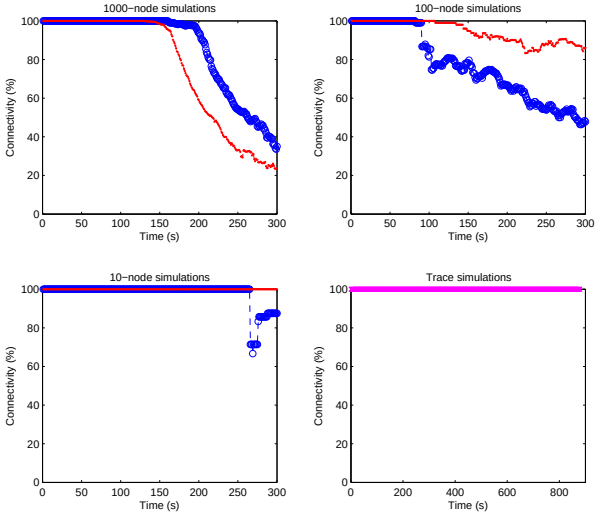

Figure 5.14: Percentage of connectivity throughout Vast simulations.

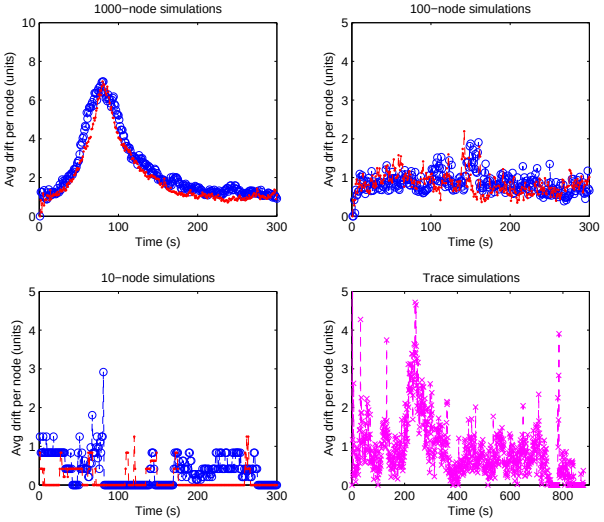

Figure 5.15: Average drift per node in Vast simulations.

because of the 'dead' neighbour lingering in its world view. As a result of this, the connectivity algorithm cannot hop along neighbour lists to reach each node in the simulation, and the overall connectivity of the game world is shown to have suffered. With less frequent neighbour leave actions, the more time each node has to build a proper game view. The trace simulation maintains 100% connectivity throughout, because of the well-informed nodes, as described previously. This can also be attributed to the small number of players present, and the small game world.

Drift in the Vast simulations (Figure 5.15) remains relatively low when the network experience less joining and leaving nodes. The 1000-node graph shows that the

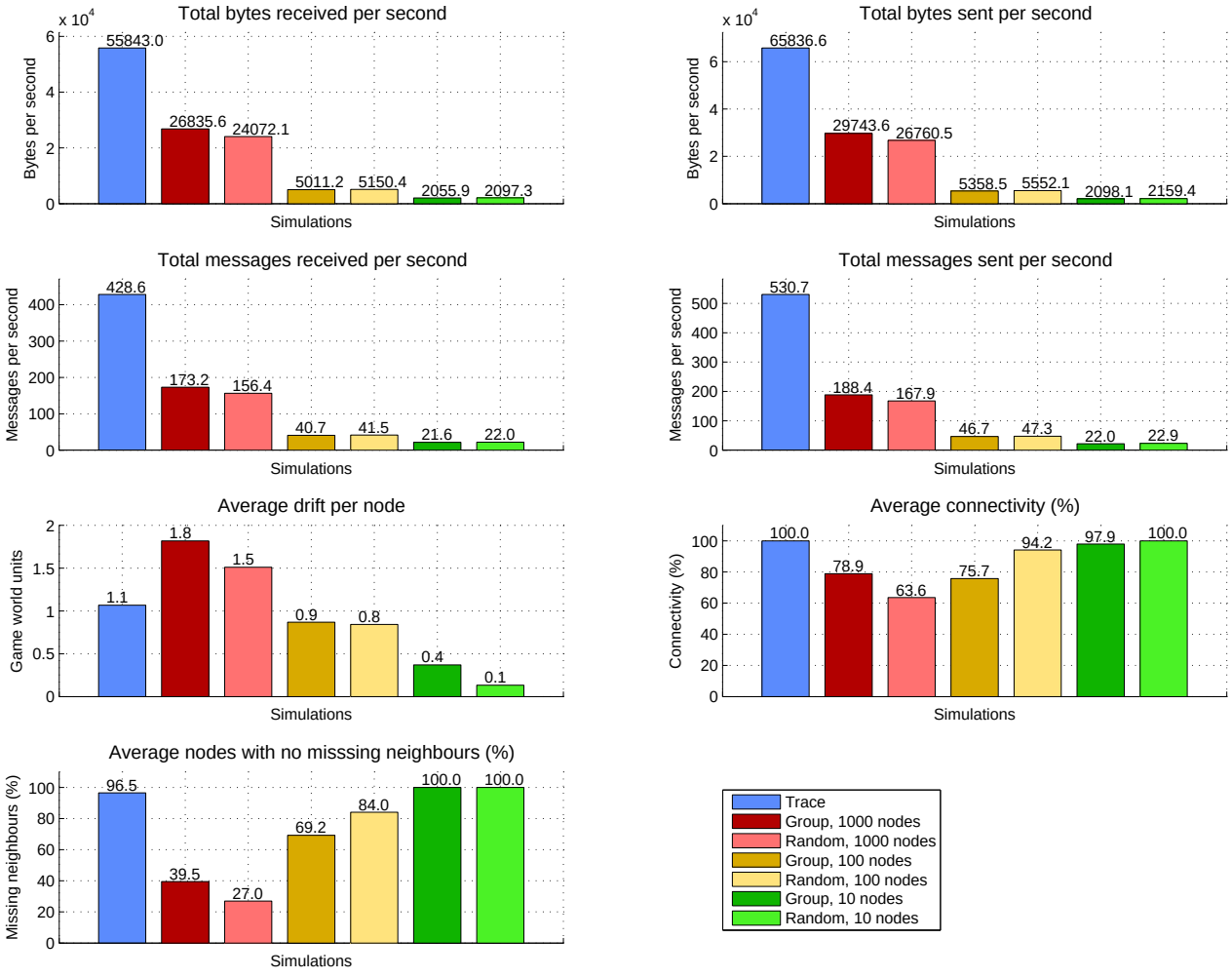

Figure 5.16: Scalar readings from Vast simulations.

average drift remains high over the period that most nodes are still joining. After the joining process is complete and nodes get into more direct communication (as opposed to relying on neighbour lists to get the first positions of a joining node's neighbours), the drift becomes less over time. The trace simulation has slightly more erratic drift counts per node. As stated, since six movement updates are fired in rapid succession, it is not always guaranteed that the latest movement update that was sent will be the last one to be received. This would explain the fairly frequent shift in average drift.

From Figure 5.16, we can see that Vast has a low average node drift, especially at lower populations. The trace simulations consumed more bandwidth than most of the other simulations, mostly due to its frequent updates. This is also reflected in the number of messages sent. The 1000-node simulation uses between 25 and 30

kB/s bandwidth, both up and down, and the less-populated simulations all clock in at under 6 kB/s in both directions. Connectivity is maintained well in the real-world trace simulations, as well as in the simulations with low populations. It does, however, suffer in the larger simulations, partly due to the aforementioned absence of a proper mechanism for handling nodes unexpectedly leaving. The same can be said for the missing neighbours.

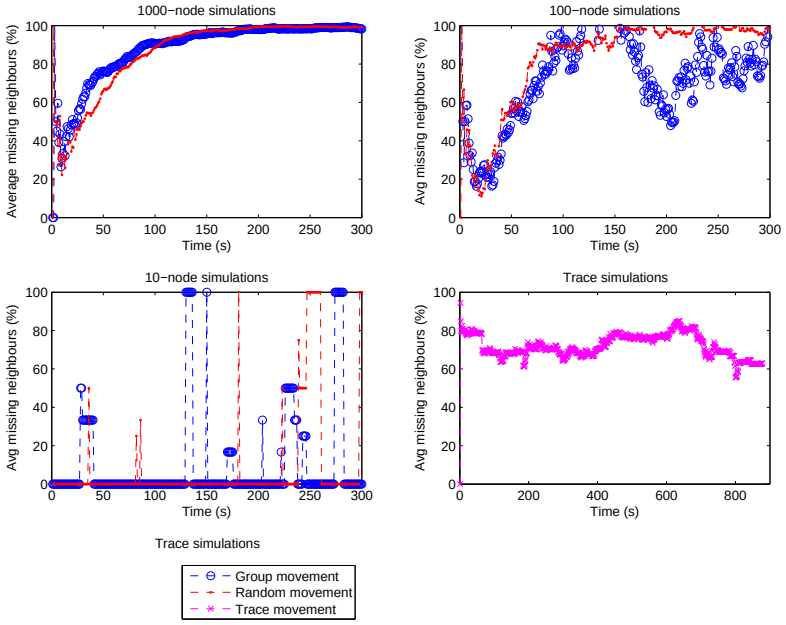

Figure 5.17: Average missing neighbours per node in MOPAR simulations, as a percentage of the expected neighbours.

5.6.4 MOPAR performance

MOPAR uses a mixture of structured and unstructured communication methods to create a hybrid IM scheme. The DHT helps to keep master and home nodes in communication with one another and to make slave nodes aware of their neighbours, while slave nodes communicate directly with their neighbours. While every effort was made to replicate the MOPAR scheme in code, it must be said that the reproduction was done without input from the original designers, and could therefore be a suboptimal implementation of the scheme.

The MOPAR scheme simulations (Figure 5.17) display a very high number of missing neighbours. Even the trace simulations perform poorly in this metric. This is most likely due to an implementation fault causing nodes to direct their master node registration information to nodes that do not stay home nodes for long. With the continuous addition of new nodes, home nodes are constantly being reassigned, and slave nodes trying to find the appropriate master nodes may end up routing a message to an uninformed home node, leaving multiple master nodes with different views of the same cell. This is a fault in the implementation that could not be addressed due to time constraints.

The low number of extra neighbours, shown in Figure 5.18, is heartening, but in conjunction with the high number of expected nodes that are missing from each neighbour list, it illustrates that each node is not only under-informed, but also

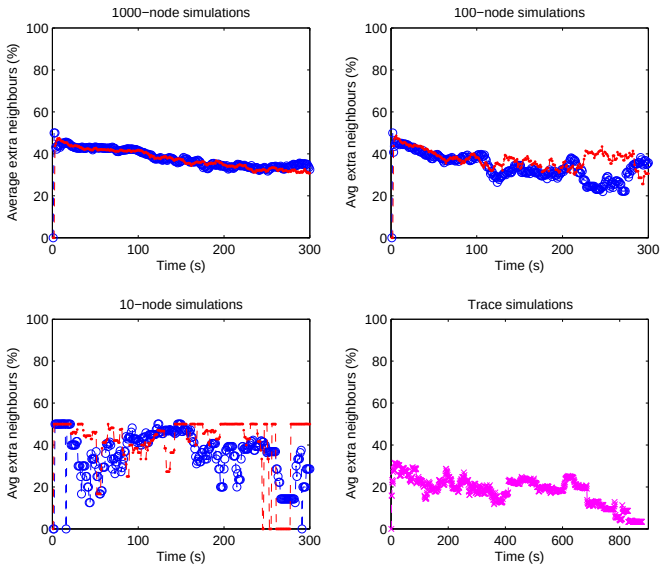

Figure 5.18: Average extra neighbours per node in MOPAR simulations, as a percentage of the recorded neighbours.

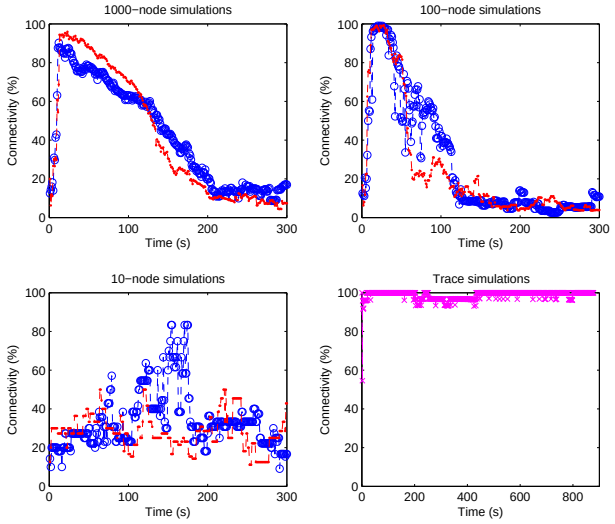

Figure 5.19: Percentage of connectivity throughout MOPAR simulations.

frequently misinformed.

The low connectivity index, as shown in Figure 5.19 is difficult to explain, especially for the high-population simulations. The high frequency of nodes constantly entering and exiting cells and switching master nodes, coupled with the longer turnaround time of routing messages to master nodes via the DHT could result in nodes not finding their neighbours in a timely fashion. The high connectivity experienced in the trace simulation can be attributed to the low amount of overall movement made by players in the game world; the area surveyed was rather densely populated

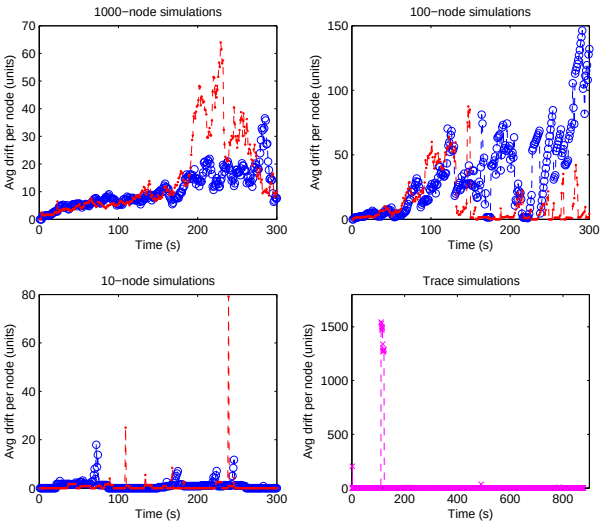

Figure 5.20: Average drift per node in MOPAR simulations.

for its size, but it was overall a small area. This could result in a low number of nodes ever moving into new cells, and only a handful of active master nodes in the game world.

Figure 5.20 shows the average drift in MOPAR to be very high, most likely due to the uninformed master nodes feeding the slave nodes false information. Especially at larger densities, the average drift per node is high, indicating a very incorrect view of the world for each node. This issue stems from the fault stated above.

One thing that MOPAR does perform admirably in is the low bandwidth usage, even in high populations, as seen in Figure 5.21. It should be noted that this is the average of the total messages and bytes sent; master nodes will experience more load than other nodes in the simulation, and in a more advanced implementation, it would be advisable to have master nodes change according to the specifications of each node's hardware or bandwidth connection. MOPAR maintains a bandwidth use of less than 30 kB/s up and down, and the number of messages sent and received per node only crosses the 200 messages per second mark in the 1000-node simulations. The number of messages sent in the less dense simulations does not decrease by the same ratio as, say, Vast, but it is a substantial decrease nonetheless. The correctness of the nodes' world view suffered as a result of the suboptimal implementation of the scheme.

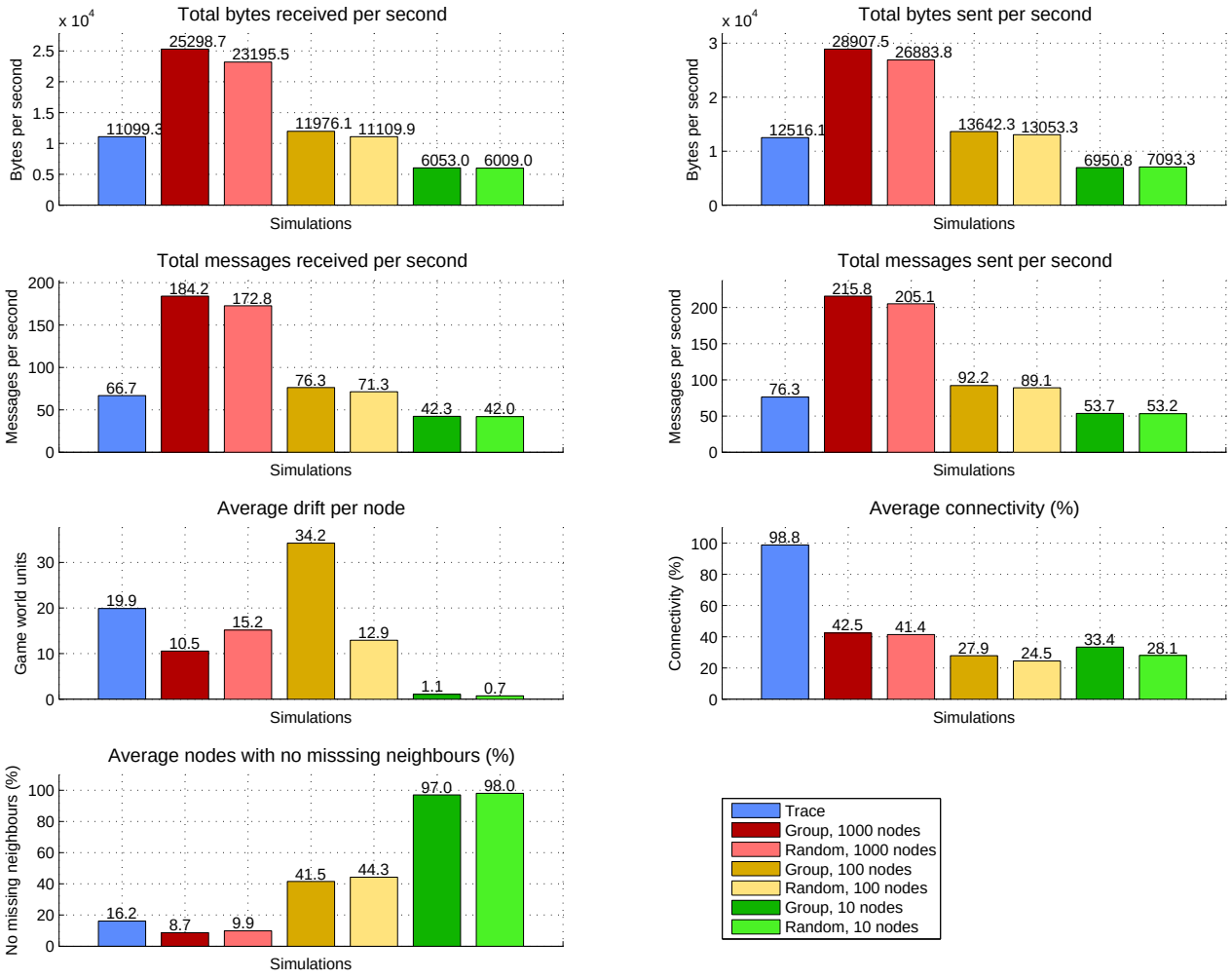

Figure 5.21: Scalar readings from MOPAR simulations.

5.7 Comparison of simulation results

In this section, the results of the previous section are discussed, and the overall performance of each of the schemes is ranked. Most of the information will be taken from the respective scalar graphs (Figures 5.11, 5.16, 5.21 and 5.6).

5.7.1 Neighbour correctness

The correctness of a node's view of the neighbours, and by extension other objects, in the game world is dependent on the correctness of the neighbours' positions relative to the node, i.e. the lack of player drift, and the number of neighbours missing in the node's neighbour list. Below, the average measurements of these two metrics are compared on a per-scheme and per-simulation basis. Note that no FullConn measurements are illustrated in the 1000-node graphs in Figures 5.22 to 5.28, since there were no 1000-node FullConn simulations performed.

It will come as no surprise that, when comparing the drift visible in the simulations (as shown in Figure 5.22) that the FullConn scheme performs without fault, indicating zero player drift over all nodes throughout the duration of the simulations. The suboptimal implementation of the MOPAR scheme severely impacts the amount of drift each node sees in the game world. SimMud and Vast perform roughly the same at the 1000-node density, but Vast takes a very clear lead when the population drops. The trace simulations also favour the Vast scheme in terms of average drift count.

The average percentage of nodes with no missing neighbours is depicted in Figure 5.23. Again, FullConn behaves ideally, and again, MOPAR suffers in this metric, but it becomes less evident when the simulation is less populated. Vast manages to keep track of each node's neighbours near perfectly at lower densities, but struggles at higher densities. The Vast simulation has a 13% increased neighbour retention in the 1000-node group simulation compared to the random movement simulation of the same population. This is to be expected, as nodes that are surrounded by neighbours moving in the same direction leads to less neighbours being swapped or removed from the neighbour list, keeping a fairly constant list of neighbours. The trace simulations experienced little missing nodes under the Vast scheme, again partly because of the low population and the frequent movement updates. SimMud serves just over two-thirds of its 1000-node population with the right neighbours, although this number is still sub-par to what is expected in an actual MMOG. At lower populations, SimMud performs much better, keeping over 87% of nodes informed of their correct neighbours. This could be attributed to the lowered strain on the coordinator nodes and the reduced rate of nodes entering and leaving the

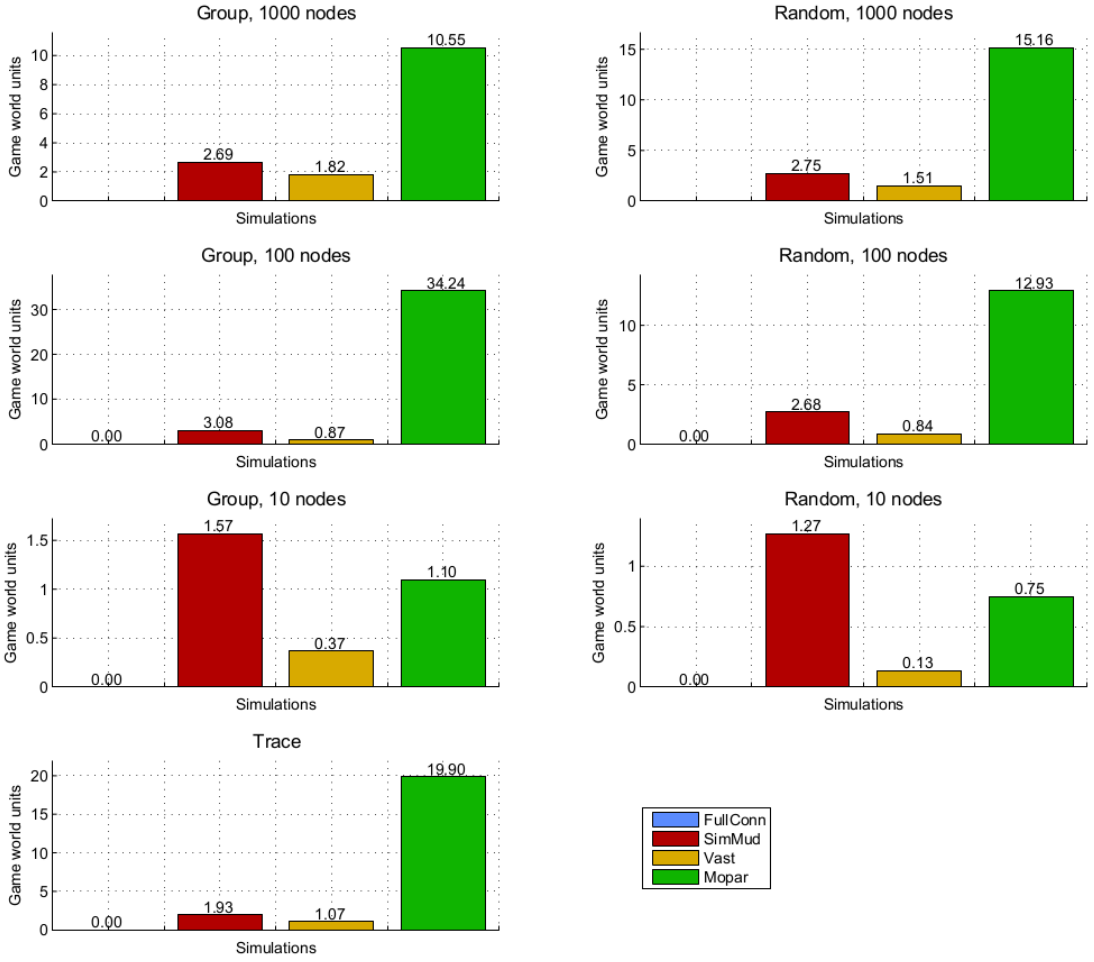

Figure 5.22: The average drift of each interest management scheme per simulation type

Pastry overlay, requiring less coordinator node information hand-off as new nodes enter and exit the Pastry key space ring. The SimMud scheme does not handle the trace simulations well, most likely for similar reasons; constantly connecting nodes taking up new spaces in the Pastry key space, resulting in coordinator hand-off overhead, and the crowded regions causing large amounts of network traffic and computation for the coordinator nodes.

5.7.2 Game world connectedness

The degree to which each node is connected in the game world is a good indication of how well the entirety of the game's participants are aware of the players around them. Below the connectivity index of each simulation is discussed and compared.

Figure 5.24 shows the average percentage of nodes connected to one another through the neighbour-hopping mechanism described in section 5.1. Again, MOPAR

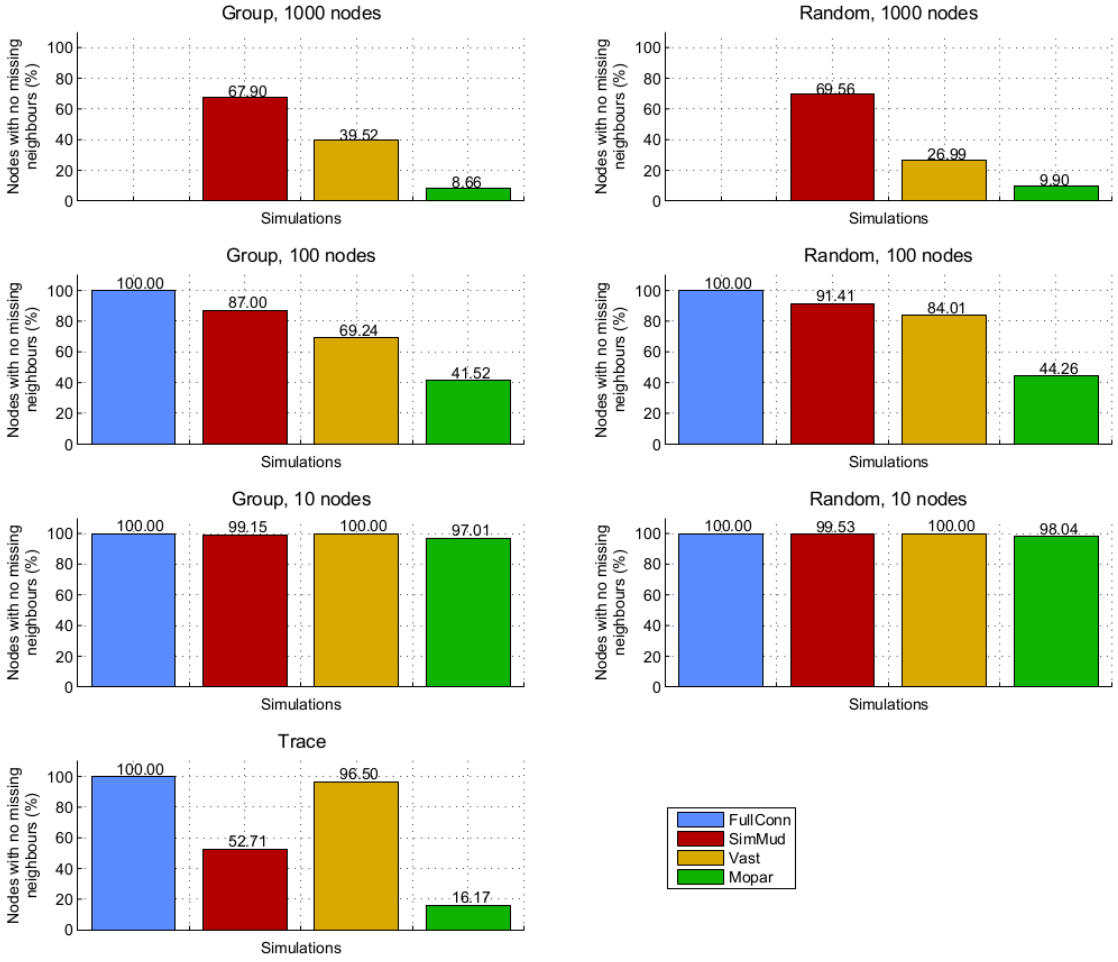

Figure 5.23: The average percentage of nodes with no missing neighbours for each interest management scheme per simulation type

has weak performance in this aspect, except for the trace simulations. This result is possibly skewed by the small size of the occupied game world, since a few master nodes keep track of most of nodes in the game world, making the recursive hopping of a master node's neighbour list encompass most of the nodes in the game world. FullConn performs ideally, as expected. SimMud keeps a high level of connectivity overall in the large and medium populations, but the connectivity falls short in the low population simulation. This is due to the way the nodes communicate with their coordinators; since a node routes a message through the Pastry overlay to communicate with a coordinator as opposed to keeping it in its neighbour list to communicate directly. Only when nodes are near one another in the game world and subscribe to multiple regions will coordinators have a high level of connectivity in the game world. The trace simulation appears to have a lower connectivity because of the

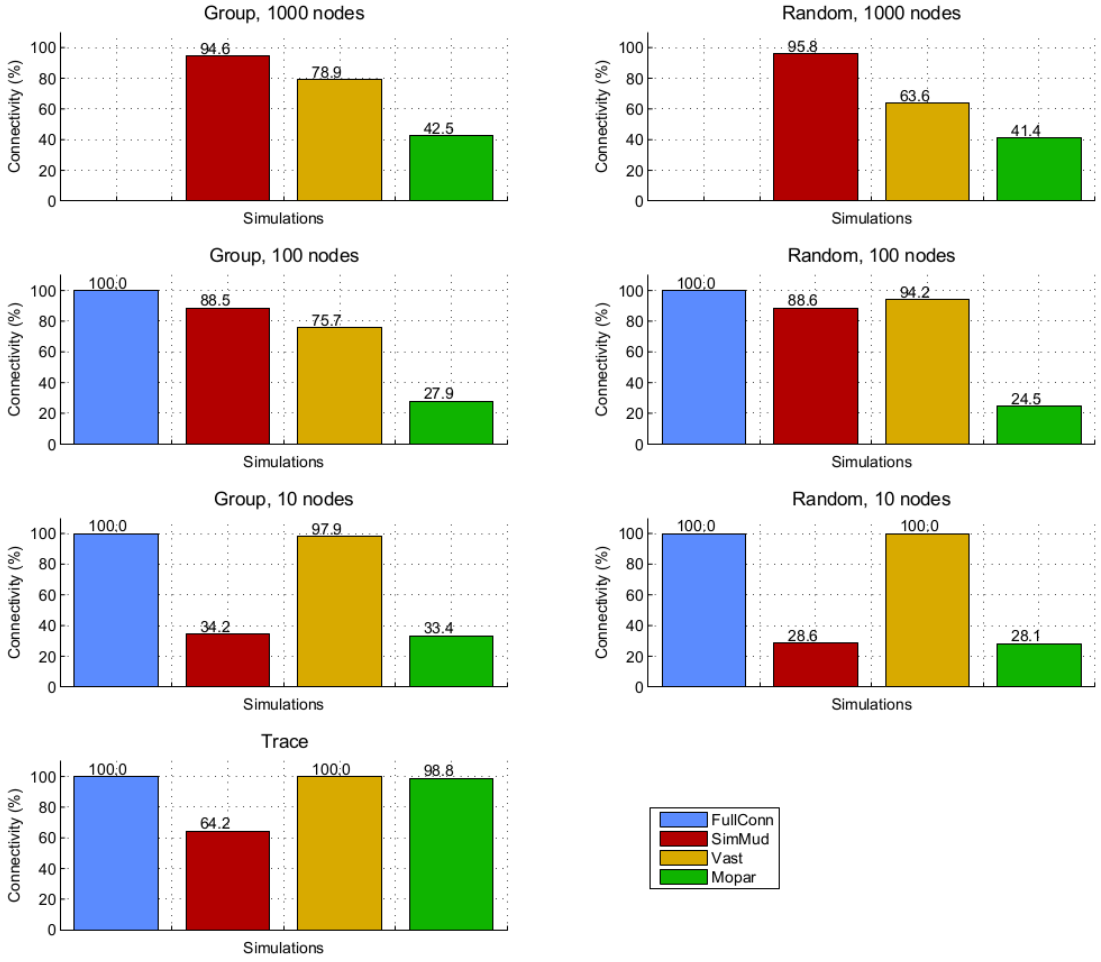

Figure 5.24: The average connectedness of each interest management scheme per simulation type

sparser player count. Vast displays improved game world connectedness in the less-populated simulations. The way that Vast is structured should, however, provide a near-100% connectivity throughout all simulations. The connectivity does, however, start to diminish after nodes start to leave the overlay unexpectedly, compounding the misinformation as the simulation continues. Vast's connectivity in the trace simulations measures 100%, helped once again by the more frequent updates and slightly denser population.

5.7.3 Message and bit rate

A big factor in the performance of any interest management scheme is the overall bandwidth the mechanism uses. If the bandwidth use is too high, players will suffer lag when wandering the game world, and players who have limited data will quickly

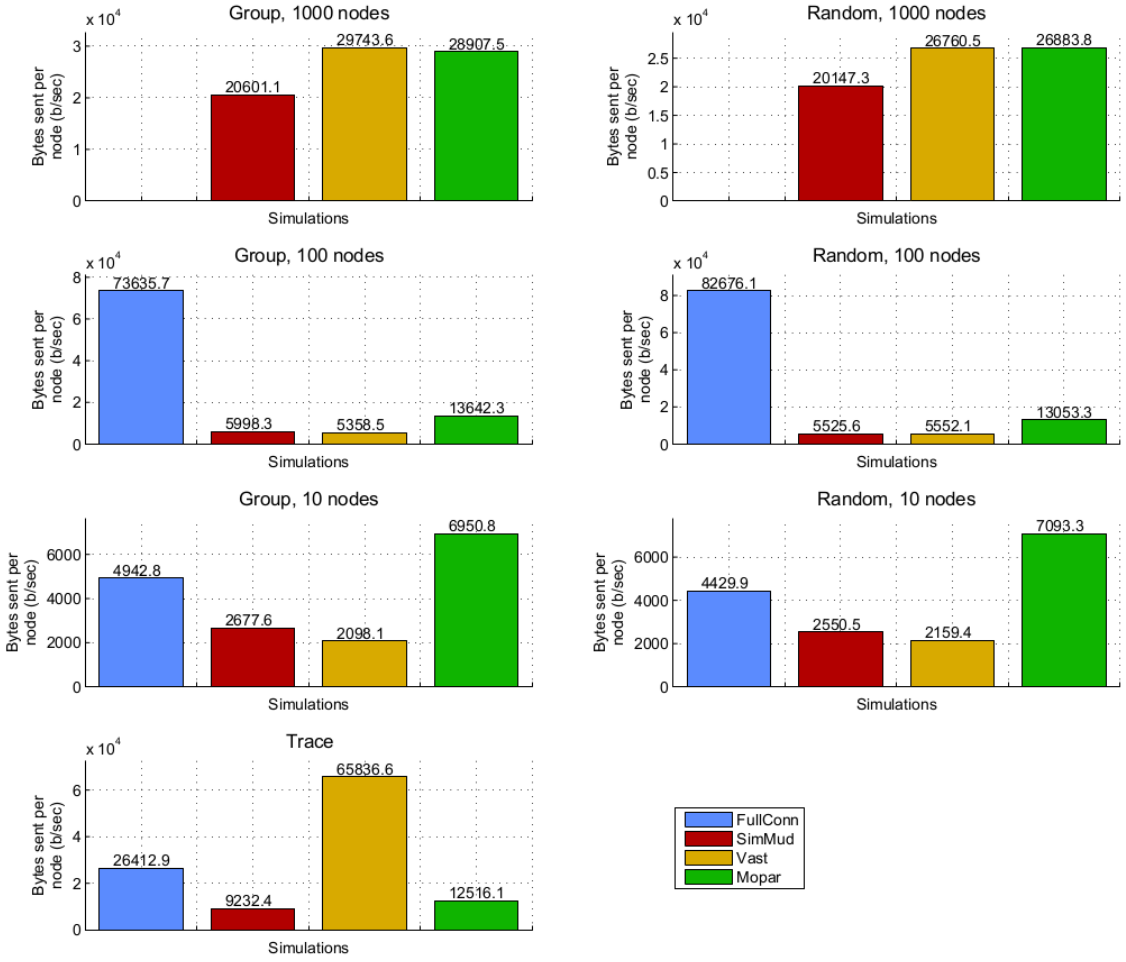

Figure 5.25: The average bytes per second sent per node for each interest management scheme, grouped by simulation type

burn through their allotted data while playing the game. An MMOG should aim to lower their bandwidth consumption by limiting the number of messages sent. In part, this is why interest management schemes exist; to keep the player from receiving and transmitting too much information too frequently to parts of the game world that offer no possible interaction. The IM schemes that connects nodes with too many neighbours outside their AOI will also struggle to keep their bandwidth low.

The bit rate of each IM scheme per simulation, as measured in bytes per second, is captured in Figures 5.25 and 5.26. Here the major flaw in the FullConn design is evident; the bit rate of the 100-node simulations are more comparable to the bit rates of all the other simulations' 1000-node simulations. It is clear that this approach does not scale effectively. For smaller simulations, however, the bit rate is more comparable to those of the other IM schemes. This is to be expected, however,

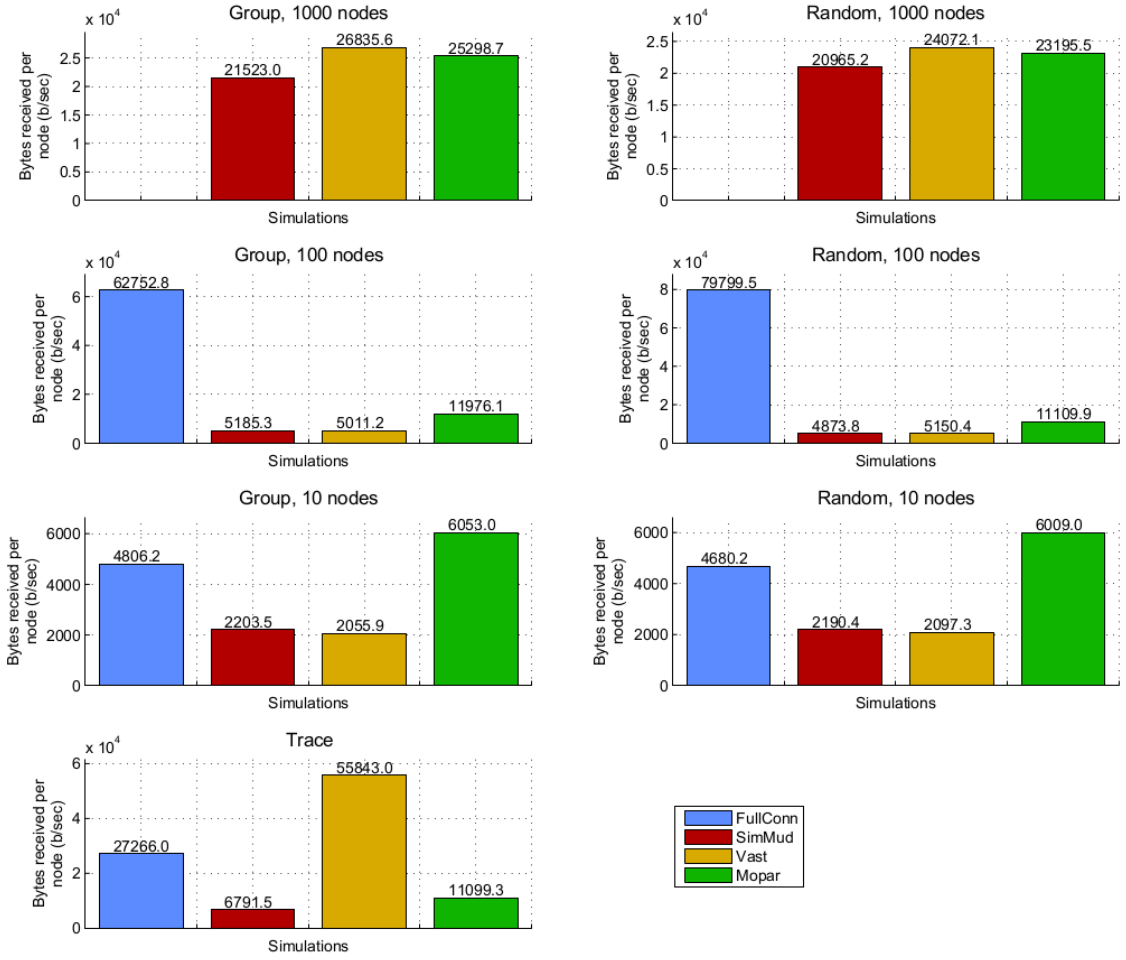

Figure 5.26: The average bytes per second received per node for each interest management scheme, grouped by simulation type

and it still uses roughly twice as much bandwidth as SimMud and Vast in the 10-node simulations. SimMud uses the least amount of bandwidth per node of all the IM schemes for the 1000-node simulations. Because the coordinators only receive updates from nodes when they leave the region or change their movement direction, the large amount of nodes have less information to communicate to the coordinators. Neighbour lists are not exchanged, so large amounts of information are not sent in one message. In the 1000-node simulations, MOPAR becomes comparable, but this could once again be attributed to the poor connectivity and high number of missing nodes displayed by the implementation. Vast consumes more bandwidth than SimMud at the 1000-node simulations, which is to be expected, since a more populated environment leads to a lot more messages sent and received to keep nodes updated. As the population decreases, the bandwidth use grows smaller, leading

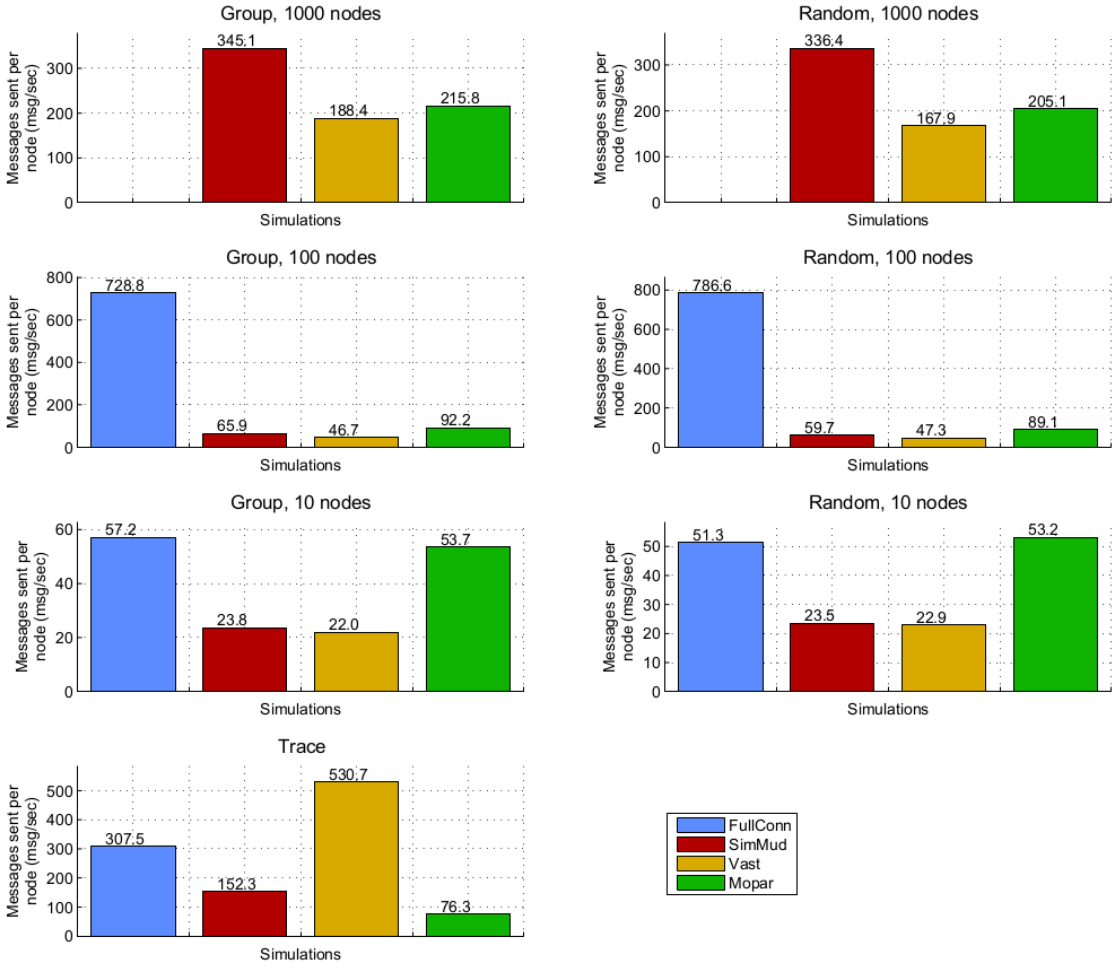

Figure 5.27: The average messages per second sent per node for each interest management scheme, grouped by simulation type

to Vast becoming the least expensive IM scheme in terms of bandwidth. During the trace simulation, however, the bandwidth use is high, caused by the frequent movement updates broadcast to the Voronoi neighbours. Because the Vast trace simulation contains more of its expected neighbours, i.e. the neighbours in its AOI, Vast broadcasts and receives more messages to these neighbours.

As seen in Figures 5.27 and 5.28, SimMud sends a lot of messages per node when compared to the other IM schemes. This is as a result of the multi-hop route messages need to take to be routed to their destination in the Pastry overlay. Since nodes in a region only update their coordinator, getting a message to the coordinator via the Pastry overlay routing constitutes $O(\log n)$ message hops, where n is the total nodes in the Pastry key space. This is why, despite the lower bandwidth usage, the number of messages sent by SimMud is higher than that of Vast. Vast forwards

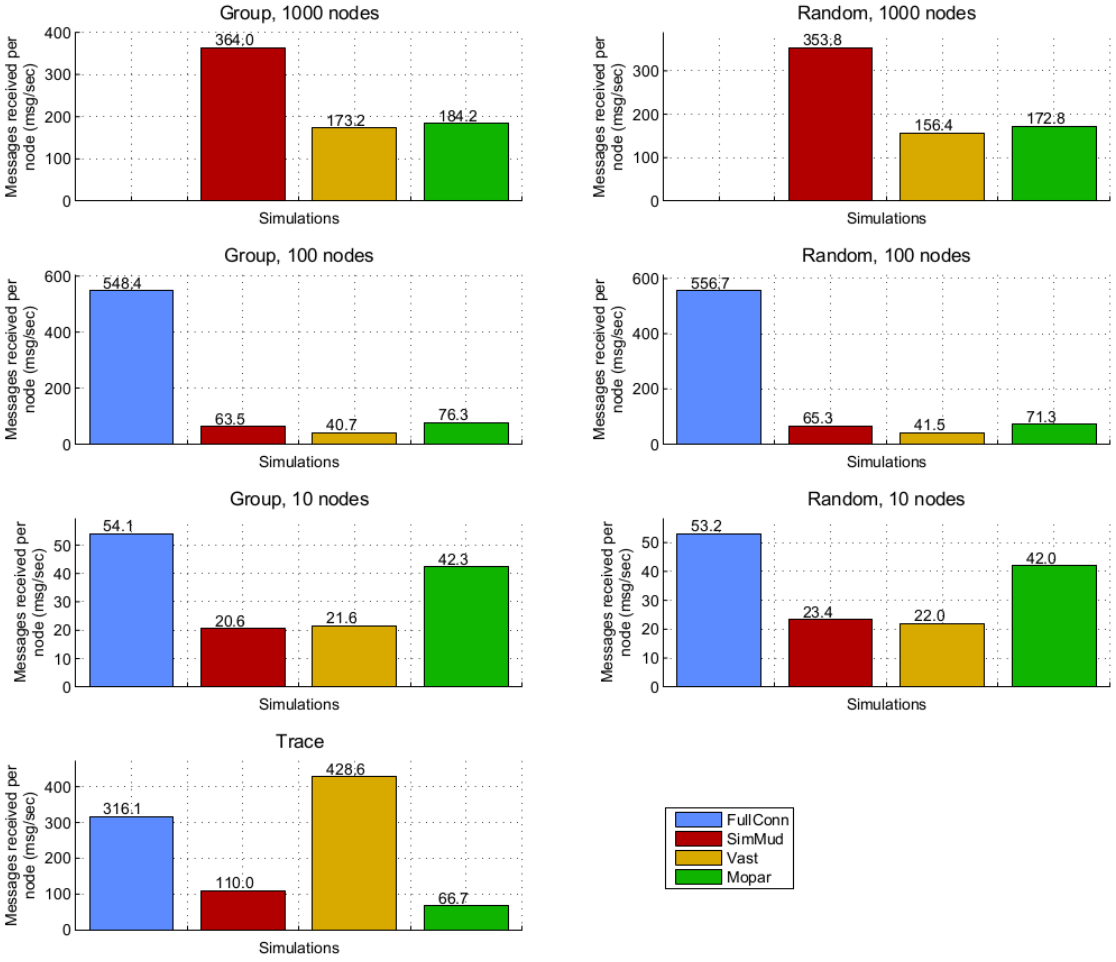

Figure 5.28: The average messages per second received per node for each interest management scheme, grouped by simulation type

less messages, since the messages are sent directly to a node's neighbours, but it will still send movement updates to all interested neighbours. Joining lists for new nodes are aggregated into a single message, helping to minimize the number of messages sent. Vast sends far more messages than SimMud in the trace simulations, again due to the high update rate. Vast nodes have no prediction algorithm in place, so updates are sent regardless of direction change or lack thereof. The dead reckoning algorithm helps SimMud to have a lower message rate. FullConn again displays how inefficient it is, with its 100-node simulations sending more messages each second by an order of magnitude when compared to Vast or SimMud. MOPAR sends less messages at higher densities, while sending more at lower densities. The use of mutual notification to keep neighbours up to date would explain the fewer messages at the higher populations, and the more frequent use of messages routed through

the Pastry overlay (as used with master node registration and subscription) would explain the increases message sending in the lower populations.

5.8 Summary

This chapter covered the metrics used to compare the interest management schemes, as well as the expected results based on the information gathered from the literature. Two comparative simulations were created to test how closely the OverSim environment could match the literature results, and these were compared and found to be within expected deviations for the most part. The different parameters used to compare the four selected interest management schemes were outlined, and their results were discussed and compared. Vast was found to be very useful for the trace simulations, and SimMud performed better in the simulated movement experiments. FullConn does not scale well, and the MOPAR implementation was not ideal, failing in most categories.

Chapter 6

Conclusion

This chapter discusses the results attained and illustrated in chapter 5 and recommendations for future research in the topic of peer-to-peer interest management and the surrounding topics of a peer-to-peer NVE in general.

6.1 Conclusions drawn from simulation results

The research objective of this thesis was to compare a subset of the P2P MMVE interest management schemes available in the research literature. Three IM schemes (Vast, SimMud and MOPAR) as well as one naive implementation (FullConn) were set up with identical parameters and compared. By making use of both real-world movement patterns and simulated movements, we were able to determine the effectiveness of each IM scheme based on their bandwidth consumption, connectedness and state correctness.

Firstly, it is evident that the MOPAR scheme, as implemented for the purposes of this thesis, is flawed. It performs unsatisfactorily in most of the measured criteria. This is not to say that MOPAR itself is flawed. The design of the implementation of MOPAR for the OverSim environment was done with only the single document [69] as source. This should, however, not reflect negatively on MOPAR as a design; it is merely this implementation that is flawed. If any future tests are to be conducted using this interest management scheme, improvements will have to be made.

FullConn, being the naive implementation, remains too impractical an approach to use in a real-world MMOG. The exponential scaling results in huge bandwidth consumption and number of messages sent and received per node, and will soon flood a network, causing incredible latency for all players involved. At low populations, this is a good scheme to implement. To some extent, it's what helps Vast (and in theory, MOPAR) to overcome the limitations set by a structured DHT routing scheme. The direct communication with a small number of nodes relieves some of

the overhead on any kind of administrative node, if any are present, while keeping a node's neighbours informed of its whereabouts.

SimMud proved to be a reliable model for interest management. The Pastry overlay and DHT routing is useful for maintaining connectivity over a wide, dynamic network. That's why MOPAR makes use of Pastry to serve as its underlying architecture to identify and contact home nodes. The large number of message hops, however, incur a latency cost, since the message is routed through multiple nodes on its way from the node to the region coordinator. This also explains the high number of messages sent by SimMud; to route a message is to resend the same message along the path. Although these messages contain less information, since SimMud is a relatively simple implementation, more messages are sent overall; more so than any scheme when the game world is particularly densely populated. The gain in bandwidth caused by the infrequent contact with the region coordinators coupled with the small message load is offset by the time it takes to route the messages. There is no major discernible difference between the group-based or random movement schemes for SimMud, since the neighbours have minimal communication with one another directly. The message traffic is still handled by the coordinators. SimMud's performance in the trace simulations, although bandwidth-efficient, suffers when it comes to neighbour correctness and connectivity.

By contrast, Vast performs well in the trace simulations. The frequent updates help the average drift in the network to remain low, while the smaller active game area keeps the player connectivity high and each node is informed of the vast majority of its neighbours. This comes at a significantly higher bit and message rate, since a node's movement updates are broadcast without any regard for prediction of movement by its neighbours. It is possible to add prediction mechanisms, e.g. dead reckoning, to help limit the number of updates in the Vast simulation; however, since Vast relies on mutual notification to maintain state consistency, limiting the number of updates sent to neighbours could negatively affect the game state consistency. In contrast to the trace simulations, Vast does not perform as well in the simulations with the random and grouped movements. The significant drift in the 1000-node simulations, the drop in connectivity in the 100-node simulations and the average of between 17 and 32% of nodes missing neighbours in both aforementioned densities indicate that Vast on its own is not sufficient to keep an NVE up to date. When looking at the correctness metrics, Vast shows a slight improvement in the grouped movement simulations over the random movement ones. The constant changing of nodes incurs overhead for Vast nodes, and that overhead could lead to player drift and possible false neighbours being introduced to the node. A longer-lasting connection between a node and its neighbours is bound to be less prone to errors.

When comparing the trace simulations against the random and group movement ones, the value of the trace simulations should carry slightly more weight than the rest. Since these traces were collected from a real-world, existing MMOG, it is a much closer approximation of player behaviour than the pseudo-randomly generated movements created by the other simulations. It also shows how a commercially successful MMOG such as World of Warcraft, times their player updates, which could be useful in further simulations.

Among the interest management schemes compared in this thesis, Vast is a more likely candidate, given the results gained from the trace simulations. How Vast scales to a larger sample set is left for future work. Despite the poor performance of the MOPAR implementation this paper, hybrid schemes are worth exploring further. Theoretically, their use of both structured and unstructured message routing mechanics will help mitigate the drawbacks of the other, and it can even be expanded to be used in the cloud, as is evidenced in [51].

6.2 Related work

During the literature review, it was noted that none of the proposed architectures provide an empirical comparison between the different IM schemes, with the exceptions of Krause [38] and Boulanger [13], although the latter compares IM mechanics in MMVEs using a client/server architecture. This thesis aims to build on the work done by Krause by comparing the structured and unstructured P2P IM schemes with one another as well as introduce the hybrid P2P IM scheme to the comparison.

6.3 Future work

The purpose of this study was to compare a subset of the available P2P IM schemes to determine their performance under different loads and their contribution to the network traffic. There are multiple other interest management schemes still under consideration as discussed below, each growing and changing to suit the needs of the application it is being used in. Although interest management is an important part of a networked virtual environment, it is one of many obstacles that need to be overcome before a commercially viable, secure and compelling peer-to-peer MMOG can be produced.

6.3.1 Improvement of interest management schemes

The IM schemes used in this thesis are rather dated models, and as of writing, improvements have been made to some of the IM schemes used. For example,

using AOI-casting methods like Vorocast [35] could improve the scalability of the Vast IM scheme. Implementing these improvements into the OverSim simulation environment would benefit testing in later research.

As stated before, the MOPAR implementation in this thesis has proven to perform in a suboptimal manner, skewing many of the results and forcing MOPAR out of favour as a viable IM scheme. That is not to say that MOPAR, as a hybrid IM scheme, would not be useful in mitigating the limitations of both the unstructured P2P mechanism and the structured DHT-routing used in Pastry. It would be beneficial to fix or redesign the OverSim implementation of MOPAR to reflect the intended functionality more closely, as described in the paper [69].

6.3.2 Comparison of other interest management schemes

In order to determine whether the interest management schemes discussed and compared in this thesis are most suited to host peer-to-peer MMOGs, other interest management schemes and overall P2P architectures should be tested by the same standards. Other schemes, such as Solipsis [24], Badumna [40] and Quon [5], can still be tested.

Other interest management schemes can be divided into different groups based on their characteristics. Often, however, interest management approaches are mixes of different techniques and earlier approaches. For example, VoroGame [14] is a hybrid interest management approach, which is adapted from Vast's use of a Voronoi diagram to disseminate player movement and determine the range of a player's AOI, and the structure of a DHT to help store and retrieve object data. Still other IM schemes have started to implement cloud-based approaches, as discussed in section 6.3.4.

6.3.3 Generate more diverse data sets

By varying more parameters in the simulations, for example movement speed, game world size and a more varied set of populations, researchers could get a better understanding of the workings of the interest management schemes. Faster-moving nodes could incur more overhead in IM schemes that have a static game world partitioning, since moving between regions is usually an expensive process, both computationally and with regards to communication. This could be an important test to perform, and could help shed light on how the IM schemes handle varied movement speed, such as mounted travel in-game.

6.3.4 Real-world implementation of testing methods

The interest management schemes proposed in this study have not been implemented in any real-world NVEs outside of simple test case applications [37], [69]. The user experience of an NVE is not entirely objective, since players are willing to tolerate different latencies in different types of games [4]. Therefore, the true test of the usefulness of peer-to-peer as a method of hosting NVEs is to produce a stand-alone application and allow users to connect to the network and participate in game-like activity. While simulations are a good testing ground for new NVE approaches, the true test of their effectiveness lies in real-world implementations.

One possibility would be to implement a real-world application of each of the interest management schemes and have one instance of the application running in the cloud, such as an Amazon Elastic Compute Cloud (EC2) instance [2]. This will allow for multiple concurrent instances running simultaneously with identical hardware specifications.

Instead of using the cloud to host single instances of a node acting as a player, however, the cloud can be integrated into the design of the MMOG itself. In Najaran and Krasic's article [51], they have implemented a distributed first-person shooter game on Amazon's EC2 cloud platform. The interest management mechanism draws inspiration from the Donnybrook implementation [8], breaking an avatar's interest up into coarse and fine-grained interest sets based on criteria of proximity, visibility and recency. The message routing scheme takes the form of a modified Scribe implementation creating multicast topics for each player in the network, publishing and subscribing to the fine-grained interest set. Using a cloud-based platform in lieu of the traditional client/server architecture has all the benefits of a distributed P2P architecture, i.e. no single point of failure, very high scalability, lower hardware provisioning costs and responsive provisioning for peak concurrent user traffic. Cloud-based platforms have the added bonus of game logic processing units being in much closer physical proximity to one another, with high-speed connections between them. The message travel time between game logic processing nodes in a single data centre is much shorter than that of nodes located in different countries. For example, if an NVE is created with similar workings to SimMud, outlined in section 3.5.1, with the coordinator nodes residing in the cloud rather than on individual players' host devices, coordinator nodes can exchange information on nodes moving between regions in a much shorter time. Cloud-based approaches to MMOGs hold much promise and it would be wise to pursue more research on the subject.

Another option would be to distribute the real-world application to a community of users who each have their own computer to run the application on. This will show how well the applications perform under heterogeneous machine specifications

and capabilities as well over a real internet connection. An example of this implementation is the Koekepan adaptation to Minecraft [20], which uses an existing popular game to test different implement, test and compare IM schemes as well as other facets of distributed NVEs. Using an existing game has the benefit of having a pre-existing player base, and Minecraft's encouraging of modification and active community allows for a large pool of players to help in testing.

6.3.5 Implementation in commercially available MMOG

The ultimate test of the P2P MMOG architecture would, of course, be a commercially available game that could attract an active player base. If the pitfalls of the distributed network architecture can be ameliorated, there is not much standing in the way of a game development studio implementing an MMOG in a distributed fashion.

Badumna [40] is a commercially available, decentralized network engine designed for use in networked game play. Although no commercially successful games have been launched using the Badumna engine at time of writing, the existence of such a tool is indication that the video game industry has given decentralized hosting of MMOGs a fair amount of consideration as a viable alternative to client/server-based architectures.

ImonCloud [28] is another platform being developed to ease the development of P2P MMOGs. The ImonCloud platform handles scaling, interest management, server redundancy, load management and message dissemination transparently while the developers can focus on game logic. Players connect to their geographically closest entry server, which connects to the ImonCloud service on the player's behalf. The entry server communicates through a communication layer with the app server, data storage and lobby of the platform for game logic execution, before returning the updates via the communication layer and the entry servers back to the players. This platform is still in development, with the aim of lowering the barrier to entry for smaller developers to build distributed NVEs.

Bibliography

[1] M. A Vouk, "Cloud computing–issues, research and implementations," *CIT. Journal of Computing and Information Technology*, vol. 16, no. 4, pp. 235–246, 2008.

[2] Amazon Web Services Inc. (2013) Aws | amazon elastic compute cloud (ec2) - scalable cloud servers. Accessed: Oct 2013. [Online]. Available: http://aws.amazon.com/ec2/

[3] M. Amoretti, "A survey of peer- to -peer overlay schemes: Effectiveness, efficiency and security," *Recent Patents on Computer Science*, vol. 2, no. 3, pp. 195–213, 2009.

[4] G. Armitage, "An experimental estimation of latency sensitivity in multiplayer quake 3," in *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, Sept 2003, pp. 137–141.

[5] H. Backhaus and S. Krause, "Quon: a quad-tree-based overlay protocol for distributed virtual worlds," *Int. J. Adv. Media Commun.*, vol. 4, no. 2, pp. 126–139, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1504/IJAMC.2010.032139

[6] I. Baumgart, B. Heep, and S. Krause, "Oversim: A flexible overlay network simulation framework," in *2007 IEEE Global Internet Symposium*, May 2007, pp. 79–84.

[7] M. d. Berg, *et al.*, *Computational Geometry: Algorithms and Applications*, 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[8] A. Bharambe, *et al.*, "Donnybrook: Enabling large-scale, high-speed, peer-to-peer games," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 389–400, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1403002

[9] Blizzard Entertainment. (2013, Oct.) World of warcraft official game site. Accessed: Oct 2013. [Online]. Available: http://eu.battle.net/wow/en/

[10] I. Blizzard Entertainment. (2015) World of warcraft system requirements. Accessed: Dec 2015. [Online]. Available: https://us.battle.net/support/en/article/world-of-warcraft-system-requirements

[11] J.-S. Boulanger, "Interest Management for Massively Multiplayer Games," Master's Thesis, McGill University, 2006.

[12] J.-S. Boulanger, "Interest management for massively multiplayer games," Master's thesis, McGill University, School of Computer Sciences, August 2006.

[13] J.-S. Boulanger, J. Kienzle, and C. Verbrugge, "Comparing interest management algorithms for massively multiplayer games," in *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, ser. NetGames '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1230040.1230069

[14] E. Buyukkaya, M. Abdallah, and R. Cavagna, "Vorogame: A hybrid p2p architecture for massively multiplayer games," in *2009 6th IEEE Consumer Communications and Networking Conference*, Jan 2009, pp. 1–5.

[15] M. Castro, *et al.*, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489–1499, Oct 2002.

[16] J. Chen, *et al.*, "Locality aware dynamic load management for massively multiplayer games," in *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '05. New York, NY, USA: ACM, 2005, pp. 289–300. [Online]. Available: http://doi.acm.org/10.1145/1065944.1065982

[17] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, p. 40, 2006.

[18] Cooperative Association for Internet Data Analysis. (2013, Jul) The cooperative association for internet data analysis: skitter. Accessed: Oct 2013. [Online]. Available: http://www.caida.org/tools/measurement/skitter/

[19] Daeity. (2011, Mar.) Digital castration: World of warcraft operating costs. Accessed: Oct 2013. [Online]. Available: http://www.wowguideonline.com/blog/2011/03/25/digital-castration-world-of-warcraft-operating-costs/

[20] H. A. Engelbrecht and G. Schiele, "Transforming minecraft into a research platform," in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Jan 2014, pp. 257–262.

[21] L. Fan, P. Trinder, and H. Taylor, "Design issues for peer-to-peer massively multiplayer online games," *Int. J. Adv. Media Commun.*, vol. 4, no. 2, pp. 108–125, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1504/IJAMC.2010.032138

[22] S. Fiedler, M. Wallner, and M. Weber, "A communication architecture for massive multiplayer games," in *Proceedings of the 1st Workshop on Network and System Support for Games*, ser. NetGames '02. New York, NY, USA: ACM, 2002, pp. 14–22. [Online]. Available: http://doi.acm.org/10.1145/566500.566503

[23] S. Fortune, "A sweepline algorithm for voronoi diagrams," in *Proceedings of the Second Annual Symposium on Computational Geometry*, ser. SCG '86. New York, NY, USA: ACM, 1986, pp. 313–322. [Online]. Available: http://doi.acm.org/10.1145/10515.10549

[24] D. Frey, *et al.*, "Solipsis: A decentralized architecture for virtual environments," in *1st International Workshop on Massively Multiuser Virtual Environments*, 2008, pp. 29–33.

[25] Gaikai Inc. Gaikai.com. Accessed: Feb 2014. [Online]. Available: https://www.gaikai.com/

[26] J. Gilmore and H. Engelbrecht, "Pithos: a state persistency architecture for peer-to-peer massively multiuser virtual environments," *2011 IEEE International Workshop on Haptic Audio Visual Environments and Games*, no. March, pp. 1–6, 2011.

[27] S. Y. Hu, S. C. Chang, and J. R. Jiang, "Voronoi state management for peer-to-peer massively multiplayer online games," in *2008 5th IEEE Consumer Communications and Networking Conference*, Jan 2008, pp. 1134–1138.

[28] S. Y. Hu and M. Lien, "Imoncloud: Easing development and deployment for scalable networked games," in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Jan 2014, pp. 251–256.

[29] S. Y. Hu, *et al.*, "A spatial publish subscribe overlay for massively multiuser virtual environments," in *Electronics and Information Engineering (ICEIE), 2010 International Conference On*, vol. 2, Aug 2010, pp. V2–314–V2–318.

[30] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "Von: A scalable peer-to-peer network for virtual environments," *IEEE Network: The Magazine of Global*

Internetworking, vol. 20, no. 4, pp. 22–31, July 2006. [Online]. Available: http://dx.doi.org/10.1109/MNET.2006.1668400

[31] S.-Y. Hu and G.-M. Liao, "Scalable peer-to-peer networked virtual environment," in *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, ser. NetGames '04. New York, NY, USA: ACM, 2004, pp. 129–133. [Online]. Available: http://doi.acm.org/10.1145/1016540.1016552

[32] S.-Y. Hu and G.-M. Liao, "Scalable peer-to-peer networked virtual environment," in *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, ser. NetGames '04. New York, NY, USA: ACM, 2004, pp. 129–133. [Online]. Available: http://doi.acm.org/10.1145/1016540.1016552

[33] S.-Y. Hu and G.-M. Liao, "Scalable peer-to-peer networked virtual environment," in *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, ser. NetGames '04. New York, NY, USA: ACM, 2004, pp. 129–133. [Online]. Available: http://doi.acm.org/10.1145/1016540.1016552

[34] S. Hu, "Spatial publish subscribe," March 2009. [Online]. Available: http://pap.vs.uni-due.de/MMVE09/papers/p8.pdf

[35] J.-R. Jiang, Y.-L. Huang, and S.-Y. Hu, "Scalable aoi-cast for peer-to-peer networked virtual environments," in *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems Workshops*, ser. ICDCSW '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 447–452. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.Workshops.2008.80

[36] J. Keller and G. Simon, "Solipsis: A massively multi-participant virtual world." pp. 262–268, 2003. [Online]. Available: http://dblp.uni-trier.de/db/conf/pdpta/pdpta2003-1.html#KellerS03

[37] B. Knutsson, *et al.*, "Peer-to-peer support for massively multiplayer games," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1, March 2004, p. 107.

[38] S. Krause, "A case for mutual notification: A survey of p2p protocols for massively multiplayer online games," in *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, ser. NetGames '08. New York, NY, USA: ACM, 2008, pp. 28–33. [Online]. Available: http://doi.acm.org/10.1145/1517494.1517500

[39] S. Kulkarni, "Badumna network suite: A decentralized network engine for massively multiplayer online applications," pp. 178–183, Sept 2009.

[40] S. Kulkarni, S. Douglas, and D. Churchill, "Badumna: A decentralised network engine for virtual environments," *Comput. Netw.*, vol. 54, no. 12, pp. 1953–1967, Aug. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.05.015

[41] D. Kushner, "Engineering everquest: online gaming demands heavyweight data centers," *IEEE Spectrum*, vol. 42, no. 7, pp. 34–39, July 2005.

[42] Linden Research. (2013, Oct.) Second life official game site. Accessed: Oct 2013. [Online]. Available: http://secondlife.com/

[43] Y. Makbily, C. Gotsman, and R. Bar-Yehuda, "Geometric algorithms for message filtering in decentralized virtual environments," in *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, ser. I3D '99. New York, NY, USA: ACM, 1999, pp. 39–46. [Online]. Available: http://doi.acm.org/10.1145/300523.300527

[44] E. Makuch. (2012, Jan.) Star Wars: The Old Republic cost $200 million to develop. Accessed: Oct 2013. [Online]. Available: http://www.gamespot.com/articles/star-wars-the-old-republic-cost-200-million-to-develop/1100-6348959/

[45] Mammoth Team. (2011) Mammoth: a multiplayer game research framework. Accessed: Oct 2013. [Online]. Available: http://mammoth.cs.mcgill.ca/

[46] J. McGonigal, *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. Penguin Group , The, 2011.

[47] D. C. Miller and J. A. Thorpe, "Simnet: the advent of simulator networking," *Proceedings of the IEEE*, vol. 83, no. 8, pp. 1114–1123, Aug 1995.

[48] Miniwatts Marketing Group. (2013, Aug.) Internet growth statistics - the global village online. Accessed: Oct 2013. [Online]. Available: http://www.internetworldstats.com/emarketing.htm

[49] K. L. Morse *et al.*, *Interest management in large-scale distributed simulations*. Information and Computer Science, University of California, Irvine, 1996.

[50] MuchDifferent. (2013) "unitypark suite developer site | general - server architecture". Accessed: Dec 2015. [Online]. Available: http://developer.muchdifferent.com/unitypark/General/ServerArchitecture

[51] M. T. Najaran and C. Krasic, "Scaling online games with adaptive interest management in the cloud," in *Network and Systems Support for Games (NetGames), 2010 9th Annual Workshop on*, Nov 2010, pp. 1–6.

[52] F. Nolte, "World of warcraft player tracking software," 2011, final year project.

[53] L. Pantel and L. C. Wolf, "On the suitability of dead reckoning schemes for games," in *Proceedings of the 1st Workshop on Network and System Support for Games*, ser. NetGames '02. New York, NY, USA: ACM, 2002, pp. 79–84. [Online]. Available: http://doi.acm.org/10.1145/566500.566512

[54] I. Rackspace US. (2015) Rackspace cloud pricing calculator. Accessed: Dec 2015. [Online]. Available: http://www.rackspace.com/calculator

[55] A. E. Rhalibi and M. Merabti, "Interest management and scalability issues in p2p mmog," in *CCNC 2006. 2006 3rd IEEE Consumer Communications and Networking Conference, 2006.*, vol. 2, Jan 2006, pp. 1188–1192.

[56] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, Aug 2001, pp. 99–100.

[57] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, ser. Middleware '01. London, UK, UK: Springer-Verlag, 2001, pp. 329–350. [Online]. Available: http://dl.acm.org/citation.cfm?id=646591.697650

[58] A. I. T. Rowstron, *et al.*, "Scribe: The design of a large-scale event notification infrastructure," in *Proceedings of the Third International COST264 Workshop on Networked Group Communication*, ser. NGC '01. London, UK, UK: Springer-Verlag, 2001, pp. 30–43. [Online]. Available: http://dl.acm.org/citation.cfm?id=648089.747486

[59] G. Schiele, *et al.*, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, ser. CCGRID '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 773–782. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2007.97

[60] N. Sheldon, *et al.*, "The effect of latency on user performance in warcraft iii," in *Proceedings of the 2Nd Workshop on Network and System Support for*

Games, ser. NetGames '03. New York, NY, USA: ACM, 2003, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/963900.963901

[61] A. Statista. (2013, July) Number of world of warcraft subscribers from 1st quarter 2005 to 2nd quarter 2013 (in millions). Accessed: Oct 2013. [Online]. Available: http://www.statista.com/statistics/208146/number-of-subscribers-of-world-of-warcraft/

[62] I. Statista. (2015) Countries with fastest internet 2015. Accessed: Dec 2015. [Online]. Available: http://www.statista.com/statistics/204952/average-internet-connection-speed-by-country/

[63] I. Stoica, *et al.*, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, Aug. 2001. [Online]. Available: http://doi.acm.org/10.1145/964723.383071

[64] U. Tech. (2015) Gamasutra - crowfall, and saving the mmo from 'stagnation'. Accessed: Dec 2015. [Online]. Available: http://www.gamasutra.com/view/news/237147/Crowfall_and_saving_the_MMO_from_stagnation.php

[65] The Trustees of Princeton University. (2007) Planetlab | an open platform for developing, deploying, and accessing planetary-scale services. Accessed: Oct 2013. [Online]. Available: http://www.planet-lab.org

[66] A. Varga. (2013, Sept) OMNeT++. Technical University of Budapest. Accessed: Oct 2013. [Online]. Available: http://www.omnetpp.org/

[67] A. Yahyavi and B. Kemme, "Peer-to-peer architectures for massively multi-player online games: A survey," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 9:1–9:51, July 2013.

[68] S. Yamamoto, *et al.*, "A distributed event delivery method with load balancing for mmorpg," in *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, ser. NetGames '05. New York, NY, USA: ACM, 2005, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/1103599.1103610

[69] A. P. Yu and S. 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*, ser. NOSSDAV '05. New York, NY, USA: ACM, 2005, pp. 99–104. [Online]. Available: http://doi.acm.org/10.1145/1065983.1066007