Technical Report

UCAM-CL-TR-809
ISSN 1476-2986

Number 809

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Distributed virtual environment scalability and security

John L. Miller

October 2011

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted October 2011 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Hughes Hall.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

Distributed virtual environments (DVEs) have been an active area of research and engineering for more than 20 years. The most widely deployed DVEs are network games such as *Quake*, *Halo*, and *World of Warcraft* (*WoW*), with millions of users and billions of dollars in annual revenue. Deployed DVEs remain expensive centralized implementations despite significant research outlining ways to distribute DVE workloads.

This dissertation shows previous DVE research evaluations are inconsistent with deployed DVE needs. Assumptions about avatar movement and proximity - fundamental scale factors - do not match WoW's workload, and likely the workload of other deployed DVEs. Alternate workload models are explored and preliminary conclusions presented. Using realistic workloads it is shown that a fully decentralized DVE cannot be deployed to today's consumers, regardless of its overhead.

Residential broadband speeds are improving, and this limitation will eventually disappear. When it does, appropriate security mechanisms will be a fundamental requirement for technology adoption.

A trusted auditing system ("Carbon") is presented which has good security, scalability, and resource characteristics for decentralized DVEs. When performing exhaustive auditing, Carbon adds 27% network overhead to a decentralized DVE with a WoW-like workload. This resource consumption can be reduced significantly, depending upon the DVE's risk tolerance.

Finally, the Pairwise Random Protocol (PRP) is described. PRP enables adversaries to fairly resolve probabilistic activities, an ability missing from most decentralized DVE security proposals.

Thus, this dissertation's contribution is to address two of the obstacles for deploying research on decentralized DVE architectures. First, lack of evidence that research results apply to existing DVEs. Second, the lack of security systems combining appropriate security guarantees with acceptable overhead.

Acknowledgements

There are several people without whom this research never could have taken place. Without them, I never would have tried, let alone succeeded.

First and foremost, thank you to Mitch Goldberg, Andrew Herbert, and Microsoft Research Cambridge for their support in undertaking a PhD. Andrew provided words of encouragement at the right time to tip the balance towards pursuing this degree. Mitch's unflagging confidence, optimism, encouragement and support helped make sure I did not abandon the effort when the going got tough. Thank you Mitch!

Thank you to my advisor Jon Crowcroft for his guidance and patience with ideas good and bad, and invaluable insights into what constitutes useful research. And thanks to Steve Hand for his feedback on both my thesis proposal and this dissertation, which improved the quality of both.

Finally, THANK YOU to my wife Salwa and son Ryan for their patience and support. In addition to putting up with years of my burning the candle both ends, Salwa provided invaluable assistance, ensuring results were correctly presented and human-readable. In addition to her computer science insights and technical proof-reading skills, she is a veteran hand at *World of Warcraft*, and helped to gather much of the data this research is based upon. Thank you Salwa, and thank you Ryan!

Table of contents

1 Introduction 11

1.1 Research statement 11

1.2 Research contributions 11

1.3 Publications 12

2 General background 13

2.1 DVE responsiveness and correctness 15

2.2 DVE scalability 17

2.2.1 Traffic reduction 17

2.2.2 Redistributing simulation workload 19

2.3 DVE security 21

3 Towards a realistic DVE workload 25

3.1 Introduction 25

3.2 Relevant background 26

3.2.1 World of Warcraft 26

3.2.2 The Arathi Basin battleground 27

3.2.3 Avatar behavior and traffic classification 28

3.3 Methodology 29

3.4 Analysis 31

3.4.1 Avatar participation characteristics 32

3.4.2 Waypoints 33

3.4.3 Hotspots 36

3.4.4 Group movement 39

3.5 Conclusions 42

4 Near-term infeasibility of P2P DVEs 43

4.1 Introduction 43

4.2 Background 43

4.2.1 World of Warcraft 43

4.2.2 Broadband speeds 44

4.3 Methodology 45

4.3.1 World of Warcraft network attributes 46

4.3.2 Simulation trace generation 47

4.3.3 Simulator 48

4.3.4 Topology choices and metrics 49

4.3.5 Simulation characteristics 49

4.3.6 Simulator validation ... 50
4.4 Results ... 51
4.4.1 Simulation results ... 51
4.5 Conclusions ... 55
5 Trusted auditing of decentralized DVEs ... 57
5.1 Related work ... 57
5.1.1 DVE threat models ... 57
5.1.2 P2P DVE frameworks ... 58
5.1.3 DVE security work ... 58
5.1.4 General DVE characteristics ... 59
5.2 Threat model ... 61
5.3 Carbon ... 63
5.3.1 Nomenclature and DVE requirements ... 64
5.3.2 Carbon audit client: "Reporter" ... 65
5.3.3 Carbon auditor: "Auditor" ... 66
5.3.4 Carbon system operation ... 67
5.4 Analysis ... 69
5.4.1 Audit coverage ... 69
5.4.2 Comparison with PeerReview ... 72
5.4.3 Carbon and PeerReview client overhead analysis ... 73
5.4.4 Carbon and PeerReview auditors and overhead ... 75
5.5 Conclusions ... 77
6 Untrusted collaboration ... 79
6.1 Introduction ... 79
6.2 Related work ... 79
6.2.1 Secure multi-party computation ... 80
6.3 Pairwise Random Protocol (PRP) ... 80
6.3.1 Resolving a single action ... 81
6.3.2 Resolving an unbounded random sequence ... 83
6.4 Results ... 83
6.4.1 Security ... 84
6.4.2 Performance ... 85
6.5 Conclusions ... 86
7 Conclusions and future work ... 87
Bibliography ... 91

1 Introduction

Modern distributed virtual environments – DVEs – are typically immersive 3D real-time simulations. The most prevalent examples are online computer games, such as *Call of Duty: Black Ops* [2] and *World of Warcraft* [21] (abbreviated *WoW*).

DVEs are commercially successful, but are constrained by the state of the art in networking and software: popular deployed DVEs use a client-server model for maintaining the virtual environment and propagating simulation state. This design choice greatly simplifies security and design, but has scalability ramifications.

Over the last fifteen years, a significant body of research has emerged exploring ways to improve DVE scalability. Many of the research results appear promising, with evaluations showing their ability to scale well past the scope of deployed DVEs. Why, then, are these advances not used in any broadly deployed DVE?

1.1 Research statement

Previous DVE research makes a wide variety of assumptions in terms of message workloads, latency, and the importance of security. If these assumptions do not match the needs and behavior of DVEs intended for broad deployment, the evaluations are less likely to persuade DVE developers to consider adopting said research.

I hypothesize that assumptions for evaluating previous DVE research are incorrect. This hypothesis will be investigated in this dissertation by examining *World of Warcraft's* messaging model, which is typical for Massively Multiplayer Online Games (MMOGs).

I also hypothesize that when one takes into account the state of residential Internet connections worldwide, a pure peer-to-peer DVE which meets existing DVE messaging requirements cannot be deployed. On a happier note, residential Internet access continues to improve, and it should only be a matter of time before network resources are no longer a barrier to deploying peer-to-peer versions of DVEs similar to those available today.

Most peer-to-peer DVE research proposals do not include security solutions, but one is absolutely required for broad deployment.

I demonstrate that an auditing solution using trusted auditors can provide good security for decentralized DVEs with acceptable overhead, both for the DVE operator and for clients.

The next section provides details on the contributions my research and this dissertation provide towards these research goals.

1.2 Research contributions

This thesis contains contributions across three areas of distributed virtual environment research, as described in the previous section.

Chapter 3 examines assumptions affecting messaging workloads in *World of Warcraft*. DVE Messages are typically propagated based on proximity to message source, so realistic avatar movement and grouping models are critical for evaluation correctness. Existing workloads make significant incorrect assumptions about avatar movement and clustering. This dissertation provides in-depth evaluation of avatar movement in *World of Warcraft* battlegrounds, a player-avatar-driven scenario. While it stops short of proposing new movement models, it provides a wealth of relevant information for those wishing to do so.

Chapter 4 details investigation into the feasibility of pure peer-to-peer DVE architectures, given the results from chapter 3 and real-world network resource constraints. It provides simulator results showing that even with liberal assumptions including perfect information with zero overhead for peer-

to-peer infrastructure maintenance, average and peak message latency would be too high for a pure peer-to-peer solution to be viable.

Chapter 5 proposes a security solution to secure decentralized (i.e. non-client-server) DVEs. Although pure peer-to-peer DVEs are not deployable today, hybrid solutions with peer specialization may be. I propose *Carbon*, an auditing system allowing untrusted adversaries to interact within the DVE without requiring immediate intervention from a trusted third party. Carbon provides a tunable model for choosing the level of protection desired, based upon DVE requirements. Overhead and protection are compared to *PeerReview* [60], a very effective security solution for general distributed scenarios.

Finally, chapter 6 proposes the *Pairwise Random Protocol*. This is a variation on secure coin flipping which allows adversaries to fairly decide probabilistic events. This extension supports an important category of DVE interactions vital to most DVEs, yet typically ignored by existing research.

1.3 Publications

Some results presented in this thesis have been published elsewhere during the course of research. Publications by chapter are listed below.

Chapter 3: Towards a realistic DVE workload

- John L. Miller and Jon Crowcroft. Avatar Movement in World of Warcraft Battlegrounds. In *Proceedings of the 8th Annual Workshop on Network and Systems Support for Games*, 2009.
- John L. Miller and Jon Crowcroft. Group Movement in World of Warcraft Battlegrounds. In *International Journal of Advanced Media and Communication*, Issue 4, Volume 4, December 2010.

Chapter 4: Near-term infeasibility of P2P DVEs

- John L. Miller and Jon Crowcroft. The Near-Term Feasibility of P2P MMOGs. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, 2010.

Chapter 5: Trusted auditing of decentralized DVEs

- John L. Miller and Jon Crowcroft. Carbon: trusted auditing for P2P distributed virtual environments. In *Technical Report TR-753,* University of Cambridge, 2009.

Chapter 6: Untrusted collaboration

- John L. Miller and Jon Crowcroft. Probabilistic Event Resolution with the Pairwise Random Protocol. In *NOSSDAV '09: Proceedings of the 19th International Workshop on Network and Operating System Support for Digital Audio and Video,* 2009.

2 General background

This chapter provides high level description of distributed virtual environments - DVEs - and their properties. It also summarizes published DVE scalability and security research. Those interested in an overall bibliography for online games and DVE research should consider visiting the bibliography of network games research at http://www.iis.sinica.edu.tw/~swc/ngbib.html for a broad set of references, including many not cited here.

I define a virtual environment (VE) as a computer simulation typically involving space and time. This definitely restricts our examination to computerized systems, where otherwise it could rightly include war simulations going back thousands of years [93]. In addition to military applications, VEs are a valuable tool in domains such as engineering. Computers have been used to simulate virtual environments since the inception of electronic computers. For example, computers were used in the Manhattan Project to model nuclear detonation, implementing a narrowly scoped, non-real-time virtual environment.

In subsequent years computer capacity has grown, and with it the scope and responsiveness of virtual environments. VEs are still used for offline numerical simulations, but are now able to simulate complex 2D and 3D environments in real-time.

With the advent of computer networks, virtual environments were adapted to utilize multiple hosts, communicating over a network. This category of virtual environment has been known by several names, including networked virtual environment, collaborative virtual environment, or more recently, distributed virtual environment. “Distributed virtual environment” or DVE, as used in this dissertation, refers to any VE with a significant networking aspect.

A DVE can be differentiated from a centralized virtual environment (VE) by the maintenance of state across two or more VE nodes. For this dissertation’s purposes, a simulation run on a single server with only raw input and display - such as keystrokes and render frames - sent between the server and remote clients is not considered a DVE. A server-based VE which delegates some simulation, state calculation, or state storage to remote clients *is* considered a DVE. A server-based VE which leverages multiple cooperating servers is also considered a DVE.

A DVE can most generally be thought of as a collection of state, and rules for updating that state. In *World of Warcraft* (WoW), state includes the race and class of an avatar, its position, velocity, speed, and health. It includes every changeable aspect of the environment, from the status of walls which can be broken (whole, damaged, or destroyed) to the time of day and weather in each part of the world.

State management and rendering in DVEs, even client-server DVEs, is extremely tricky. Users interact with the DVE in real-time and expect to see their inputs acted upon immediately. Yet it takes time for their inputs to be locally processed and vetted, transmitted to the server for validation, and the change to be approved and disseminated to other nodes. Network latency and processing delays are a serious obstacle for real-time DVE experiences: DVEs use complex strategies to provide the illusion of a single, continuous world state, when really perception and rendering of the world state can be different at every single node participating in the DVE. These complexities are part of what makes DVEs such fertile ground for research. The utility of DVEs is what makes DVE research valuable.

DVEs have a variety of applications. Some implement large-scale simulations, such as military war game training exercises [103]. Others are used as a shared virtual reality for therapeutic or social purposes, as in *Therapy World* [108] and *Second Life* [86]. And, they can be used for entertainment, as in network-enabled games. *Spasim* [26], released in 1974, was one of the first DVEs, and claims to be the first 3D multiplayer game. *Spasim* allowed up to 32 players to compete with each other in a wire-frame rendered space setting.

Most DVE research and implementation in the 1970s and 1980s was based around military combat simulations. One of the first large-scale military simulators was the Simulator Networking project (SIMNET) [112], started in 1983 and primarily developed up until 1990. At its conclusion, the SIMNET environment consisted of a network of 250 simulators across 11 sites [98]. SIMNET

introduced many concepts still in used today, such as node treatment of simulation entities as objects with an authoritative owner providing state updates, and the use of dead reckoning to provide smoother simulation behavior between updates.

The lessons learned in constructing SIMNET evolved into the Distributed Interactive Simulation (DIS) protocols, whose first version was completed in 1992 and published as standard IEEE 1278-1993 [71] in 1993, and revised several times since [69] [70]. SIMNET was a successful demonstration of the potential of distributed simulation, but was never intended to provide an open, well documented protocol. The DIS effort was implemented to address this shortcoming, by specifying and standardizing all aspects of communications between nodes. The architecture was quite similar to SIMNET, but more fully specified [121]. In both cases, simulations were restricted to thousands of entities, as messages were broadcast from each node to all other participants, and quickly became a scaling bottleneck.

As the personal computer became ubiquitous and computer networks grew together into the Internet in the 1990s, consumer applications for DVEs - specifically networked computer games - became increasingly common.

Doom [67], a computer game released in 1993, helped establish the PC as a local network gaming platform, introducing millions of people to DVEs. As the Internet matured and consumer Internet access become more common, DVE games began to operate over the Internet as well as local networks, allowing more DVE instances to be created, and more people to participate in them. For example, the *Quake* [68] and *Unreal* [6] games allowed players to join games hosted at arbitrary servers around the world, or even to host their own game on their own machine. Although millions of people were playing these games, most DVE instances remained relatively small, supporting between 2 and 8 players.

In the late 1990s, a new type of network game DVE came into prominence, the *massively multiplayer online game* (MMOG). Rather than providing a short scenario in which only a few players could participate, these games provide a years-long experience supporting hundreds of thousands – or even millions – of simultaneous participants.

The first MMOGs - such as *Ultima Online* [105], released in 1997 - were descendants of pen-and-paper role playing games, and of multiplayer text adventures, known as multi-user dungeons or MUDs. Like MUDs, most MMOGs rely upon a central server or server farm to simulate each world instance and maintain consistency within that instance. Unlike MUDs, participating client machines are responsible for simulating, maintaining, and rendering portions of the world state.

MMOGs ushered in a new era of commercial success and economic incentives. Players regularly played tens of hours per week, and paid monthly subscription fees for the privilege of playing in many of the DVEs. In the earlier small-scale network game DVEs, network gaming was an additional feature of the game. With the new generation of MMOGs, the game could only be played over the network. Its value and experience were tightly coupled with the servers and the other players involved in the game.

In the twenty years since game DVEs became mainstream, they have evolved into three basic game types, each played by millions of people worldwide.

1. ***First-person shooters*** **(FPS)**. The vast majority of FPSs have less than 10 users per instance, frequent avatar state updates (e.g. every 40 ms), and high sensitivity to latency. Game state is not typically persisted across game sessions, but instead short-lived scenarios are played time and time again. Current examples include *Call of Duty: Black Ops* [2], *Medal of Honor* [48], and *Gears of War* [94].
2. **Strategy games**, especially *real-time strategy* **(RTS)** games. Most of these support less than 10 users per instance, have less frequent updates (e.g. every second), and relatively low sensitivity to latency. These games are usually scenario-based, with game state reset after each scenario is played. Examples include the *StarCraft* [20] series and *Civilization* [53].

3. ***Role-playing games*** **(RPG)**. RPGs vary from offering a few small scenarios to play through such as *Diablo II* [23], to presenting worlds with hundreds of square kilometers of unique terrain, such as *World of Warcraft* [21]. The focus in these games is on a progressive experience, where player state and accomplishments are stored across sessions for anywhere from tens to thousands of hours per player. Game session populations can range from a few players up to thousands of simultaneous players per server.

Network-enabled computer games are the most prominent example of DVEs with a wealth of technical and usage information available. They have tremendous value, in both economic and entertainment terms. For example, the release of *Modern Warfare 2* - a network-enabled first person shooter game - generated more than $550 million USD in sales its first week of release [1]. Its network play features were extremely popular, with nearly a million users simultaneously playing on Xbox-Live three days after its release. *WoW* has more than 12 million paying subscribers [22], with American and European customers paying approximately $15 USD per month to participate in its DVE. As of June 2010, the average *World of Warcraft* player logs more than 9 hours per week [106], for a total of more than 100 million person hours each week.

Despite the huge number of players online simultaneously, most deployed DVEs have significant scalability limitations. Most modern FPS games limit simultaneous participants in a game to less than 20. *Second Life* has a limit of a hundred avatars per region [85], less if they are active. *WoW* supports approximately 4,000 players per copy of the world ("shard"), though a more typical number is less than a thousand [130], and mutual interaction is limited to groups numbering in the low hundreds.

Enabling larger scale DVEs is desirable. In the commercial entertainment arena, it enables a vendor to offer more interesting experiences. In the military arena, it allows larger scale simulations to be performed, building competence in troop management and battle oversight, something difficult to acquire other ways.

A significant amount of research has been done in mechanisms to increase the scale of DVEs. Much of this research involves decentralizing DVE operations. For example, distributing object state ownership, changing messaging patterns, and employing novel caching schemes. Speculative updates – such as dead reckoning, used to success in SIMNET – are also an area of investigation.

As DVEs grow, the incentive of participants to cheat – violating the DVE's rules to gain unfair advantage - also grows. For example, a real money market trading in virtual assets – such as virtual currency - has evolved around MMOGs. Even ten years ago, this market was worth more than a billion dollars a year, with multinational corporations fighting for market dominance [42].

Cheating is believed to reduce the overall appeal of a game, and can result in players being less likely to continue to play that game [73]. Quality of the customer experience is critical to attracting and retaining customers. If customer experience is poor, the DVE will not sell well, people will be less likely to subscribe (if subscriptions are required), and the distributor will have a harder time selling any future DVE offerings.

2.1 DVE responsiveness and correctness

DVEs distribute their activities to varying degrees. For game DVEs, the end-user impression of efficacy will be related to the accuracy and the responsiveness of the experience. A user needs to be immersed in the experience, to feel as though it is responding to their inputs. They need to feel the experience is fair and – to the extent the scenario supports – predictable. For example, a user issuing a "turn left" command to their avatar's vehicle should have roughly the same result each time an avatar does it, assuming similar starting state. If the vehicle turns immediately some times, turns after a delay other times, or simply continues straight, the DVE feels neither responsive nor correct.

The lag between proposing a state change and that state change being communicated to or accepted by other elements of the DVE is called latency. Different activities can require different degrees of consensus before being acted upon. In the example above, if the local node is empowered to turn the vehicle, reaction can be immediate and responsive. However, if turning requires agreement from a

single remote node, then the action is not authoritative until a message has been communicated to the remote node, and an agreement returned. Further, a player at a third remote node may not perceive the vehicle turning for some period after the state change has been authoritatively made, because of network propagation delays.

Latency is a significant factor in user perception of DVE fairness, responsiveness, and correctness. Most deployed DVEs have latency induced by network propagation delay, so network propagation delay is the most common type of latency measured.

Latency sensitivity varies according to the type of DVE [39]. First person shooter (FPS) and other "twitch" DVEs are the most sensitive to latency. Real-time strategy (RTS) games are the least sensitive. Role playing games (RPGs) such as *World of Warcraft* are between these two extremes in terms of sensitivity.

In current generation FPS DVEs, estimates of tolerable latency range between 60 and 200 ms (ms). For example, Brun et al. [27] quote a study where player performance was not affected below 60 ms, but values above 60 ms had a negative correlation with player performance. Quax et al. [115] quote the same starting point for perceived latency. Dick et al. [43] found that *Unreal Tournament* performance is perceived by the player as degraded after 100 ms of latency, and showed significant drop in performance at 150 ms and higher latency. Armitage [8] found that *Quake 3* performance in terms of frags per minute dropped as a function of latency, with a clear downward trend starting between 50 ms and 100 ms round trip time. Beigbeider et al. [16] evaluated the impact of jitter, packet loss, and latency on *Unreal Tournament 2003* performance. They found that reasonable jitter and packet loss did not significantly impact performance, but there was a negative correlation between latency and performance in terms of frags and deaths, which started at 100 ms and became "annoying" at 200 ms.

Less evaluation has been done for RTS and RPG latency sensitivity. Fritsch et al. [55] studied players' ability to perform navigation and combat tasks in *Everquest 2* [122], an MMORPG. They found that latency above 250 to 500 ms increased the time required to successfully complete combat, reduced avatar resources remaining at the end of each combat, and increased time to navigate a fixed course. Claypool and Claypool [39] found that MMORPG players had significantly decreased performance at around 500 ms latency, and RTS players at around 1,000 ms of latency.

Research into collaborative activities outside of game DVEs has confirmed that latency is a significant factor in any shared activity. Hikichi et al. [63] performed an experiment with remote calligraphy tutoring. Interestingly, they found that in a shared activity where one participant is leading the other – such as instruction – both the teacher and instructor perceived some impairment with increasing latency. However, the teacher – who was leading the exercise and then observing the student's reaction – perceived a much greater impairment than the student, who was simply following the input stream from the teacher as it came in, regardless of when that stream was originated.

Latency impacts not only players' perceived and actual performance, but also the amount of time they choose to participate in the DVE. For example, Chen et al. [37] found that *Shen Zhou Online* players who had more than 200 ms of latency on average played less than 2/3 as long as those who had latency below 150 ms.

State update latencies in DVEs are often large enough to be perceivable. Early DVEs required all state updates to be validated by the server before they were rendered. This provided a correct state delay, but broke the illusion of continuous control.

Today, DVEs allow client nodes to work speculatively with the local state snapshot. Unimportant state decisions are left wholly to the local node, and may not appear consistent between all nodes. For example, decorative aspects of the environment may be randomly added at each node, but not synchronized between them.

Moderately important states can have changes proposed and rendered by a local node, but that node may have to revert the value if the proposal is denied, resulting in inconsistent rendering at the local

node. For example, the user could have proposed to pick up an item (changing its "owned by" state) and have rendered locally picking up the item, but then when the request is denied, the item disappears.

Finally, critical state may await rendering / update of the local cached value until confirmation of the proposed change is received. This provides the worst experience for the user, and so is not often used in modern DVEs.

These three choices show the tradeoff between responsiveness and correctness. Enabling responsive behavior in DVEs inevitably affects correctness, as not all state conflict can be easily resolved. Mauve [91] gives an example of three DVE participants in such a situation. Player A shoots a bullet at player B which will kill player B. Player B shoots at player C after the bullet from A should have arrived, but before receiving that update. The bullet from B hits C, but should never have been shot. Two common mitigations for this problem are local lag and time warp [92].

Local lag defers executing proposed state changes for a certain period. Change proposals are transmitted with time stamps. Local interpretation of state proposals lags behind the actual time slightly. In the example above, B would still send their "shoot at C" message, but would ideally receive the "shot by A" message before acting upon that proposed state change, and so would know to ignore it. C would also ideally receive both messages in time to correctly resolve the conflict and cancel being shot.

Time warp means rolling back state when conflicting state changes are approved as executed. Chains of conflicting changes can be arbitrarily complex, and so rolling them back can be difficult.

Note that local lag relies upon nodes being well behaved for success. It can be easily exploited by cheaters, as will be discussed in detail in section 2.3, after an outline of DVE scalability research.

2.2 DVE scalability

Most DVEs broadly deployed today are online games. These games have significant scalability limitations. For example, first person shooters are typically limited to between 8 and 16 mutually interacting players, though some (such as *Medal of Honor Allied Assault* [47]) go as high as 64 players. MMORPGs blur this line. *World of Warcraft* allows thousands of active avatars to share the same server, but in practice no more than a few hundred can gather within mutual interaction distance of each other without causing performance problems, or even crashing the server.

Significant effort has been put into investigating resource consumption of game DVEs, and devising ways to apply more resources to the DVE. The main dimensions researched are reduction of network traffic required, and redistribution of simulation workload.

2.2.1 Traffic reduction

Traffic reduction efforts have been focused in two dimensions: reducing the number of parties a given message is sent to, and reducing the number of messages required for the same fidelity of DVE.

DVEs are a collection of state information, and a process for updating those states. Many state changes can be localized. For example, a DVE node simulating an avatar only needs to be aware of state changes which affect that avatar. If two avatars are interacting, then many state changes resulting from that interaction can be determined using only the state of the two avatars. Further, resolution of the combat between those avatars is probably not relevant to avatars on the other side of the world.

Two categories of solutions exploit this behavior: region-based computation and message propagation, and area of interest (AoI) based message propagation.

In region-based methods, the DVE is divided into regions which act as compartments for state and/or for message propagation related to that state. Nodes subscribe to regions they are interested in, typically the region they maintain state for, and regions near their avatar. Regions can be fixed – for example divided into a uniform grid – or variable in size. In some cases, discussion of region management is treated as synonymous with load distribution. Note that most FPS DVEs can be

considered as region-based with a single region. *Second Life* also uses this model, with the world divided into a large number of small regions, each owned by exactly one simulation server.

Yamamoto et al. [133] propose dividing the game world into identically sized regions for management of resources and propagation of messages. Lee and Lee [83] divide the DVE into even sized regions determined by the number of servers processing data, e.g. 16 grids per server, with 16 servers, for a total of 256 regions. FreeMMG [32] divides the DVE into fixed regions, requiring nodes within each region to agree with each state change before that change is ratified. Jardine and Zappala [74] propose a hybrid architecture using fixed regions to limit propagation of avatar movement messages. Chan et al. divide the world into fixed slices – synonymous with regions – for computation in their Hydra [35] architecture.

Sometimes regions are dynamically rather than statically created. Alvic-NG [114] uses a quadtree to partition the world and allow subdivision of processing, with new sub-partitions created when a given partition load is unacceptably high.

In area of interest (AoI) based propagation, objects within the DVE which send or receive messages have an AoI and an aura. An object sends notification messages to any other object whose area of interest intersects its aura. In the degenerate case, an object's aura is a point centered on the object, and so messages are sent to objects based solely on the area of interest of the potential recipient.

AoI approaches ideally propagate messages only to parties which need them. Boulanger et al. give a good overview and comparison of existing AoI schemes in [25]. They experimentally obtain both the maintenance overhead and the message propagation properties of the schemes they evaluate, ranging from aura / AoI distance to tile-traversal distance for various tiling strategies (which take into account obstructions) to Delaunay triangulation, and finally, ray tracing from source to sink, which was used as ground truth. The authors found their results varied according to DVE spatial properties and frequency of occlusion, but in general tile-based approaches provided a good compromise between accuracy and speed of execution.

Refinements to reduce unnecessary message propagation are one message-related way to improve DVE scalability. However, it may be possible that some of the messages propagated to a node – despite being theoretically relevant to that node – are unimportant. In these cases, the unimportant messages can be deprioritized, and sent less frequently or with a lower priority than critical messages.

A couple of notable attention-based schemes have been proposed in literature. Donnybrook [17] calculates a focus for a local node, and then adapts the update rate from other avatars based on the attention the local node is paying to them. The half dozen most significant avatars send the node frequent updates, with others sending updates up to an order of magnitude less often. Avatar movement with this lower frequency is smoothed using bot movement patterns. This approach was implemented on a modified copy of *Quake 3*. Experimental evaluation [109] showed that most players were satisfied with the experience this approach provides.

MultiVR [44] uses a different approach. It calculates interest of the user's avatar in other event sources (avatars), and scales the number and kind of updates according to the interest metric. However, it does not pursue any specific modeling of the user's behavior to fill in gaps, assuming instead that the user will not notice the less frequent updates.

SmartCU3D [129] proposes that interest relationships are more complex than just distance or observer focus, and that characteristics of both the observer and the potentially observed objects (such as category or kind), should help determine whether or not to propagate state information, and how frequently to do so.

Beeharee et al. measure visual attention, and propose the VABIC [14] model for characterizing visual importance of objects, and therefore updates required to appropriately render those objects for a given observer. They applied their approach to a city simulation with observers walking around the simulated city, and found they could in some cases reduce the number of required updates by 70%.

2.2.2 Redistributing simulation workload

Redistribution of simulation workloads runs a broad spectrum ranging from server-based solutions to pure peer-to-peer solutions. In broad strokes, these solutions can be roughly divided into three categories: server-oriented solutions; trusted third party solutions; and client-oriented solutions.

2.2.2.1 Server-oriented

Server-based solutions are centralized solutions where most significant state storage and calculation for the DVE is performed on one or more servers owned by the DVE operator. This allows the DVE to benefit from operating on a known, trusted configuration, and to apply well-known maintenance and security techniques.

A single server solution does not need to coordinate its maintenance of state and so can always provide consistent replies. However, it is limited by the amount of processing it can perform, and the number of clients whose messages it can service. One mitigation for this problem is the use of server farms – groups of cooperating servers - with many servers working together to meet DVE needs.

The transition from single server to server farm introduces a host of challenges: server selection, load balancing, state transition, and reliable maintenance of state spread across multiple servers to name a few.

Some architectures such as the Butterfly Grid [29] propose middleware to make the division between servers transparent to the DVE author. Ploss et al. propose a middleware called Real-Time Framework (RTF) allowing multi-server solutions to be implemented several different ways. In [111] they describe an implementation of *Quake 3* using state replication between servers in a server farm to improve scalability.

Partitioning work in multi-server solutions is one of the key problems to solve to enable server farms. Most solutions use properties of the DVE geometric space to divide up state ownership: a single server – possibly with a backup server – is responsible for authoritative ownership of state in a given region. Each client's state is authoritatively managed by the region they are in. Clients receive events either directly or indirectly from their own region, and the regions near them. Each server handles processing and message distribution in one or more regions.

There are a number of different implementations of this approach in literature. Nguyen et al. [46] describe a load migration technique based upon a uniform grid subdivision of the DVE environment, and dynamic transfer of responsibility for individual grids based upon server loads. Lee and Lee's multi-server environment [83] proposes dividing the DVE geometric space into a uniform grid, with each server owning many cells of the grid. When a server's load gets above a certain threshold, it negotiates with an eligible server among a set of candidate neighbors to balance load with, transferring ownership of some cells.

Lui and Chan [89] provide an analytical view of load partitioning, proving exhaustive evaluation is NP-complete. They describe a parallel partitioning algorithm comprised of three sub-algorithms which can be used to partition large virtual worlds. The initial partition of the world is expensive, but subsequent partition updates are two to three orders of magnitude less costly.

Morillo et al. [102] propose a CPU-based quality metric as optimization criteria for partitioning server loads. They indicate that experience quality for clients degrades nonlinearly as servers approach saturation, and that the metrics used by most partitioning algorithms do not reflect this fact. The authors implemented a DVE ostensibly based upon DIS and HLA standards, and evaluated its behavior with synthetic loads, comparing its performance to the Adaptive Region Partitioning Technique. They found their system does a better job of minimizing per-server workloads than other comparable systems.

2.2.2.2 Trusted third party

Trusted third party solutions are those which rely upon significant DVE workload assistance outside of the primary server set and the client pool. These solutions afford the external agents – often proxies – a trusted status in the DVE. However, the role of these trusted third parties still falls short of that enjoyed by the main DVE servers.

Aggarwal et al. [3] propose a system with additional proxy servers between DVE clients and servers. The proxies help with fan-out of messages, and connect to servers on behalf of the clients. This results in the proxies potentially having shadow copies of state from all active DVE servers. However, it keeps clients from having to connect to different servers as the clients move around, and as workload is repartitioned.

Bauer, Rooney, and Scotton propose development of Booster Boxes [11] to provide improvements in quality of service for DVE participants analogous to those provided by web cache solutions such as Akamai [4]. They suggest these Booster Boxes can have a pluggable architecture, allowing a variety of game providers to support their game protocols.

Quax et al. propose Alvic-NG [114]. The system defines several independent roles for the main servers used in the DVE. In addition, it implements a proxy server layer, which funnels all client connections to the main DVE servers. The proxies are allowed to cache and replicate state, and in some cases, to resolve DVE state change transactions. They are responsible for maintaining appropriate region manager connections on behalf of clients as those clients move around the DVE, and the DVE simulation space is repartitioned.

2.2.2.3 Client-oriented

Most recent DVE scalability papers propose solutions based around hierarchical architectures, or symmetric peer-to-peer architectures with clever ways of choosing sets of peers to work together, and information scoping to reduce the events and sheer traffic which must be processed by each peer.

HLA [40] and DIS [69] [70] are older U.S. government-sponsored DVE standards efforts which detail event generation, subscription models and event contents in excruciating detail.

SimMud [79] provides one of the most convincing peer-to-peer DVEs, based upon Pastry [117] and Scribe [30], and is one of the few scalability solutions to address some aspects of security implications in their design.

ATLAS [84] is a hybrid (client-server + peer-to-peer) system which uses central servers for authoritative state change commitment, and a pub-sub broadcast model where participants are responsible for propagating their current state to all subscribers.

Peer-to-Peer DVEs which rely upon virtual “servers” to be chosen out of the pool of peers are well represented. Anthes et al. [7] propose specializing some peers as “domain servers” to help manage overall topology and fan-out event notifications. Mediator [50] separates peers into a variety of specialized roles, formalizing responsibilities of each role for message propagation, computation, and state storage. Zone models [72] describes a variety of roles (and participants assigned to those roles) for every DVE region (zone).

Several papers focus on creation of mesh overlay networks for assigning DVE simulation state ownership and in some cases propagating events. VAST [66] divides simulation responsibility by dynamic regions calculated based on Cartesian position of each participant, and structuring the participants in an overlay mesh network matching a Voronoi diagram. Varvello and Diot’s Delaunay triangulation approach [128] is similar in spirit, but uses Delaunay triangulation rather than Voronoi graphs to create the mesh and assign state ownership. AtoZ [137] divides maintenance responsibilities based upon participant location, but uses time to reach each position from a participant's current position, rather than pure Cartesian distance to partition ownership. Solipsis [76] formalizes requirements and guarantees of an overlay mesh in rigorous mathematical terms. [90] describes

mechanisms for mesh creation and event propagation. Colyseus [18] uses regions of interest and a pub-sub model to limit scope of state update propagation.

One property shared by most of the peer-to-peer DVEs - with the possible exception of SimMud - is a profound vulnerability to attack. Many rely upon neighbors to propagate messages without any way of verifying message propagation. DVE state variable is stored and updated by a small set of nodes - often a single node - at best relying on "random" choice of that node to provide security. Validation of simulation state correctness and state changes is not addressed, implying attackers can make any sort of assertion about themselves they please.

Several DVE security mechanisms have been separately proposed, some of which can address some of these problems. The next section details relevant literature.

2.3 DVE security

DVE architectures range from client-server to pure peer-to-peer. Client-server DVEs offer the strongest security, as all important state transitions can be verified and safely stored on the server.

In the most secure client-server DVE all state is authoritatively stored on a single central server. The server accepts client input directly, processing that input based upon the server's local state and time. The server has total control over how the DVE state is updated, and can take into account any factors deemed relevant. Clients are given only the data required to render their current view.

Even in this relatively secure scenario, there are a large number of vulnerabilities, such as timestamp cheats, packet modification, botting, and so on. In most DVEs these weaknesses are vectors which enable cheating. Several cheat taxonomies have been proposed to help classify these vulnerabilities.

Yan and Randell propose a cheat taxonomy in [134], culminating in a grid relating the categories of cheats to type of vulnerability exploited, damage category for the attack, and the required parties to initiate the attack successfully.

Ki et al. [77] propose another taxonomy, covering similar attacks to those suggested by Yan and Randell. However, their taxonomy is organized into layers: client, server, network, and environment hacks. In addition, they describe tools and mitigations available today for many of the attack categories.

A pure client-server solution may set the standard in terms of expected security, but we hope to improve on overall performance by adopting other architectures allowing load to be more broadly distributed. Some distribution of work is required even for server-based solutions: A solution where a single server renders and transmits 60 four megapixel frames per second per client will not be scalable. A more efficient approach is to allow each client to offload portions of the work from the server. A typical solution has client software which accepts user input, performs preliminary validation and coalescing, then propagates summaries of state changes to the server. The client renders state locally based on its snapshot of global state and its position. The server still verifies the most important state updates, and acts as a conduit to pass state changes between relevant participants. This model works best for kilobits of data per client per second, rather than gigabits. This model is – to the best of the author's knowledge – that implemented by virtually all commercial DVEs.

Unfortunately, the moment portions of validation and rendering are delegated to clients, cheating becomes significantly easier. A variety of different cheating techniques have been implemented to attack games [9], including:

- **Aimbots**, where a local modification to the DVE client automatically calculates optimal firing solutions and executes them on behalf of the user.
- **Wall hacks**, where walls are made transparent to allow the local client to see other players who are intended to be hidden from that client.
- **Map hacks**, where the local client obtains information about the map they are not entitled to.

- **Speed and movement hacks**, where through a variety of mechanisms, a client enables their avatar to move more quickly and in ways not intended by the game publisher.

Some security systems have been proposed which help to mitigate cheating and general unintended behavior in network games.

ORTS [28] is a security approach for client-server RTS games. It requires all important state be stored and updated on the server. It calculates the minimum necessary distribution of state information, passing clients only that information required to render their view.

Fung [56] proposes detecting movement cheats such as speed-hacks by having a trusted third party (TTP) - the DVE server - validate legality of each move reported by participants, rather than simply accepting those moves without examination.

Another way to detect cheats is by profiling participant behavior, and noting deviations from expected or typical behavior. Central validation of the input stream for DVEs [120] allows detection of player skill augmentation cheats, such as aimbots. RET [36] is a scheme for profiling a participants' activities, then observing an activity stream and asserting whether or not it is produced by the same participant. Chen et al. [38] analyzed *Quake 2* traces to show that automated players (bots) can be differentiated from human players 98% of the time with a 12 minute trace. This methodology can be used to help combat the use of bots in MMOGs, for example those illegally used to gather resources in DVEs such as *World of Warcraft*.

Laurens et al. [82] profiles aggregate player behavior to categorize a player as cheating vs. non cheating. They show an example where wall-hack cheats are detected by examining the fraction of time a client aims at an occluded opponent. Players with wall-hacks can see opponents who should be occluded by walls, and aim at those opponents more often than non-hacked clients. In a similar vein, DeLap et al. [41] propose a scheme for performing runtime validation of transactions in games, triggering validation whenever a player's performance falls outside of typical or expected bands. For example, an avatar which earned a disproportionate amount of gold per unit time would have their transaction histories audited using this scheme.

Aggarwal et al. [3] took a step away from server DVE security by empowering proxies to take on some of the security load. Their authoritative proxies can cache state and validate some client state changes on behalf of the central server.

Some DVE security solutions are intended to work with peer communities. They mimic some of the properties of client-server DVEs by electing participants into privileged roles, and trusting those participants to behave correctly.

The Public Server approach [33] allows MMO participants to set up their own game servers. Centrally issued certificates protect critical state - such as distribution of valuable virtual property - to prevent replication cheats. Additional calculations are performed to provide evidence that player owned servers are obeying the DVE rules, rather than distributing disproportionate amounts of virtual resources. While promising, this architecture has scalability issues in terms of the PKI used as evidence and proof of ownership.

Kabus et al. [75] present a variety of mechanisms for detecting cheating, such as electing auditors to verify event streams provided by participants, and using trusted computing bases to protect data streams and executable code.

Some solutions can work in peer-to-peer environments as well as hybrid and client-server environments.

Chambers et al. propose a round-based bit commitment protocol [34]. *Warcraft III* [19] adversaries periodically send hashes of their moves to each other, then exchange full move logs after game completion. When the full logs are received, each player can validate that the sequence of moves provided corresponds to the hashes received during play, and that the movement sequences are valid.

Ferretti and Roccetti [52] describe a way for participants to detect a specific class of time-based cheating - running the game and wall clocks at different speeds - by including game and wall-clock timestamps in update messages, allowing recipients to verify rate of time change.

FreeMMG [32] uses consensus to detect cheating, with a quorum of participants evaluating each state change request, requiring unanimous agreement to commit the change.

A significant body of peer-to-peer DVE security research is focused on ensuring fair ordering and disclosure of events between participants. Asynchronous Synchronization [12] [13] is a lockstep protocol which prevents "look-ahead" cheats by requiring all participants to commit to their next action before seeing adversaries' choices. The authors give an example of using this protocol to prevent intentional exploitation by time manipulation, for example suppressing updates. This helps prevent intentional induction of the "dead man shooting" behavior described by Mauve in [91].

Ghost [123] improves Asynchronous Synchronization (AS), allowing it to function with less strict synchronization requirements. NEO [57] is similar to AS, but rather than committing state changes by sending hashes in one round and the state change the next, state changes are transmitted encrypted, followed later by the key to decrypt them. SEA [59] asserts that lack of full protocol specification in NEO resulted in several security holes, and proposes fixes for those holes.

Mobile Guards [101] provide replaceable digital rights management (DRM) and obfuscation-style protection modules which the DVE author frequently updates to prevent cracking. While practical and in general effective, this commits the DVE owner to an arms race they are unlikely to win [113]. In effect, this approach is a trusted computing base (TCB) approach, relying upon portions of the system which cannot be undetectably compromised.

Feng, Kaiser, and Schluessler [51] propose using an embedded coprocessor in gaming systems to validate the system has not been compromised, and suggest several attributes of game systems which can be protected, for example detecting emulated input, injecting foreign code, and so on.

The most practical game DVE cheat-prevention mechanisms are targeted at detecting memory and executable modification of specific DVEs [104], such as Valves' VAC technology and Cheating Death.

Security mechanisms continue to be proposed – indeed, as I do later in this dissertation – indicating that the battle to protect DVEs and online games from cheating and exploitation is far from over.

The next four chapters present my work related to DVE scalability and security. Chapter 3 presents an analysis of avatar movement in part of the *World of Warcraft* DVE, motivated by the need for realistic evaluation workloads. Chapter 4 builds upon this work, and asserts that one of the three architectures for network DVEs – peer-to-peer – simply is not deployable in today's Internet. Chapter 5 describes Carbon, an auditing system which can detect cheating in MMOG scenarios with significantly lower resource consumption than more general auditing solutions for decentralized DVEs. Chapter 6 presents the Pairwise Random Protocol, a simple extension to secure coin flipping which can enable adversaries to fairly determine the outcome of probabilistic events.

3 Towards a realistic DVE workload

This chapter is based upon the article *Group Movement in World of Warcraft Battlegrounds* published in the *International Journal of Advanced Media and Communications*. It provides an examination of the behavior of player avatars in a commercially deployed DVE. This research shows that assumptions about avatar movement and behavior in earlier research are not consistent with certain categories of avatar behavior and message flows in a system such as *World of Warcraft*. This indicates a need for DVE scalability research to be evaluated with realistic movement traces.

3.1 Introduction

Distributed virtual environments come in all shapes and sizes, from simple turn-based games to the more prevalent real-time three dimensional simulations. Performance and scalability evaluations of these systems are based upon assumptions about game and player avatar behavior. Often times these assumptions are based upon anecdotal experience or the traffic patterns of a game purpose-built to evaluate the proposed platform.

Choices made for evaluating systems can vary by orders of magnitude. For example, does an avatar move once per second, or continuously with updates as frequent as the render frame rate, up to 120 *Hz* on modern systems? Do avatars move independently with loose synchronization requirements, or in groups? Do they navigate using fixed features, or is their movement too complex and varied to characterize in this fashion?

Answering these questions in a convincing fashion is critical to building a case supporting a given architecture's merits. Building a DVE and showing that it *can* be used in an imagined fashion is very different from building a DVE which *will* be used. I believe that existing broadly deployed DVEs are the best source of models for evaluating DVE research results intended for DVE adoption.

World of Warcraft (*WoW*) is by far the world's most popular DVE, and fertile ground for gathering data about actual player avatar behaviors. I instrumented part of one category of the *WoW* experience – player-vs-player (PvP) battlegrounds – to focus my findings. Battlegrounds were chosen because of a combination of tractability, and applicability to DVE performance modeling.

Three common assumptions people make about DVE performance are compared to observed behaviors, all related to the way avatars move:

1. **Is a waypoint model a good fit to describe avatar movement**? *Probably not*. This research shows that waypoints cannot be easily applied to describe player avatar movement in battlegrounds. I demonstrate this by implementing and applying an appropriate existing waypoint detection model described by [127].
2. **Do avatar movement patterns result in significant hotspots**? *Yes*. There is evidence of hotspots, with 5% of visited territory accounting for 30% of all time spent in a typical battleground.
3. **Do player avatars organize into coherent groups for inter-hotspot journeys**? *Not usually*. Surprisingly, player avatars usually make significant journeys within the battlegrounds alone rather than in groups, despite clear game incentives to travel in groups.

The remainder of this chapter provides supporting evidence for these findings. *World of Warcraft* is described for those not familiar with the game, with particulars about the evaluated battleground, Arathi Basin. Next, data gathering methods are described for obtaining data on player avatar movement within battlegrounds, and accuracy and completeness of that data is discussed. Finally, data relevant to waypoint, hotspot, and group movement models for DVE player avatar movement is provided.

3.2 Relevant background

3.2.1 World of Warcraft

World of Warcraft (*WoW*) is the most popular DVE in history. With more than 12 million subscribers worldwide as of 2010 [22], *World of Warcraft* has the majority of market share for all massively multiplayer online games (62% as of 2008 [132]). This ubiquity makes *WoW* especially relevant as a DVE user study test bed. Examination of other popular DVEs in this category – such as *Aion* and *Age of Conan* – shows they support PvP battleground scenarios, making results gleaned from *World of Warcraft* likely to apply to other titles as well.

3.2.1.1 Types of experiences

WoW experiences can be divided into five main categories. Those categories, along with a sample distribution of time spent in each are: *questing (37%)*, *capital cities / trading (16%)*, *instances (25%)*, *raids (9%)*, and *battlegrounds / PvP (13%)* [124]. This chapter examines data from the last category, battlegrounds. Battlegrounds – while not the most popular activity – still have significant avatar participation, and so must be supported.

The game world requires significant resources to simulate and to communicate with DVE clients. The game scales by running many simultaneous world copies called *shards*, with each avatar belonging to exactly one shard. A typical shard has between a few hundred and a few thousand active players online at any given time [110], out of a population of tens of thousands assigned to that shard.

Questing describes activities where player avatars wander the game world individually or in small groups. The game world is large and detailed – requiring more than an hour of real time for an avatar to walk across one of its four continents - and player avatars are relatively sparse. Indeed, player avatars undertaking this activity are usually out of interaction range of one another unless they explicitly seek each other out.

The game world contains ten large cities called *capital cities*. These offer a plethora of facilities, and are densely populated relative to the rest of the world. For example, I found Dalaran, the version end-game capital in *World of Warcraft* when this data was analyzed, could have a quarter of the active population on a given shard concentrated in less than 0.1% of the world's geography. While densely populated, capital city interactions tend to be infrequent and lightweight, with most avatars sitting still and performing social or character maintenance activities. Some previous research [124] [125] calls this category of activity *trading*.

Instances and *raids* are small, self-contained adventures which groups of players play together. Just as a shard is one of many copies of the game world, an instance is one of many copies of that adventure. These adventures are shared by either a self-selected or a randomly matched set of avatars – typically five, though up to 40 are allowed in certain raid instances. Together they solve puzzles and fight powerful AI-controlled avatars. An instance or raid can be thought of as a private Player vs. Environment (PvE) experience requiring a group of players to complete. Unlike other PvE experiences such as questing, players in instances and raids tend to stay together and move in a tight group from place to place.

Battlegrounds are a special type of instance, with a PvP focus – and in fact are referenced as PvP in some prior research. Like PvE instances, there can be many identical battlegrounds active and reachable from a given world shard. A single battleground instance can be populated by player avatars from multiple shards, known collectively as a battle group. Battlegrounds are characterized by scenarios which reward PvP, usually to achieve an objective or dominate a resource. Battlegrounds have intense continuous activity with between 20 and 240 mutually interacting participants (depending upon the battleground) split into two opposing sides called factions. For comparison, download traffic I measured in a *capital city* with more than 250 player avatars was 40 Kbps for a given client, while download traffic in the Wintergrasp battleground with 200 player avatars often reaches 250 Kbps sustained for a given client, and can have peak download speeds well over 500 Kbps.

Battlegrounds, with their high traffic requirements, interaction, and movement characteristics make an ideal test environment for DVE research related to player avatar movement. I chose the Arathi Basin battleground to measure these behaviors. While it is not the largest battleground, it is big enough to be interesting, and small enough to be tractable for measurement and analysis.

3.2.2 The Arathi Basin battleground

Battlegrounds are organized around inter-faction – Alliance and Horde – competition. *World of Warcraft* version 3 has six different battlegrounds. Arathi Basin is a 30-person battleground where teams compete for control of five stationary flags. Gaining control of a flag requires a team member to use the flag without interruption for ten seconds, and to prevent any enemy team members from using the flag for an additional minute. Each team receives points every few seconds based on the number of flags they control. The first team to reach 1600 points wins the battle. Both teams are rewarded, though winners receive better rewards, incenting each team to participate and to win.

The battleground environment is approximately 600 yards by 600 yards in size, with flags evenly spaced around the center of the map as shown by the circled huts in Figure 1. The circled houses next to the crests at the corners of the battleground are the starting point for each faction, alliance at the northwest, and horde at the southeast. In terms of movement, some terrain slows down avatars, or is impassable. For example, water slows most avatars down to approximately four yard per second, a quarter of normal mounted movement speed. Most cliffs and steep hills cannot be traversed, and falling off them can injure or kill an avatar. The view from the lumber mill plateau is shown in Figure 2.

Figure 1: Arathi Basin map

Figure 2: Arathi Basin from the lumber mill plateau

Traversing the map requires about one minute mounted, or two minutes on foot, assuming no enemy engagements. Avatars typically must be within either melee range (5 yards) or ranged combat range (usually 30 yards) to interact. I call the latter distance an *interaction interval*. The nearest flags are between 170 and 250 yards from each other in a straight line, several interaction intervals apart. In other words, players halfway between two flags can interact with each other, but not with players at either flag.

Players are rewarded for controlling flags and for killing avatars from the enemy faction. When an avatar is killed, it is turned to a ghost and teleported to the graveyard near the closest controlled flag, or to the faction base if no flags are controlled. Every 30 seconds all ghosts at a graveyard are resurrected, and granted full health and mana.

Arathi Basin battles are usually less than a half hour long, involving 15 players on each side at any given time, with some turnover in participants. Real life and network problems force some players to drop out, and they are usually replaced by others waiting to battle. As a result, a given player avatar may be in a battleground for as long as the entire match, or as little as a few seconds.

3.2.3 Avatar behavior and traffic classification

Little research has been done on avatar movement patterns.

[110] provides an examination of *World of Warcraft* shard populations using *WoW's* built-in extensibility. Pittman and GauthierDickey found that the workloads used in simulations to evaluate DVE infrastructure were unrealistic. User sessions in *WoW* are on average less than half an hour, but can reach 24 hours. Peak populations on a shard are typically five times their minimum population.

[99] describes a mechanism for detecting automated cheats called "bots" in *World of Warcraft* by using waypoints to characterize avatar movement. They found that many cheats use a scripted form of automation where the avatar follows the same path repeatedly. They provide a waypoint extraction algorithm and verify that it works to discriminate between player and automated avatars.

Their waypoint extraction algorithm acts upon a series of avatar movement traces. They apply the Douglas-Peucker line simplification algorithm [45] to reduce the traces to simpler lines with fewer vertices. They search for clusters of vertices, and label these as waypoints, as they are endpoints for many paths. Mitterhofer et al. [99] found that scripted bot movement tended to replay the same movement paths, and even with jitter resulted in movement that closely followed detected waypoints.

This approach is useful for simplifying player avatar movement as well as bot movements. Later I describe my efforts to apply the waypoint detection algorithm to characterize player avatar movement.

[95] analyzes avatar movement in the Arathi Basin battleground. It makes several assertions about the existence of waypoints and hotspots, and suggests that group movement is not prevalent. This article builds upon the author's earlier research, quantifying and qualifying issues with waypoint detection, and providing precise definitions and metrics for analyzing group movement.

Some avatar movement models use a hotspot model to describe places avatars are likely to move towards and congregate. [127] describes an algorithm for automatically detecting landmarks, and a method for predicting movement between landmarks. The authors divide the virtual space into a regular grid. They count the number of visits each avatar makes to each cell in the grid, and compute the weighted entropy for the distribution of player visits. Cells are designated as landmarks based on their entropy, prioritized from highest to lowest entropy. Once a cell has been chosen as a landmark, its eight neighbor cells in the grid are omitted from landmark consideration, even if one or more of them have the next highest entropy value.

Evaluation of proposed DVE systems often uses a synthetic workload based on previous research, or on a model generated by the evaluators. For example, [80] compares three different categories of DVE infrastructure using a synthetic workload based upon an average session time of 100 minutes. Avatars in his evaluation are simulated using a combination group and waypoint model, where groups of simulated avatars agree on a next point to visit, and move there together. Several other frameworks [89] [90] [102] [118] assume movement and arrival / departure properties of participants without any obvious experimental basis. The assumptions appear reasonable in the abstract, but are without firm experimental grounding. I believe using a model based on actual DVE participant behavior provides better insight into actual system performance under load, and is a better basis for comparison between proposed solutions.

Player avatar migration patterns are of critical importance for evaluating geometric routing schemes. For example, VAST [66] [10] and Delaunay triangulation [128] organize themselves based upon the position of player avatars, and their overhead and efficacy depend heavily upon the density, distribution, and dynamics of those avatars. Likewise, region-based DVE architectures [50] [74] [133] organize clients by their avatar location, and are affected by the frequency of avatar transitions between regions, and any tendency of avatars to cluster in hotspots.

Understanding the movement of avatars is important for correctly evaluating region-based and geometric-mesh based DVE schemes. The remainder of this chapter provides information about avatar behavior in a real-world DVE which can be used to inform future DVE framework evaluations.

3.3 Methodology

My goal is to capture all movement events for a set of battleground sessions to ground the evaluation of waypoint and hotspot presence, and of group movement. This requires an exhaustive trace of all movement during the battleground session. Unfortunately, *WoW* clients only receive avatar movement data which is immediately relevant to them. In practical terms, this means movement data for avatars which are within avatar visual range – approximately 250 yards – and which are not blocked by large obstructions such as cliffs.

Each *WoW* client transmits its own avatar movement updates to the server, which the server redistributes to other clients. Updates are absolute positions, consisting of a client identifier, three-dimensional Cartesian position, facing information, and additional information I do not decode. The server does not send position information for avatars the client cannot perceive, such as very distant or stealthed (invisible) avatars.

Experimentation confirmed two well-placed observers receive movement updates for most of the Arathi Basin map. Moving observer avatars into these positions takes between 1 and 2 minutes from the start of a battle, depending upon enemy activity. Battles in my sample set ranged from 4 to 23 minutes in duration.

I used Microsoft Network Monitor 3.3 to capture network traffic, and FRAPS [15] to capture video from a game client's rendered view. FRAPS videos enabled the game client view to be replayed to answer questions about activity in the game associated with specific times in the network traces.

The movement data I captured is correct, but incomplete. The two leading causes of missing data are an observer being out of position, and stealthed non-observer avatars. Observers started out of position, missing some data from the start of the game. Once in position, observers were sometimes attacked and killed, moving them out of position for a minute or more. Avatar death results in the slain avatar's ghost being teleported to the nearest faction-owned graveyard. Resurrection introduces on average a 15 second delay, and returning to post takes another half minute to two minutes. I used observers with stealth capabilities to reduce the chance of detection, and presumably of being targeted by the enemy. My observers' positions in the map are marked with white X's in Figure 3. One was at the north edge of the lumber mill plateau to the West, the other on top of a waterfall at the south end of the mine valley to the East.

Figure 3 - Placement of observers in Arathi Basin

My observers avoided combat, effectively filling two of the 15 slots in the Alliance team with non-contributors. This biased the results of the battles, but not significantly. My sample set has a good mix

of battle results, with Alliance winning nearly half the observed games, in one case by a score of 1600-0. I was able to capture battles with scores ranging from 1600-1590 (the closest a battle can be) to 1600-0, the most disparate possible final score, with a good mix of Alliance and Horde victories (6 and 7 respectively).

Battles are referred to here by the difference in score between the winning and losing team, rather than the overall game score. Thus, the battle which ended with a score of 1600-1590 is battle 10, and the battle which ended with a score of 1600-880 is battle 720.

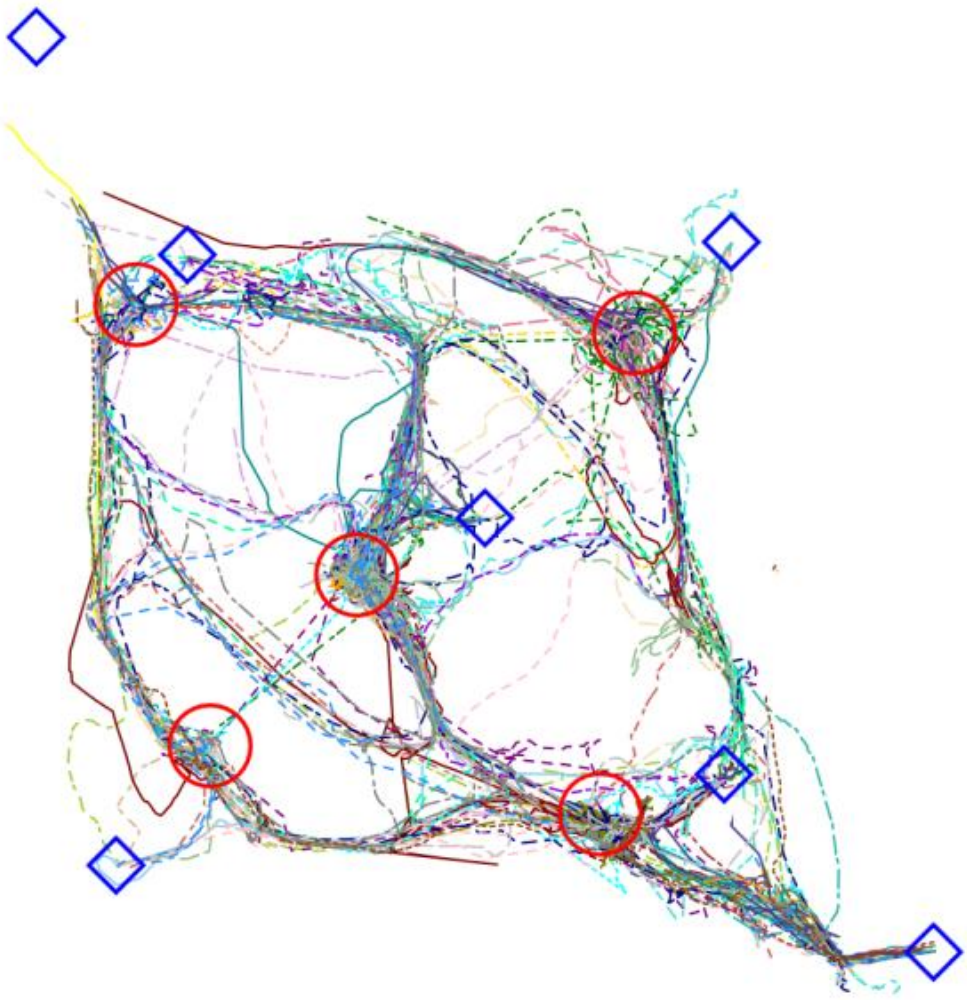

Figure 4: Battle 980 movement paths

3.4 Analysis

I captured approximately 20 traces of Arathi Basin battles, and retained 13 where my observers were mostly at their assigned posts. I am a long-time *WoW* player with more than 5,000 hours of play time, and a dozen characters in four realms, and have spent hundreds of hours in battlegrounds. I found the recorded battles consistent with my overall battleground experience. Player turnover, participation, and the flow of avatars and objectives within the battles were within the range of "normal" experiences I have witnessed in the many other battles I participated in.

13 battles provides a wealth of data for analysis. These traces comprise more than sixty hours of individual avatar movements, with 392 unique avatars in 456 avatar sessions. More details are presented in section 3.4.1.

The data captured can be summarized in a variety of useful forms. For example, the rendered movement traces for battle 980 is shown in Figure 4. I analyzed captured data to verify its correctness, and to provide information for others to evaluate suitability of avatar movement models used for evaluating DVEs. The three main phenomena I wanted to investigate were: appropriateness of waypoint models for guiding movement, existence of hotspots for hotspot-based movement models, and grouping of avatars for movement. Definitions related to each of these goals are provided below.

1. **Waypoints**. Waypoints – if they exist – are fixed virtual navigation markers used through all battleground instances. I expected flags and graveyards to be strong candidates for waypoints for movement models, along with geographical choke-points.
2. **Hotspots**. Hotspots are situational gathering points in a map, where a disproportionate number of avatars spend time during a given battle. The map has natural hotspots in the form of avatar starting locations and flags. I was curious if other hotspots would show up, and if so, if these hotspots were consistent across battles.
3. **Grouping**. Logic dictates there should be significant grouping in movement. All avatars for each team begin at the same point (their faction base) and are released simultaneously. For avatars who die – typically every avatar several times per battle – resurrection is synchronized, with all waiting ghosts at each graveyard resurrected every 30 seconds. Battle dynamics incent avatars to group to maintain numeric superiority.

Before describing my findings, it is worth presenting the characteristics of both avatar participation, and the completeness of the captured data.

3.4.1 Avatar participation characteristics

I had a series of qualitative questions. First, I wanted to provide an estimate of turnover in the battleground population. In other words, were there joiners and leavers? If so, how long was a typical session? Also, quantifying missing data is important for framing completeness of the results.

Table 1 summarizes this information for each of the 13 battles analyzed, excluding the two observers. “Lost by” shows the score difference between the winning and losing teams. “Avatars” shows the number of unique avatars recorded during the battle. The battleground allows in a maximum of 30 simultaneously present avatars, 28 factoring out the observers, but departures can be replaced. “Duration” gives the total time of each battle in seconds, from when avatars are released from their base to when one team wins and the battle concludes. “Average play” gives the percentage of the total battleground duration an average participant played. Since many enemy avatars were not observed until the observers were in position – up to two minutes after the match started - this number is biased downwards. “Average recorded” shows the percentage of avatar participation time successfully recorded for that battle. Average recorded was calculated by summing the total seconds played by all avatars, and subtracting out gaps in the traces for each avatar.

Lost By (points)	Avatars	Dur (s)	Avg Play	Avg Rec.
10	36	1423	72%	81%
300	38	1296	67%	75%
420	46	1208	52%	62%
720	36	1015	69%	75%
870	36	957	63%	62%
950	33	671	71%	69%
960	37	951	69%	76%
980	36	891	70%	78%
1050	33	885	79%	78%
1180	37	658	61%	60%
1370	36	765	66%	65%
1490	32	583	78%	83%
1600	20	266	76%	82%
AVERAGE	35	890	69%	73%

Table 1: Avatar participation summary

The data set includes a good sampling of battle scores, ranging from the largest to smallest possible difference, with an average difference of 910. The average battle had 35 unique participants, each present for an average of 69% of the battle. Participant turnover was on average 25% during the course of a battle. 392 unique avatars were recorded. Movement and position information for avatars was successfully recorded 73% of their participation time. As mentioned earlier, gaps were caused primarily by avatars becoming invisible and therefore undetectable, and by observers being killed and temporarily out of range of some avatars.

I do not believe the gaps in avatar data are significant in terms of overall analysis. The goal of this work is to evaluate avatar movement behaviors. Most avatar movement – as supported by the statistics – is observed, even ignoring instrumentation issues such as observers out of position. There is good fidelity information on the start of one faction in each battle – the side the observers belonged too, as well as data for the middle and termination of battles from both factions.

The remainder of this section describes relevance of waypoint, hotspot, and grouping models to DVE player avatar movement.

3.4.2 Waypoints

Waypoints are fixed points in the environment used for navigation, specifically resulting in identical paths for avatars navigating using the same set of waypoints. Drawn graphically, the path for an avatar following waypoints would resemble a series of overlapped line segments passing near or through the waypoints visited in the sequence they are visited. Each waypoint would reflect where the path describing an avatar's movement changes direction. Waypoint navigation is typically accomplished with straight movement paths, but nothing prevents waypoint navigation from being described with other constructs such as Bezier curves. Whatever the construct, waypoint navigation would provide identical path navigation for a variety of different avatar journeys following the same set of waypoints. This uniformity of paths traversed is the artifact we search for in our analysis.

Intuitively, strong waypoint candidates for the Arathi Basin battleground are graveyards, flags, and points on the optimal (non-water, non-cliff) routes between graveyards and flags. While such

waypoints could be manually identified, there is no guarantee that designation would be correct. Instead, a more general analytical approach was used, as outlined by Mitterhofer et al. in [99].

Using this algorithm, waypoints are extracted from movement traces by a combination of two strategies: *k-means*$^{++}$ *cluster analysis* and *path simplification*.

Avatar movement traces consist of a series of points which can be joined together to form a sequence of lines. If avatars are using waypoints for navigation, there should be clusters of line endpoints at the waypoints where some avatars change direction: even if navigation paths are described with curves, then ignoring operator error, the path followed between two waypoints would be identical, and would have identical decomposition into straight line segments. If no avatars change direction at a given waypoint, it is not actually a waypoint.

Assuming waypoints are used for navigation, their presence can be obscured by human error in controlling movement. Small diversions as avatar controllers delay turning or move in a non-optimal path can confound automatic detection of waypoints.

To mitigate this variation - and as suggested by Mitterhofer et al. in [99] – traces were simplified using the Douglas-Peucker line simplification algorithm [45], with a tolerance of up to one interaction interval error in line segmentation. This recursive algorithm reduces line complexity by approximating complex polylines with simpler, albeit less accurate lines with fewer points. The algorithm was invented nearly forty years ago, and is still considered one of the best general line simplification algorithms.

Lines are simplified as follows: Given an input path $P_{1,k}$ of points $\{p_1, \ldots, p_k\}$ and an error tolerance E expressed in the same scalar system as the points:

1. Find the point p_j in $\{p_1, \ldots, p_k\}$ which lies furthest from the line (p_1, p_k), and call its distance from the line $d_{j,1,k}$.
2. If $(d_{j,1,k} \leq E)$ the line is simplified to two points, $\{p_1, p_k\}$ and processing is complete. Otherwise:
3. The line is simplified to the union of the results of applying the Douglas-Peucker algorithm to $P_{1,j}$, and $P_{j,k}$. This line will consist of at LEAST the points $\{p_1, p_j, p_k\}$.

Line simplification significantly reduces the number of points required to represent an avatar's movement, making it easier to find clusters of points. Indeed, I successfully used this technique with an error tolerance of 30 yards (one interaction interval) to simplify the movement trace shown in Figure 5 to that in Figure 6. I then applied k-means^{++} cluster analysis to try to cluster the majority of remaining points into waypoints.

The k-means^{++} algorithm chooses clusters as follows:

First, choose a set of k seed cluster centers. A random point is selected from the data set, and then for $(k - 1)$ iterations:

1. For each data point not in the cluster center set, calculate a probability of being selected equal to the square of the distance from the candidate to the closest cluster center set member.
2. Choose a random point to add to the cluster center set from the candidates, weighted by the probability of selection.

Next, execute normal k-means cluster analysis using this set of k seed cluster centers as the starting point.

Even with dramatic simplification, k-means^{++} cluster analysis was unable to identify a consistent set of waypoints to describe avatar movement. Successive runs on the same data set identified potential waypoints, but many of these diverged widely between different runs, depending upon the initial waypoints selected by the algorithm. Reviewing the point cloud for both simplified and original movement traces battleground revealed the cause. While there are clear asymmetries in point density for describing movement traces, no reasonable number of clusters can encapsulate the majority of

simplified points, even when dramatic simplification tolerances are used (e.g. allowing errors greater than an interaction interval).

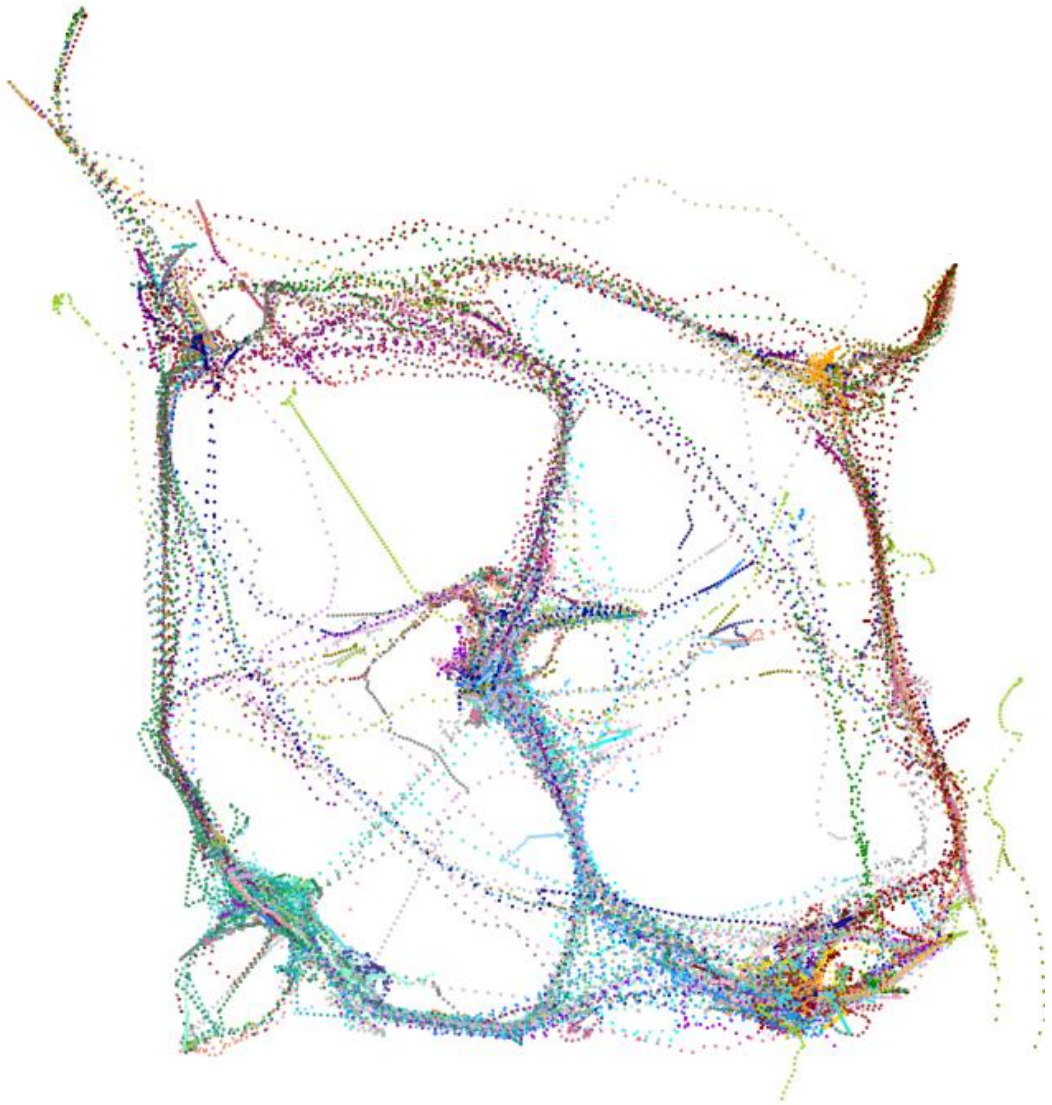

Figure 5: Point cloud for battle 1590

Figure 6: Simplified point cloud (+/- 30 yard tolerance)

I conclude that waypoints are not appropriate for characterizing general player avatar movement in World of Warcraft battlegrounds. This conclusion supports Mitterhofer et al.'s implication [99] that waypoints are not a good fit for characterizing typical player-controlled avatar movement.

3.4.3 Hotspots

This section describes efforts to characterize avatar movement patterns using hotspots. Hotspots are portions of the battleground where avatars spend the most time. Mathematically, hotspots are determined by dividing the map into equal sized cells, summing the number of seconds spent by each avatar in each cell, and designating the k cells with the highest totals as the k hotspots. Cells in the 8-neighbors of an existing hotspot are precluded from being designated as hotspots, to prevent runs of adjacent hotspots. I implemented the technique for hotspot detection proposed and applied by Thawonmas et al. in [127].

I expected hotspots at each of the five flags because they are game objectives with most activity occurring at them, and at the seven graveyards because participants die many times each battle and await resurrection in them. Most of the first 12 hotspots contained either a flag or graveyard. However, the order of "hotness" of these points of interest varied from battle to battle. In some battles particular

graveyards and flags were never included in the top hotspots. Non-graveyard, non-objective hotspots *were* encountered, reflecting battles where significant concentrations of avatar activity occurred away from flags and graveyards. Although such hotspots were seen in most battles, no common characteristic was found for them. While specific non-objective hotspots were present in more than one battle, none were present in the same location for a majority of battles.

Potential hotspots are shown in dark grey or black in the player time-density maps in Figure 7, with the five most active hotspots labeled 1 through 5. The trace on the left shows battle 420 with hotspots at the mine, stables, lumber mill, and blacksmith flags. The fifth hotspot is on the path from the farm graveyard to the mine flag. The trace on the right shows battle 1600 with hotspots near each of the flags.

.

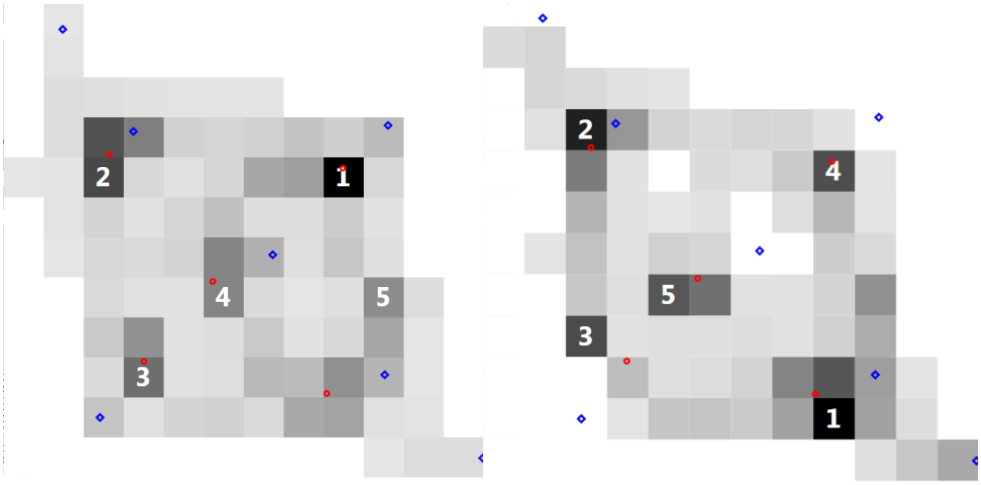

Figure 7: Activity density maps from two different battles

As forecast, hotspots were usually located where there was heavy contention over a flag, or a battle which migrated from flags towards arriving combatants. Based on this, an adaptive hotspot-based model taking into account current populations at hotspots should be useful for gaining insight into avatar movement within battlegrounds, possibly allowing a generative movement model.

One weakness in this hotspot model is the tendency for cells which would otherwise have been hotspots to be excluded because of their abutment against a higher-ranked cell. This biased calculations, as it prevented hotspots from being correctly located.

To compensate for this, I extend the cell-based hotspot model using a centroid which need not be aligned on a hotspot boundary. Centroids are circles centered on a hotspot, but without the restriction to align at the granularity of cell boundaries. An example showing centroids overlaid on the original hotspot map is shown in Figure 8, alongside the density map used to calculate centroids at a resolution of 20 yards.

To calculate centroids with a radius of one interaction interval (a diameter of 60 yards) perform normal hotspot calculation with cell width one third of that diameter, e.g. 20 yards. Note that using a finer grid for hotspot squares may result in different hotspots than result from a coarser grid, or even different ordering for similar hotspots. In the example shown above, results are comparable between the two resolutions of density maps.

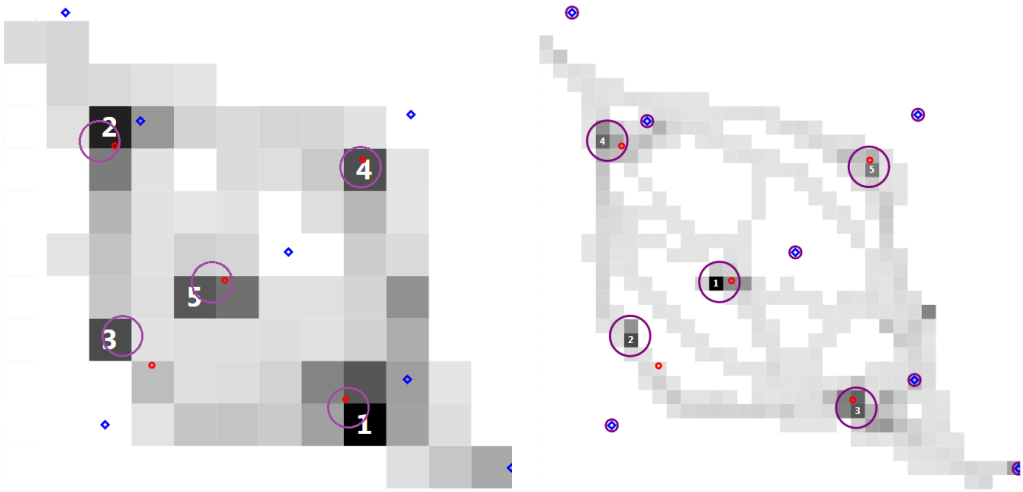

Figure 8: 60-yard and 20-yard density maps with centroids

For each 9-square centered on a given hotspot, calculate the center of mass for avatar movement in that 9-square, weighted by avatar dwell time in each cell. The resulting hotspot centroid does a better job of covering the *actual* hotspot (defined by unit area time-spent density) than a simple cell-based designation of the same radius, and better even than the finer grained density map it is calculated from.

Table 2 shows a comparison of hotspot player dwell density (the metric used to identify hotspots) for battle 1600 using a 60-yard fixed grid, a 20-yard fixed grid, and a centroid with a 60-yard diameter. The 60-yard grid is used to calculate the dwell density based upon the grid as shown in the left side of Figure 8. The 20-yard grid column calculates dwell density using the 9-square centered on the hotspot closest to the original 60-yard grid hotspot, to cover the same area. The centroid column calculates density based upon the area covered by the large purple circle closest to each hotspot, based upon the 20-yard resolution 9-square. The higher-resolution 20-yard grid provides on average 25% improvement in density over the 60-yard grid. The centroid approach, however, provides on average 56% improvement over the 60-yard grid, significantly better than the 20-yard grid alone. I interpret this higher density to mean the centroid-based hotspot is doing a better job of describing the actual hotspot location than arbitrarily aligned grids.

The reason for this improvement is simple: the centroid is positioned over the actual center of mass, rather than defined by a square with arbitrary borders that happens to contain some areas of high traffic.

These revised hotspot definitions and other points of interest enabled deeper analysis of avatar movement. The next section evaluates the tendency of avatars to move between points of interest – including these hotspots - in groups.

HotSpot	60-Yard Grid	20-Yard Grid	Centroid
Farm (1)	0.200	0.311	0.379
Stables (2)	0.172	0.197	0.250
Lumber Mill (3)	0.134	0.135	0.167
Mine (4)	0.132	0.132	0.159
Blacksmith (5)	0.123	0.191	0.245

Table 2: Avatar dwell time density, seconds per square yard

3.4.4 Group movement

I define group movement as the coordinated movement of multiple avatars between points of interest within an interaction interval of each other. Points of interest in my model include avatar spawn points – as many trips start at spawn points – and hotspots, typically flags and the sites of heavy battles.

There is strong incentive for players to group within a battleground: a lone combatant has very little chance of defeating multiple enemies. A fight between parties from the two factions is almost always won by the larger force. Success in combat implies success at controlling flags, which in turn leads to battleground victory and greater in-game rewards.

Regardless of the benefits of group movement, two factors provide a disincentive for forming and maintaining such groups. First, the difficulty of coordinating group formation and maintenance using default communication channels such as text chat. Second, the conflict between group and individual goals: without an accepted group leader, these often diverge.

Even when a group is well coordinated (for example via a voice over IP solution such as Ventrilo [54]) and has an acknowledged leader, maintaining group coherence is difficult. If a group member is slain, they become a ghost, and must resurrect and travel back to the body of the group. Barring enemy interference, this can take up to two minutes, half the battle length in some cases. In the meantime, the group typically continues towards its next objective, with subsequent deaths splintering the group further.

Is group movement a reasonable model to apply to describe the majority of travel between points of interest? [95] proposes a general grouping metric called “affinity.” I extend this metric with a more precise definition.

Leveraging the work on hotspot identification above, I define the concept of a *journey*. A journey is a trip between points of interest which are at least two interaction intervals (60 yards) apart. Taking a circuitous route between two nearby hotspots does not typically constitute a journey, but travelling between two hotspots whose closest edges are 60 yards apart – without passing through a third point of interest – does. Two avatars which move between the same two points of interest maintaining a distance of no more than one interaction interval for at least half of their trip are considered as having affinity for that journey.

The seven graveyards / spawn points were selected as fixed points of interest, and another seven centroids (including hotspots which happen to overlap spawn points) were chosen by the algorithm described in the previous section as per-battleground instance points of interest. The centroids included at least four of the five flags in every battle, though with the centroid at slightly different locations. Most battles had at least one non-flag, non-graveyard centroid.

As mentioned previously, this analysis is based upon trace data containing approximately 73% of overall avatar sessions in the analyzed time period. Movement segments are divided into four categories:

1. **Inter-centroid movement**, journeys between different centroids. Some are degenerate cases (return to the same centroid) or too short to be eligible for these calculations, but still fall in this category.
2. **Centroid-anchored movement**. These segments typically departed from a centroid, then terminated before reaching another centroid. An avatar which spawns at a graveyard and is killed before it can reach another centroid would be in this category.
3. **Intra-centroid movement**, movement segments within a graveyard or centroid which do not leave that centroid.
4. **Extra-centroid movement**. These segments are usually artifacts of missing data. The battleground start is a special case. If the starting bases are not picked as centroids by automatic hotspot detection, then the initial path of any avatars killed or departing the battle before reaching a centroid will be in this category.

Table 3 provides a summary of the percentage of recorded time which falls into each of the four movement categories. Of the more than 60 avatar-hours of traces obtained, approximately 32% is inter-centroid journey data. This gross figure includes inter-centroid journeys which are too short to qualify for further analysis: the actual percentage of eligible inter-centroid journey traces is 40% of the inter-centroid journey traces, or 14% of overall captured traces. This resulted in 951 candidate journeys for analysis.

Lost By (points)	Recorded (s)	Inter Centroid	Centroid Anchored	Intra Centroid	Extra Centroid
10	30,786	9,922	4,499	15,764	601
300	25,870	7,299	6,367	10,577	1,627
420	19,470	6,169	3,751	9,024	526
720	20,056	6,596	3,214	9,397	849
870	14,314	4,432	3,166	5,935	781
950	11,100	3,304	3,021	4,333	442
960	19,531	5,343	3,582	8,641	1,965
980	18,245	6,642	3,037	7,599	967
1050	19,000	6,743	4,227	6,890	1,140
1180	10,409	2,498	1,861	5,045	1,005
1330	12,285	3,717	3,265	4,221	1,082
1390	12,431	4,760	2,866	4,584	221
1600	3,442	1,278	719	1,415	30
AVERAGE	16,688	5,285	3,352	7,187	864

Table 3: Avatar movement trace categories

Affinity analysis run across all battleground traces are summarized in Table 4. The data in this table is for journeys completed between centroids, and affinity is evaluated in a binary fashion: either the entire journey is considered as having affinity between two or more avatars, or none of it is.

Lost By (points)	Journey (s)	Affinity (s)	Journey (count)	Affinity (count)	% Affinity (s)	% Affinity (count)
10	5,769	735	174	41	12.7%	23.6%
300	3,296	754	96	24	22.9%	25.0%
420	2,940	405	95	19	13.8%	20.0%
720	2,250	331	86	20	14.7%	23.3%
870	1,205	52	34	4	4.3%	11.8%
950	1,154	64	44	7	5.5%	15.9%
960	2,279	519	76	24	22.8%	31.6%
980	3,250	679	107	30	20.9%	28.0%
1050	2,663	748	76	26	28.1%	34.2%
1180	821	239	28	10	29.0%	35.7%
1330	1,594	274	57	13	17.2%	22.8%
1390	1,882	304	57	13	16.1%	22.8%
1600	703	94	21	3	13.4%	14.3%
AVERAGE	2,293	400	73	18	17.4%	24.6%

Table 4: Avatar journey affinity

The table shows that journeys with affinity make up only a small fraction of overall journeys, with the average seconds spent in journeys per battle being 2,293, and the average fraction of that time spent in journeys with other avatars being 17.4%. This percentage varies from 4.3% to 29.0% across the battles evaluated. I examined features of battles such as distribution and weight of hotspots, number of journeys evaluated, and battle length, but was unable to find any correlation between the affinity fraction and other characteristics of the battle. This may make an interesting direction for future research.

Some journeys are terminated or *truncated* before reaching their intended goal, for example because of the avatar encountering a battle and being killed. Any pair of truncated journeys in which avatars are within an interaction interval of each other for at least 8 seconds (the time required to travel two interaction intervals) are considered as having affinity. This ensures two avatars passing each other in different directions are not considered as having affinity. However, two avatars which travel a short distance then stop within an interaction interval of each other *will* be considered as having affinity.

Inclusion of truncated journeys in the evaluation of journey affinity more than doubled the number of seconds considered as spent in journeys, while only raising the percentage of journeys time spent in affinity from 17% to 22%. The disparity between these two metrics I attribute primarily to battles fought outside of the top hotspot centroids, where two or more avatars were fighting each other for at least 8 seconds, and therefore had affinity for the purpose of this analysis. I believe this result shows that the decision to use full journeys for affinity analysis rather than partial affinity is a reasonable simplification.

These results confirm and quantify earlier findings. Despite incentives, the majority of journeys in battlegrounds are made alone, rather than in a group. Only 17% of inter-centroid journeys by time are made in a group. The number looks slightly better when viewed in terms of journey counts rather than journey durations, but is still less than 25% affinity. Interestingly, the disparity between affinity journey seconds and journey count percentages – and the relatively higher truncated journey affinity rate - indicate that longer journeys are less likely to be made in groups than shorter ones.

3.5 Conclusions

Distributed virtual environment architectures have been evaluated using a variety of criteria and loads. These criteria look reasonable, but do not necessarily correspond to the sorts of behaviors seen in broadly deployed DVEs.

I examined validity of three common assumptions using actual behaviors in *World of Warcraft*: waypoints to describe avatar navigation; the presence of hotspots within the movement traces; and finally, movement of avatars in groups.

More than 60 hours of avatar movement traces were gathered from the Arathi Basin battleground in World of Warcraft, consisting of 456 avatar sessions with 392 unique avatars across 13 battles.

I used line simplification and k++ means cluster analysis to show that these avatar traces cannot be characterized using waypoints. Waypoints would dictate high-density clusters of endpoints after line simplification, but no such artifacts were found.

I examined avatar movement data and found evidence of hotspots in the data. These hotspots can be useful for adhoc avatar navigation models, but cannot be used as a basis for waypoint navigation, as they vary from battle to battle: they are an artifact of the battle, rather than of the battleground. I proposed a centroid-based method for describing avatar hotspots. The centroid method provides better player dwell density than grid-based hotspots of approximately the same area, even when those grids have a resolution three times higher than the diameter of the centroid-based hotspot.

Finally, I examined avatar movement between points of interest and hotspots, and found that most avatars do not make journeys in groups, despite clear incentives to do so. More than three quarters of avatar journeys between points of interest in battlegrounds are made alone, even though avatars spawn in groups.

Two of the discounted mechanisms, waypoints and group movement, are common in previous distributed virtual environment research evaluations. Their lack of applicability to measured data suggests that a new way of generating movement traces is needed. Gathering real-world traces and using them for testing is of course very valuable. However, gathering traces by hand is expensive, and may not generalize well.

I suggest research be done on ways of synthesizing avatar movements more consistent with real data and observed behaviors. Rather than relying upon global algorithms such as group movement between waypoints, generation may benefit from building and simulating simple avatar agent state machines. For example, a given avatar could be attracted to some criteria and repelled from others. The presence of other avatars – both friendly and hostile – should be considered, as should natural terrain, and the impediment to movement or protection it affords. Such a model should also take into accounts different classes of avatar sophistication, and various methods of avatar control, such as the contrast between high-precision control and low-interaction movement in a series of straight line segments. Combining these factors in a supportable way, and comparing the results to analysis of real movement traces could provide a useful synthetic workload generator.

4 Near-term infeasibility of P2P DVEs

This chapter is based upon the similarly named paper presented at NetGames 2010, and published in the conference proceedings. Chapter 3 dealt with the question of characterizing workload in a subset of a deployed DVE, *World of Warcraft*. This chapter broadens the data capture and parsing initiated to encompass all five main areas of *World of Warcraft* activity.

Practical experience and literature show that *World of Warcraft's* traffic model is the same general flavor as that of most contemporary MMORPGs and first person shooters: clients send their updates to the server, which then propagates them to other relevant clients. *WoW* and most MMORPG models are less network resource-hungry than typical first-person shooters (FPSs), which are characterized by higher message frequency, and therefore more overall network traffic. Also, some of the FPS games propagate all updates to all players, whereas the larger scale games such as MMORPGs restrict message propagation to clients likely to benefit from the update messages. If a pure peer-to-peer approach cannot work for *WoW*, then it likely will not work for other RPGs and FPSs operating on a similar scale.

4.1 Introduction

Nearly every aspect of running a reliable DVE becomes more challenging in P2P architectures. Ignoring all other issues, can existing residential network infrastructure support the bandwidth requirements of a P2P DVE?

I gathered and analyzed network traces from the world's most popular DVE, *World of Warcraft* (*WoW*). I used these traces as the basis of a workload for simulating client-server and an idealized peer-to-peer publisher / subscriber solution. I was able to characterize both absolute and relative performance of both types of messaging solutions, and found even an idealized and zero-overhead P2P solution performs too poorly to deploy.

The remainder of this chapter describes this investigation. First, I provide background relevant to *WoW* and the broadband bandwidth model I used. Next I discuss my methodology for creating workload traces, and my simulator's behavior. I analyze the gross attributes of my parsed *WoW* workloads, and apply them to both client-server and P2P pub-sub message schemes. Finally, I present conclusions and directions for further research.

4.2 Background

This section provides information on *World of Warcraft* and its main characteristics. It also gives a brief summary of the OfCom-provided residential broadband statistics [107] used in my bandwidth model.

4.2.1 World of Warcraft

World of Warcraft (*WoW*) is the most popular Massively Multiplayer Online Game in history. With more than 12 million subscribers worldwide [22], *WoW* has the majority of market share for all MMOGs: 62% as of 2008 [132].

Player avatar activities in *WoW* can be divided into five categories, as described in [124]:

1. **Questing**. The avatar interacts with the game environment and AI-controlled avatars called NPCs.
2. **Capitals**. This category is a subset of the *trading* category proposed in [124], and consists of time spent in the game's heavily populated capital cities.
3. **Battlegrounds**. Player versus player combat. I use this category for inter-player battles in zones called *Arenas* or *Battlegrounds*. There are six unique battlegrounds.

4. **Instances**. Small, scenario-driven experiences where up to five avatars work together to kill a series of powerful NPCs known as "bosses." There are 56 unique instances.
5. **Raids**. A large instance. Instead of 2 – 5 bosses, raids have between 1 and 12. They are designed for 10-40 players and may include hundreds of NPCs. There are 24 unique raids.

[110] examines *WoW* server populations using built-in *WoW* extensibility functionality. The authors found DVE simulation workloads were unrealistic. They found peak server populations are typically five times their minimum population, underscoring the need to model peak behaviors.

[95] analyzes avatar movement in the Arathi Basin battleground. It proposes a mechanism for obtaining game positions from avatar traces. This section extends that work, parsing several other types of messages, and obtaining both player and NPC messages and positions.

Evaluations of proposed DVE systems often use synthetic workloads based on previous research, or on a model generated by the evaluators. For example, [80] compares three different categories of DVE infrastructure using a synthetic workload based upon an average session time of 100 minutes. Avatars in that evaluation are simulated using a combination group and waypoint model, where groups of simulated avatars agree on a next point to visit, and move there together. Several other frameworks [89] [90] [102] [118] assume movement and arrival / departure properties of participants without any obvious experimental basis. The assumptions used in all these cases are logical, but without firm experimental grounding, and often contradicts actual measurements from real DVEs such as *WoW*.

4.2.2 Broadband speeds

My analysis examines behavior of different topologies involving residential nodes. It requires a realistic consumer broadband model.

Overall speed summaries are available from several sources, such as Akamai's quarterly *State of the Internet* report [5]. The Q4 2010 report from Akamai includes an appendix outlining average download speeds observed by their servers for a half billion unique IP addresses across 45 countries from all six significantly populated continents. Key countries listed include Japan, South Korea, Australia, Egypt, the UK, Brazil, and the United States. While peak average broadband speeds observed were significantly higher in many countries, it is the average broadband speeds we are concerned with. These ranged from 0.6 Mbps in Sudan to 13.7 Mbps in South Korea.

Unfortunately, no upload statistics are provided in the Akamai report, and large-scale measurements of upload and download speeds are difficult to find. I found only one report of significant enough scope and reputation for me to rely upon, the 2009 UK Broadband Speeds report from OfCom [107]. This report includes profiles of both upload and download speeds observed for a variety of users across the ISPs serving 91% of the UK's broadband customers.

The UK is in the middle of the pack for download speed listed in the Akamai report, with 4.3 Mbps average download speed compared to a weighted average (based on number of addresses sampled) of 4.59 Mbps download speed. This makes the UK data a good basis for simulation.

Comparing the average UK download speed listed in the 2009 OfCom report to that listed in the Q4 2010 Akamai report, I found them to be within a few percent of each other, 4.1 Mbps and 4.3 Mbps respectively. This similarity is reassuring when seeking to correlate results back to the Internet at large.

The OfCom report provides a wealth of information about both advertised and observed UK broadband speeds at a variety of times. Actual residential broadband speeds are on average significantly slower than their advertised speeds. Salient statistics are summarized below:

- OfCom divided subscriptions into three categories, based upon their advertised download speed: 2 Mbps or less, > 2 Mbps and <= 8 Mbps, and > 8 Mbps. See Table 5 for relevant statistics.
- Round-trip latency as measured to UK servers was on average below 55 ms, with peak average latencies (at high-traffic times of day) of up to approximately 110 ms.

- Connections encountered low jitter, typically less than 6 ms (9% of the 55 ms RTT latency).
- Packet loss averaged less than half a percent.

This chapter's evaluation of P2P feasibility does not model jitter or packet loss. However, it makes use of the population percentages and upload and download speeds given above, as summarized in Table 5.

Advertised Speed	Subscriber Percent	Down (Kbps)	Up (Kbps)
<= 2 Mbps	29%	1700	280
2 to 8 Mbps	57%	3900	420
> 8 Mbps	14%	9300	580

Table 5: Client node bandwidth model

The next section describes the overall approach used for analyzing *World of Warcraft* captures and making simulation traces from them.

4.3 Methodology

To determine whether a P2P message propagation scheme could propagate the messages required for an MMOG to operate, two questions had to be answered about message propagation. First, which avatars receive which messages? Second, which network attributes are most relevant to the simulation, and which can be safely ignored?

MMOGs typically strive to propagate messages only to clients affected by the message. For example, if player X's avatar is moving and within visual range of player Y's avatar, then X's DVE instance needs a copy of Y's movement messages.

Most DVE messages reflect a state change initiated by and centered on a single avatar. Some messages have a broader effect – such as a command to render rain in a large zone – but these commands are relatively rare. Messages are received by a superset of the avatars affected by that message. For example, any avatars within visual range of a spell being cast need to know about the spell, even if they are not facing the caster. A *WoW* avatars' visibility range is usually between 250 and 500 yards. Interaction range is shorter, about 5 yards for melee and trading, or 30 yards for most ranged interactions.

For an ideal P2P messaging scheme, I make the following simplifying assumptions, to err on the side of NOT ruling out P2P feasibility.

1. All clients can communicate with each other.
2. Clients have perfect knowledge of avatars (and their associated clients) entering and leaving their interaction and visual range.
3. There is no overhead associated with setting up client network links or with tracking remote nodes beyond what is normally propagated by the DVE.
4. There is no additional security overhead, nor other costs usually associated with P2P message propagation.
5. There is no packet loss or jitter.
6. The underlying transport is TCP. The only simulation impact of this choice is message framing overhead, and the TCP window enforcing a limit on outstanding bytes for a given connection.

7. The radius of event propagation is 250 yards, close to the minimum for *World of Warcraft*.
8. Broadcast and unattributed messages are not included in the computations. If a message from our traces cannot be attributed to a specific peer, it is excluded from the simulation. This reduces overall message load.

Most of these assumptions reduce traffic. If the network cannot support this simplified system and its requirements, then it certainly cannot support a more realistic (and complex) solution.

The P2P pub-sub message propagation scheme being simulated here implements a pure P2P model. Each message is transmitted by the originating client directly to each subscriber requiring a message copy.

4.3.1 World of Warcraft network attributes

WoW uses a proprietary client-server protocol to communicate between DVE nodes. Most data between a *WoW* client and its server is exchanged over a single TCP connection. Most traffic results from client node actions. The acting client node sends a message to the server, which makes appropriate DVE state changes, and then forwards the message or related updates to any other affected nodes.

Each client DVE instance does an advisory check of local user inputs and state change requests before submitting them to the server. If the server accepts a request as legal based on its world state, the message (or resulting update) is propagated to clients to whom it may be relevant, sometimes including the client who submitted the message. If the action was rejected, the server replies to the requester, and that requesting client reverts any relevant state. In most cases a DVE state change is propagated using an identical message payload to all clients which can perceive that change.

WoW network messages are small, 36 bytes on average [124], with some as small as 4 bytes. During active periods a client can send more than 10 messages per second. Conversely, in idle periods the client may go minutes between sending messages. Messages consist of an encrypted header and an optional unencrypted body. Messages can be combined within a single TCP packet, and if needed can span TCP packet boundaries. During periods of intense local activity clients may receive bundles of many messages several MTUs in size.

Most messages include a globally unique avatar identifier (GUID) identifying the party initiating and / or affected by the message. Other common fields include X, Y, and Z position in yards, facing information, and game time.

Since message headers are encrypted, I devised heuristics to identify a few of the more complex (easier to identify) and useful message types. Like *WoW*, the simulation propagates messages based on virtual world proximity, so position messages are the highest priority. After parsing a candidate list of move messages, incorrectly parsed messages are identified by comparing their contents against median values and reverted. Correctly parsed sequences of messages have similar values in many fields (such as position and game time), whereas incorrectly parsed messages are more likely to have divergent values.

Using this strategy, 25% to 42% of the bytes in each capture were successfully parsed. Using knowledge of packet format, between 55% and 93% of the bytes in each capture were parsed or attributed.

Any data which could not be successfully parsed or attributed was excluded from use in simulation traces. This exclusion reduces overall bandwidth and message requirements for my simulations, compared to actual message loads. This in turn means that at worst our simulations would show that deployed peer-to-peer DVEs are feasible when in fact they are not.

This is discussed further in section 4.4.

4.3.2 Simulation trace generation

The simulation traces are based upon processed *WoW* network captures. Each capture is between 5 and 120 minutes long. In some cases the captures were from a single client instance, while in others a pair of clients was used to extend the region of visual information.

Before parsing messages, TCP payloads of the trace are divided into *blobs*, aggregating adjacent packet payloads which are clearly part of the same blob. For example, two MTU-sized packets followed by a third smaller packet would all be combined into a single blob.

The blobs were iterated over, attempting to parse message types from most to least restrictive. After parsing each message type, heuristics were used to revert clearly erroneous parsing. For example, if the GUID did not meet structural expectations, or did not appear frequently enough in the parsed trace, the message was considered invalid and reverted.

Once all messages were parsed and filtered, remaining unparsed blobs were searched for known GUIDs. Whenever a GUID was found, it was presumed to be at a common offset for GUIDs within messages, and the attributed message was set as starting an appropriate number of bytes before the GUID. I called this process *attribution*. The message was assumed to continue to the end of the blob, or to the start of the next attribution, whichever was shorter.

The logic behind this strategy is as follows: identifying a party involved in a propagated transaction almost always means the message originated within interaction distance (30 yards) of that party. Since interaction between avatars is typically limited to an area much smaller than – and fully contained within – the maximum visual AoI, a slight variance in the propagation center should not significantly affect the receiving audience. This assumption allows assignment of the majority of traffic not successfully parsed. While not strictly correct, this provides a good approximation of message origin and propagation. As data later shows, using this strategy the parser successfully parsed or attributed more than 90% of message bytes.

Earlier I described the five categories of play within *WoW*. I captured data in all five categories to gain a better understanding of difference between them, and to ensure I evaluated the network topologies in all styles of play.

Fidelity of the captures and their overall utility varied by category:

1. **Questing**. I captured and parsed solo play with both highest-level characters and characters advancing in the game. Solo PvE usually involves significant travel in sparse areas. Captures are limited to the solo player's AoI, and are low fidelity from a simulation perspective because of a lack of context. Fortunately, these are also the least interesting to simulate because of their low network traffic.
2. **Capitals**. Capitals are heavily populated, with a high degree of transience. Most avatars are quiescent, performing simple maintenance activities or waiting for game events. Traces were captured in Dalaran, the most popular capital city, with two avatars strategically placed to capture most of the city's messages. Dalaran's population ranges from tens to hundreds of players, with turnover as high as thousands of unique avatars per hour. Dalaran captures have good fidelity.
3. **Instances**. In instances a player avatar explicitly works with up to four other avatars to complete a bounded objective such as killing a series of NPCs. Players stay within interaction distance of each other, so the capture contains all relevant information.
4. **Raids**. A raid is a special type of instance for 10 to 40 avatars rather than 5. Other than scale, they are similar to instances, and have good capture fidelity.
5. **PvP battlegrounds**. Data was captured in two of the games' six battlegrounds, Arathi Basin and Wintergrasp. Arathi Basin captures have high fidelity, as the two instrumented nodes were sufficient to cover most of the battleground. Wintergrasp is larger, and data was only captured at the main hotspots. This is sufficient to characterize the highest activity areas centers, but

misses smaller battles located away from the hotspots. Despite limitations of the Wintergrasp captures, they are interesting because of their scale, often more than a hundred participants, with the majority mutually interacting.

Parsing and processing this data as described above resulted in a series of traces appropriate for simulation. The next section describes the simulator's basis. Summary statistics for data capture and parsing can be found in section 4.4.

4.3.3 Simulator

I performed a cursory evaluation of existing simulators, but concluded that none would give the combination of simulation resolution, performance, and ease of modification that I desired. I had the following requirements:

1. **Scalability.** The simulator must scale to up to 1,000 simultaneous active nodes to support capital city scenarios.
2. **Latency.** The simulation must model inter-node latency and packet latency in a realistic fashion.
3. **Throughput**. The simulation must accurately model bandwidth constraints in the presence of competing streams, and upload and download constrained transfer. It should also model TCP-style windowing, whose impact on throughput is negatively correlated with inter-node latency.
4. **Overhead**. Packet headers are a significant source of traffic in DVEs, which have frequent, small messages. The overhead of protocol headers and encapsulation need to be accurately modeled.
5. **Performance**. The simulator needs to be able to operate faster than real-time against typical simulation traces in order to allow several runs against the more than 60 hours of simulation traces.
6. **Leavers and joiners**. The simulator must be able to realistically handle leavers and joiners, as these are a significant presence in DVEs such as *World of Warcraft*.

I considered using the WiDS [87] and NS-2 simulators. For logistical reasons I was unable to use NS-2. The WiDS simulator, while it has good scalability properties, lacked most of the features I felt were important for simulation, most importantly the correct handling of throughput, especially in cases of link contention.

I developed a medium-fidelity packet-level network simulator which provides a reasonable approximation of Internet behavior for my scenarios.

- The simulator assumes TCP / IP as the message transport, and uses a packet-based model for propagating data.
- Nodes are assigned "network positions" on a 2D plane. Latency is the Cartesian distance between two nodes' network positions.
- Messages have framing overhead (TCP and IP). Large messages are divided into packets based upon MTU.
- The simulator accounts for first- and last- hop bandwidth limitations, but as a simplifying assumption treats capacity between these two points in the cloud as infinite.

The simulator propagates messages at three levels: The TCP/IP stack, the network adapter (NIC), and the last-hop router. Each is outlined in more detail below.

4.3.3.1 TCP / IP level

The TCP/IP level mirrors behavior of relevant portions of a typical Windows TCP/IP stack. It accepts messages for transmission, applies TCP windowing, aggregates messages, and forwards them to the

NIC queue to simulate outbound transit behavior. I use a TCP Window size of 17 KB, the default TCP Window size for Windows XP.

Message transmission through the TCP/IP stack is simulated in the following sequence:

1. Locally originated messages are entered into a src→dest keyed outbound queue.
2. Every millisecond the state of the src→dest connection is checked to see how many packets have left the IP stack at src, but have not been fully received at dest. If the payload bytes exceed the TCP window, the messages remain buffered.
3. Otherwise, the TCP/IP stack transfers enough messages to the outbound NIC level to fill the TCP window.

4.3.3.2 NIC level

The NIC level of the simulation handles inbound and outbound packets on a first-come, first-serve basis. Packet transmission is simulated in millisecond time slices, subject to node bandwidth limits. When packets finish transit, they are removed from the NIC-level queue and passed to the next layer, either the TCP/IP queue for inbound packets, or the remote "last-hop router" queue for outbound messages.

When a packet is transferred to a "last-hop router" queue, it is time-stamped with the current simulation time plus any inter-node latency. In other words, the packet is marked for future delivery.

4.3.3.3 Last-hop router

Internet latency and inbound NIC bandwidth are both modeled using the "last-hop router" before the destination node.

The last-hop router has a priority queue with zero or more packets, ordered by their projected arrival time. Once a packet's arrival time passes, the last-hop router moves it from the priority queue to a host transmission queue. The packets are moved into the inbound NIC at the destination host on a FIFO basis, using the time slice bandwidth budget. Whenever a packet finishes transmission, it is passed into the receiving node's NIC queue, where it then bubbles up the stack as a received *WoW* message, and is processed.

4.3.4 Topology choices and metrics

DVE topologies can be divided into three broad categories:

1. **Pure client-server**. The server has "perfect" information about the location of every avatar, and uses this as the basis for propagating messages. This model is used by *WoW*.
2. **Region-based pub-sub**. The game space is divided into regions. A subset of nodes propagate messages for each region. This region also includes most hybrid super-node solutions.
3. **Avatar-based "P2P" pub-sub**. Avatars subscribe to event streams for other avatars within a certain distance of themselves, i.e. within their Area of Interest (AoI).

The primary goal of these schemes is to ensure every node gets necessary messages in a timely fashion. "Timely" depends upon the DVE. *WoW* defines three latency thresholds: <= 300 ms latency is *good*, <= 600 ms is *adequate*, and > 600 ms is *poor*. Other DVEs have documented acceptable latency tolerance as low as 200 ms and as high as 1250 ms. Lower latency is almost always better.

4.3.5 Simulation characteristics

Three sets of simulations were run, identified by a message aggregation interval. For each set, all simulation traces were executed on both a pure client-server topology, and a P2P pub-sub topology.

For each simulation, the following attributes were set:

1. Weighted random assignment of client node bandwidth, based upon OfCom statistics, as shown in Table 5.
2. Random assignment of 2D Cartesian network coordinates to model inter-node latency, with a maximum inter-node latency of 90 ms and an average inter-node latency of 45 ms.
3. For client-server simulations,
    a. Symmetric server bandwidth (if a server is present) of 1 Gbps.
    b. Server is assigned the center of the latency modeling space to mimic "optimal network placement" of the server.
4. AoI of 250 yards. All nodes subscribed to events from any other node within their AoI. If events originate at a non-node (e.g. an NPC), they are propagated by the closest node.
5. TCP window size of 17 KB.

The three simulation sets had a single parameter varied between them, the message aggregation interval. For each interval, all messages to be transmitted over a given TCP connection were bundled up and an aggregated message – divided into multiple IP packets if required - transmitted. The first run had an aggregation interval of 0, meaning messages were immediately processed without aggregation. Subsequent simulation sets used aggregation intervals of 1ms and of 50ms. Surprisingly the results between 1ms and 50ms were not substantially different from each other, so only the difference between the 0 ms and 1 ms aggregation intervals are described here.

For both client-server models and P2P models, perfect knowledge of subscribers is assumed, based upon the actual position of avatars at the message origination time.

The next section describes simulation results. These include the percentage of messages parsed and attributed for each of the *WoW* activity categories.

4.3.6 Simulator validation

Since the simulator was written from scratch, I performed a variety of tests to verify it behaved as expected. Steps taken to validate each desired behavior are as follows:

1. Verified that leaves and joiners were correctly handled. Packets in flight were dropped upon sender or receiver departure. Joiners received all new messages in their AoI radius.
2. Ensured bandwidth constraints were respected in the single connection case. I tested sender constrained, receive constrained, and equal bandwidth allocations.
3. Tested bandwidth constraints and bandwidth sharing over multiple flows. Single sender multiple receiver, single receiver multiple sender, and multiple sender multiple receiver were all tested.
4. Verified TCP fragmentation / reassembly, windowing, and latency (and its impact on windowing) were correctly respected. Latency included "cloud" latency modeled by Cartesian distance, and transmission and receipt delays caused by congestion, packet queuing, and finite transmission speeds at sender and receiver.
5. For cases where sender message aggregation was implemented, ensured messages on a given TCP flow were correctly aggregated.
6. Ensured message transmission decisions were correctly made as nodes moved into and out of message origin AoI, for sender, receiver, and both moving.
7. Verified instrumentation reported the correct bit transmission and receipt quantities per unit time.

This is not an exhaustive list of validation performed, but is intended to convey the spirit of testing done.

4.4 Results

Table 6 shows the percentage of payload data successfully parsed and attributed by play category. “Parsed” is the data fraction parsed into known message types. “Attributed” is the data fraction attributed to a specific origination GUID, but not parsed. “Discarded” is any remaining bytes, since their propagation AoI could not be determined.

Data which could not be attributed was discarded to prevent negative simulation bias. Each byte of data omitted is a byte that would have needed to be transmitted at least once in a perfect simulation of the DVE. Omitting this data results in a workload which is a subset of the original DVE workload. Like our other simplifying assumptions, this policy errs on the side of saying peer-to-peer DVEs are possible.

The largest proportion of unrecognized, unattributed data in traces came from questing. Nearly half of all questing data captured could not be attributed or parsed, and so was discarded. The most action-intensive categories, Raiding and PVP, had less than 10% of data discarded, and so provide the most realistic simulation traces.

Table 7 shows the aggregate duration of captured data, in minutes.

	Parsed	**Attributed**	**Discarded**
Questing	25%	30%	45%
Capital	37%	39%	24%
Instance	31%	58%	11%
Raid	26%	66%	8%
PVP	42%	51%	7%

Table 6: Parse success percentages

Category	**Total Minutes**	**Average Per Capture**
Questing	1339.07	44.64
Capital	204.54	29.22
Instance	1136.80	27.73
Raid	703.14	31.96
PVP	310.55	16.34

Table 7: Minutes of data captured

While the traces for Capital may look small from a time perspective (just over 3 hours of data captured), it is high fidelity data from a relatively uniformly behaved area (per time of day) and contains hundreds of thousands of messages.

4.4.1 Simulation results

The simulation turned up two significant results:

1. **P2P pub-sub is not feasible today**. For most scenarios nodes occasionally fell behind, in some cases receiving messages minutes after their origination. This is unacceptable for DVEs such as *World of Warcraft*.

2. **Message Aggregation reduces P2P pub-sub latency**. Common wisdom and published speculation says delaying message transmission is always harmful in latency-sensitive applications. The simulation found otherwise, especially in bandwidth-constrained scenarios.

Each of these results is described in further detail below.

4.4.1.1 Feasibility of P2P pub-sub

My goal was to determine whether peer-to-peer MMOGs are possible with today's infrastructure. My conclusion is that they are not. This section provides supporting evidence for that conclusion.

To evaluate this hypothesis, a subset of recorded *World of Warcraft* data was run through a network simulator. Analysis indicates the *WoW* protocol is quite efficient. Upload and download bandwidth consumption for typical clients in an older version of the protocol was found to be on average 2.1 Kbps upload, and 6.9 Kbps download [126]. [124] analyzed a later version of *World of Warcraft* and found that upload bandwidth was usually 10 Kbps or less, and average download bandwidth varied from less than 10 Kbps for questing, to approximately 50 Kbps for raiding. My observations on *WoW* version 3 are consistent with these latter numbers.

As mentioned earlier, simulations were run across traces captured from each of the five categories of play, with a message propagation radius of 250 yards, at the lower end of my AoI estimate for *WoW*. Simulations used traces consisting of data whose origin could be attributed to a specific node. Some data was excluded, which makes it possible that the simulation could erroneously indicate a *WoW*-like P2P DVE is possible. However, this was not the case in our simulation.

Running the simulation against a client-server architecture, results were consistent with my first-hand experience in terms of message latency. These results are summarized in Table 8. The first two columns present the average upload and download bandwidth consumed by each client by category. The third column shows overall latency in seconds, e.g. 0.046 seconds which is 46 ms average. The final column shows the average peak latency. In other words, if you calculate the greatest latency during each player session in a given category, then average those values together, that is the "peak" value shown in this column.

Average inter-node latency was set to 45 ms. The server is placed in the center of the "network latency" map, making node-server latency on average 22.5 ms.

	Up (Kbps)	**Down (Kbps)**	**Average Lat. (S)**	**Peak Lat. (S)**
Questing	2.7	5.5	0.036	0.079
Capital	1.8	84.3	0.047	0.269
Instance	4.1	35.7	0.046	0.342
Raid	5.2	80.4	0.051	0.658
PVP	3.2	48.8	0.049	0.150
Average	3.4	51.0	0.046	0.299

Table 8: Average client-server simulation results

The same simulation was run with a P2P pub-sub scheme. *WoW's* message propagation scheme adapts perfectly to a pub-sub peer-to-peer topology: message origination is attributed as part of the original trace generation. Message delivery is made to all nodes within the AoI of the message origin. This is an optimal message pattern for pure peer-to-peer pub-sub, and would require no modification of the *WoW* protocol to run without security. Note that adapting the protocol to run P2P would require additional security overhead, though that overhead is excluded from this simulation.

P2P pub-sub did markedly worse than client-server topology in terms of bandwidth and latency, as shown in Table 9. In the pub-sub model, peers are responsible for uploading not only their own messages, but for modeling and transmitting any messages on behalf of NPCs (AI avatars) closest to them. This increases peer upload bandwidth requirements, but reduces download bandwidth requirements as the closest node is synthesizing some NPC messages rather than receiving them from a server.

Comparing the two tables, the average upload bandwidth required per publisher is a factor of 20 greater in P2P than client-server overall, and in some cases, a factor of nearly 100. Download consumption has dropped slightly, while latencies have risen to unacceptable levels for all scenarios except *Questing*, which requires very little message propagation. In addition to significant latency, peak upload bandwidth requirements exceeded capacity in almost all cases, for at least part of each trace. Invariably this was caused by a large number of avatars interacting near each other. In raids this could consist of 25 player avatars and 100 NPCs simultaneously moving and using special abilities. In PVP battlegrounds the peaks were especially obvious in Wintergrasp when a large percentage of attackers and defenders fought at one of the objective fortresses' walls. In the capital, it was simply a function of how many users were online at a given time. The more avatars online, the more who were actively moving (assuming a fixed percent of idle nodes), and the greater the number of event subscribers.

WoW is – in my opinion – a well-designed game in terms of network efficiency. The experience has been carefully crafted to require minimal per-node resources. Even with low per-node network requirements, the simple need to exchange information frequently between adjacent nodes makes it infeasible to make a responsive pure P2P MMOG, at least with today's bandwidth limitations. There is hope for the future. First, broadband speeds are increasing as new technologies are rolled out with ever-increasing speeds. Second, there are mitigation strategies such as avatar behavior modeling [109] which can be explored to reduce communication frequency and size requirements.

	Up (Kbps)	**Down (Kbps)**	**Average Lat. (S)**	**Peak Lat. (S)**
Questing	5.8	4.4	0.033	0.121
Capital	167.2	79.2	4.467	119.697
Instance	38.6	32.7	1.163	12.763
Raid	102.2	77.6	9.070	92.708
PVP	83.2	48.3	1.420	24.711
Average	79.4	48.4	3.231	50.000

Table 9: Average P2P pub-sub simulation results

Unfortunately, it is difficult to say precisely when P2P DVEs can be expected to be feasible. *World of Warcraft* is now an old game, mediocre in terms of responsiveness and simulation variety (which requires bandwidth to provide). Modernizing these behaviors would require additional bandwidth, and so I expect new games will require additional network resources. Peak requirements will vary DVE by DVE. Broadband overall will improve in speed, but there will always be pockets of subscribers whose bandwidth improves more slowly than others.

The change which would make P2P DVEs most likely to be deployable is an increase in average upload bandwidth. Even an increase to double the current average upload speed would enable DVEs like *WoW* to operate P2P in all but the most strenuous circumstances. However, it is unclear whether a new title created with *WoW's* modest bandwidth demands would be deemed acceptable by consumers looking for a new game.

I explored other strategies to reduce the latency impact of low upload bandwidth. The most promising one is message aggregation. The first simulation run did no aggregation, transmitting each message as soon as it was entered into the simulation, incurring protocol overheads for every message, such as IPv4 and TCP headers. Given the small size of transmitted message payloads – 36 bytes on average [124] – this results in a message transmission overhead of more than 100% on average. I suspected that message aggregation, in addition to reducing bandwidth requirements, might have other benefits.

4.4.1.2 Message aggregation for latency reduction

Latency is an important metric for online game experience quality. Previous work suggests game messages should be sent as soon as they are available to minimize latency. The simulations described in this section found the opposite, that message aggregation actually lowered latency.

The simulator models basic TCP windowing, but not common TCP packet aggregation strategies such as Nagle's algorithm. Instead, I implemented a simple time-based aggregation with a fixed aggregation window. These results will apply to TCP with appropriate aggregation algorithms, and to "custom" TCP-like IP protocols as proposed elsewhere.

Unfortunately, the simulation trace generation algorithm has some inaccuracy in assigning message initiation times. Simulation traces were created by instrumenting a small number of clients (one or two), recording client-server data streams, and extrapolating message transmission times. Avatar movement messages have game timestamps which allows their initiation time to be correctly attributed. No other message type has these timestamps, so I relied upon IP packet capture time. During busy intervals large numbers of aggregated messages were received. These aggregated messages are assigned identical or very similar event times in the simulation trace. These events should be more uniformly distributed, and so my analysis is biased towards showing benefits of aggregation which may be less prominent in more accurate traces.

Aggregating messages and transmitting aggregated messages once every millisecond resulted in significant reduction in bandwidth consumption, in peak latency, and in some cases in average latency. This was true for both the client-server case and the P2P Pub-sub case.

Earlier statistics were presented for non-aggregated client-server and pub-sub simulations. Below are two more tables showing the percentage change in attributes when aggregation is introduced.

Table 10 shows the changes to client-server traffic and latency attributes when all parties aggregate messages in 1 ms windows before transmitting them. This strategy introduces an average of 1 ms latency (0.5 client to server, and another 0.5 server to destination nodes) in end-to-end message delivery from aggregation delay. However, the aggregation allows more efficient transmission of messages from client to the server, reducing total bytes transferred. Even this small aggregation interval significantly reduces both average and typical peak bandwidth consumption.

	Up (Kbps)	**Up Peak (Kbps)**	**Down (Kbps)**	**Down Peak (Kbps)**	**Avg Lat. (S)**	**Peak Lat. (S)**
Questing	-9.0%	-20.3%	-14.3%	-26.0%	0.0%	-2.2%
Capital	-25.0%	-12.1%	-31.8%	-26.6%	0.8%	-4.3%
Instance	-17.6%	-22.8%	-20.4%	-28.4%	0.6%	-3.7%
Raid	-6.1%	-10.6%	-17.7%	-12.6%	0.2%	-3.5%
PVP	-31.7%	-16.6%	-38.6%	-34.1%	-0.3%	-3.6%
Average	-17.9%	-16.5%	-24.5%	-25.6%	0.3%	-3.4%

Table 10: Client-server, 1 ms aggregation changes

In the client server case there is a small increase in average latency (less than a percent), but typical peak latencies are reduced by several percent, meaning performance during very active intervals is improved.

Table 11 shows P2P pub-sub improvements across the board. The most dramatic improvements are seen in average and peak latency. Across all scenarios, average latency is reduced by nearly 42 percent, halving it. For the highest latency scenario – raiding – average latency drops to one fourth of its non-aggregated value, and typical peak latency is reduced to less than half its original value. While the results would still provide a poor experience during active intervals, it would be dramatically better than the non-aggregated case.

	Up (Kbps)	**Up Peak (Kbps)**	**Down (Kbps)**	**Down Peak (Kbps)**	**Avg Lat. (S)**	**Peak Lat. (S)**
Questing	-10.2%	-8.3%	-13.6%	-11.7%	2.20%	-3.6%
Capital	-6.5%	-0.7%	-7.1%	-7.2%	-22.3%	-20.7%
Instance	-28.8%	-19.4%	-29.0%	-18.0%	-60.2%	-55.5%
Raid	-17.5%	-8.0%	-18.6%	-22.5%	-52.5%	-42.5%
PVP	-34.3%	-11.6%	-35.7%	-12.8%	-76.9%	-59.4%
Average	-19.5%	-9.6%	-20.8%	-14.5%	-41.9%	-36.4%

Table 11: P2P pub-sub, 1 ms aggregation changes

One interesting question is the impact of different aggregation windows on latency and bandwith consumption. A single simulation run through the data took several CPU days to compute, and more time to analyze. As a result, I evaluated the smallest and largest aggregation windows I would consider, 1 ms and 50 ms. In client-server distribution, the 50 ms client aggregation window resulted in higher overall latencies than the 1 ms aggregation, and marginally lower bandwidth consumption. For the P2P pub-sub case, the 50 ms window resulted in a mixture of lower and higher latency than 1 ms aggregation, interplay between the extra transmission delay and the additional bandwidth savings. I believe further simulation of values between these two aggregation windows would find a "sweet spot" maximizing latency reduction and bandwidth savings. I believe this sweet spot would vary according to typical traffic patterns, which means it would be different between different DVEs, and even between major scenario types (such as the five we examined in *WoW*) within a given DVE. Nevertheless, this is an analysis DVE designers would be well-advised to perform.

Since aggregation provides benefits in both resource-rich (client-server) and resource constrained scenarios, it should be an integral part of any messaging strategy for games which are typified by small but frequent message exchanges. This approach will ease network and node burdens, and provide a better end-user experience.

Region server and super-node approaches were not evaluated in this work, although they are the middle part of the spectrum between client-server and pure P2P solutions. If practical concerns can be overcome, a region server approach is more likely to be deployable than a P2P solution.

4.5 Conclusions

Massively Multiplayer online games (MMOGs) are a popular form of distributed virtual environment. A significant body of work done over the past ten years studies the behavior of MMOGs, and proposes ways to enhance scalability and performance. However, much of this work is evaluated against idealized workloads, vastly differing from researcher to researcher, making comparison difficult.

I captured a variety of interactions within *World of Warcraft* (*WoW*), and created simulation traces from messages issued by the game, rather than a logically deduced workload. This allows creation of simulation traces consistent with a broadly deployed game, and simulation of projected network behavior for today's games. To help evaluation accuracy, typical residential broadband bandwidth characteristics were obtained and utilized.

A packet-based network simulator was implemented and used to simulate using those bandwidth models, modeling key network attributes such as inter-node latency, and TCP windowing. Bandwidth consumption and end-user latency of player nodes in client-server and P2P pub-sub message propagation schemes were compared and contrasted.

A client-server topology did a good job of delivering messages to clients quickly and with reasonable bandwidth consumption. P2P pub-sub was unable to deliver messages in a timely fashion, often saturating client links, despite several favorable simplifying assumptions. I conclude that pure P2P pub-sub is not feasible for MMOGs such as *World of Warcraft* and other similar games with current residential broadband.

Message aggregation for bandwidth reduction was investigated. Surprisingly, not only did message aggregation reduce bandwidth consumption, it improved message latency in most scenarios, and provided significant latency improvements for P2P pub-sub solutions.

5 Trusted auditing of decentralized DVEs

The previous two chapters established reasons to be skeptical about the performance and near-term deployability of peer-to-peer DVE architectures. However, these conclusions do not apply to all decentralized architectures – such as super-node models and region server models – and so there remains a need for a security solution which meets the needs of decentralized DVEs. This chapter and the next explore methods for improving decentralized DVE security.

The trusted auditing work described in this chapter was published as a technical report [96], and pre-dates the detailed measurements presented in the previous two chapters. As a result, the traffic figures and terminology may vary slightly. The traffic figures are recorded against an earlier version of *World of Warcraft*, which had slightly lower traffic requirements. These traffic differences are worth calling out, but are not significant in terms of the security solution presented in this chapter, nor its evaluation.

Decentralized DVE architectures can improve DVE scalability, but can be difficult to secure. Existing security technologies such as Practical Byzantine Fault Tolerance [31] provide good preventative distributed system security, but at a premium. Unfortunately, BFT-based systems have significant bandwidth and latency overhead, reducing their utility in DVEs.

Auditing is a lighter weight approach which retains good – although deferred – correctness guarantees and has significantly lower impact. Auditing focuses on validating prior DVE transactions to identify errors in state communication or determination. In game terminology, auditing catches cheaters. Caught cheaters are typically punished, ideally deterring future cheaters from misbehaving.

This chapter proposes Carbon, an auditing system specifically designed to catch certain categories of cheaters in decentralized DVEs. Owing to its specialization for DVEs, Carbon is lighter weight than other auditing solutions such as PeerReview [60] while retaining most relevant benefits of those systems. A key part of Carbon's solution is a trusted – rather than untrusted – auditor. This provides a compromise between scalability and security, one not onerous to DVE creators already used to maintaining centralized infrastructure.

The remainder of this chapter describes the Carbon auditing system, and illustrates its overhead through a detailed example. Section 5.1 provides an overview of DVE research relevant to security, such as DVE threat modeling, frameworks with integrated security, and stand-alone security models. Section 5.2 provides a threat model with categories based on existing cheat taxonomies. Section 5.3 outlines the Carbon auditing system, describing its main components and interfaces. Section 5.4 evaluates Carbon's attributes and performance, and compares Carbon overhead with the most prominent P2P auditing system today, PeerReview. Finally, section 5.5 presents conclusions and future work.

5.1 Related work

Distributed virtual environments have been researched for decades, with a significant body of research into P2P DVEs and DVE security emerging in the past eight years. Research related to security, correctness, and cheating is the most relevant to Carbon. In some cases this work is presented as an integral part of a decentralized DVE framework. In others, the work stands on its own, and can be applied to a variety of DVE architectures.

The remainder of this section presents prior art in DVE threat modeling, P2P DVE frameworks, and stand-alone DVE security research. Section 5.4.3 makes certain assumptions about DVE behavior and traffic patterns, and so DVE behavior is briefly discussed in section 5.1.4 for the benefit of those unfamiliar with the domain.

5.1.1 DVE threat models

Several DVE cheat taxonomies and threat models have been proposed over the last ten years. Each provides insight into the types of threats faced by deployed DVEs.

Yan and Randell [134] provide a detailed network game cheat taxonomy. Their taxonomy classifies most of the cheats encountered in network game DVEs, and includes several practical examples. Most of the threats described in section 5.2 are derived from Yan and Randell's taxonomy, though they are represented in other taxonomies as well.

Webb and Soh [131] present an interesting overview of cheating and their own taxonomy. They divide attacks into four categories, based upon their level in the application stack: Game level, application level, protocol level, and infrastructure level. Examples are provided of implemented cheats in each of these categories, as well as brief discussion of applying significant mitigations from recent research to combat those cheats. Most of the salient attacks are similar to those presented in Yan and Randell's taxonomy.

Yee et al. [136] propose a massively multiplayer online role playing game (MMORPG – a particular type of game DVE) threat model, and briefly describe existing mitigations. Their treatment – while interesting – is largely focused on human factors rather than those directly addressable by software mitigations. This makes their model useful for examining the overall milieu of publishing a game, but less so for evaluating software security.

5.1.2 P2P DVE frameworks

I am unaware of P2P DVEs and P2P DVE frameworks in broad use, but several are proposed in literature. Some frameworks include security measures to help prevent cheating. Two of the most notable frameworks which at least reference security for their solutions are SimMud and FreeMMG.

SimMud [79] proposes a DVE framework based upon traditional P2P infrastructure. It uses the Pastry [117] DHT to store and retrieve key-value pairs and organize other overlays. The Pastry-based Scribe protocol [30] provides application-layer multicast, enabling pub-sub event distribution. In terms of security, this architecture relies upon the indifference of randomly chosen parties for correct operation. State is described as key-value pairs. Authoritative state for each variable is stored at a master chosen by Pastry ID similarity to the variable key. State updates are disseminated to subscribers through Scribe trees, again constructed based on Pastry ID. A well-placed attacker can authoritatively modify state which does not belong to them, and prevent state from correctly propagating to subscribers. I believe this level of security to be insufficient in the face of motivated attackers.

FreeMMG [32] is a DVE framework built upon a combination of servers, peer specialization, and replication. Servers oversee division of the simulated world into segments, and monitor membership within each segment. Segments are collections of mutually interacting nodes, and are responsible for determining simulation state. Objects can only belong to a single segment, and all interactions within a segment are calculated by the member nodes of that segment. The primary cheat vector addressed by FreeMMG is collaboration among segment participants to subvert DVE rules. FreeMMG mitigates this cheat by requiring a certain number of participants (e.g. k) be present in each segment. However, this can be easily subverted by a cheater with sufficient resources, either with control over k nodes, or in collaboration with $(k-1)$ other cheaters. While a practical approach, this security model relies upon the presence of at least one honest node per segment, reducing its efficacy.

Neither of these frameworks fully defend against any of the threats listed in section 5.2. Both are significantly more vulnerable to collusion and inconsistency attacks than client-server architectures. Collusion allows an attacker to authoritatively control state, by locating the quorum or node responsible for processing state updates, and subverting it. Inconsistency attacks can be mounted by any node responsible for storing state. For example, a pastry node in SimMud could return random values for each key request made by an external party.

5.1.3 DVE security work

There are three main approaches for stand-alone protection of DVEs from cheating: protecting DVE software and communications, distributing state ownership to disinterested third parties, and auditing schemes. Kabus et al. [75] provide a good overview of all three.

Protecting software integrity and some forms of protecting communications is a trusted computing base (TCB) approach. Mobile Guards [101] are a recent example of this approach. DVE software is modified to integrate a trusted software component – a *mobile guard*. Portions of the DVE are encrypted with keys only available from a functional mobile guard. Likewise, communications and data access can have integrity and confidentiality guaranteed with keys either contained within or derived from the mobile guard. As long as the mobile guard is not compromised, the system can guarantee its communications byte stream is unaltered, and DVE rules are enforced as coded both locally and remotely. Attacker compromise of mobile guards is mitigated by issuing updates more frequently than the guards can be compromised. Assuming the mobile guard is not compromised, this approach provides strong security guarantees mitigating most of the threats described in section 5.2. Unfortunately, mobile guard efficacy is predicated on identifying an interval in which the mobile guard cannot be compromised, which must be sufficient for new mobile guards to be distributed. This core requirement – that you know when the TCB is compromised – is one of the issues dividing those who accept TCBs as sufficient and those who do not.

Distributing state ownership to disinterested third parties is another security technique. As mentioned in Section 5.1.2 above, P2P DVEs such as SimMud and FreeMMG use this method. It has been proposed separately as a mechanism for protecting DVEs. A variant presented in IRS [58] allows state to be owned by the concerned party, but verifies state update calculations by performing them at multiple untrusted nodes, then comparing the results. It assumes a disinterested third party has no motivation to break DVE rules in terms of state representation or updates, and that several disinterested parties are even less likely to collude for this purpose. Unfortunately this is not necessarily the case: third parties can maliciously tamper with data, whether it is relevant to them or not. For example, they can exploit their position to broker access to the state, requesting a fee from the data owner to keep the data secure. Or, the party with the greatest interest in a given piece of state can manipulate the system to ensure that control and auditing of that data falls to itself. The same argument holds true for compromising quorums of disinterested third parties, though of course compromising a quorum is usually more work than compromising a single node.

If detecting – as opposed to preventing – illegal state changes is an acceptable level of mitigation, then auditing schemes can provide good DVE security. The best example of this sort of protection in recent literature is PeerReview [60] PeerReview is an auditing system with good scalability and correctness guarantees. State changes and local transactions relevant to state calculations are stored in certified append-only local logs. Log contents are committed by peer exchange of signed log digests. Audits are performed by auditors called *witnesses*, who simulate forward from a state through a series of logged events to ensure correctness. Based on audit results and behavior, nodes are labeled as trusted (correct), suspected, or exposed (incorrect). Audit frequency guidance is provided in terms of number of witnesses and the probability of illegal activity. PeerReview offers protection against externally observable client misbehavior, external event modification, collusion below a certain threshold, Sybil attacks – since it uses RSA-key based identities – and inconsistency.

Since PeerReview is the system most similar to Carbon, a comparison of salient attributes of both systems is provided in sections 5.4.2 through 5.4.4. Carbon provides similar security guarantees – with the notable exception of message non-repudiation – but has significantly lower resource requirements in the DVE scenario. Note that this is enabled by Carbon not being as broadly applicable as PeerReview.

5.1.4 General DVE characteristics

DVE operation is largely characterized by four activities:

1. **Simulation**. Evaluating state change and events. This is typically a lightweight activity, for example accepting an input and issuing an update equation.
2. **Rendering**. Rendering the DVE perspective to present to the participant. This is usually the most significant activity in terms of memory, I/O, and processor consumption, by an order of magnitude or more.

3. **Communication**. DVE nodes exchange messages to determine state changes and to refresh shadow (non-authoritative) state copies. In a P2P DVE, this traffic is typically the traffic required to describe a state change multiplied by the number of outstanding shadow copies.
4. **Persistence**. Storing DVE data, for example saving avatar state for later retrieval. Persistence typically involves small amounts of data, infrequently written, for example a few kilobytes once per session.

Auditing should not directly affect a client node's simulation or rendering: these activities are identical at a client node with and without auditing, though additional simulation workload is introduced at auditors.

The remaining activities, communication and persistence, can be characterized by the attributes listed in Table 12.

Function	Variable
State snapshot in kilobytes	$\|S_i\|$
Average events per second	E_i
Average outgoing event bandwidth (per neighbor)	$\bar{R}(E_i)$
Average outgoing payload bandwidth (per neighbor)	$\bar{F}(E_i)$
Average shadow state copies	Q

Table 12: Sample DVE attributes

Let $|S_i|$ be size in kilobytes for a node's dynamic state description, such as avatar position and attributes, and shadow state for objects being tracked by that node. Let E_i be the number of locally initiated events per second. Let $\bar{R}(E_i)$ be the average traffic in Kbps required to describe a node's state transitions to a single neighbor. Since packets tend to be small and frequent, let $\bar{F}(E_i)$ be the payload portion of $\bar{R}(E_i)$, excluding packet framing overhead. Let Q be the average number of neighbors receiving shadow state updates for a given piece of state.

Incoming and outgoing non-audit bandwidth for a P2P DVE will each be at least $\bar{R}(E_i) \times Q$, because of shadow state updates. Suppose all clients initiate one event, resulting in a locally authoritative state change. Each client transmits Q copies of its state change, one to each shadow state subscriber. Since each state variable has Q shadow copies, this implies each client is subscribed to Q times as many state variables as those it owns, and so it will also receive Q updates.

Persistence load varies depending upon DVE architecture. In some cases dynamic state is regularly saved. In others, it is stored only between client sessions. In still others it is never persisted. The most relevant attribute for DVE persistence is the state snapshot, and so the dominant factor to examine is frequency of state snapshot persistence.

5.1.4.1 Network game DVE traffic patterns

Network game DVEs are consumer-grade DVEs intended for home use, and optimized to provide an immersive experience with limited resources.

These immersive experiences require interactive response to inputs. Many of these DVEs are fast-paced combat simulations, with real-time activity such as aiming and firing weapons, chasing opponents, and competing for resources. These applications are extremely latency sensitive, with latencies greater than 200 ms significantly degrading the experience [16] [61], and latencies of 100ms or less preferred.

These DVEs need to be able to run on most personal computers, and over most network links, which typically means functioning over dial-up connections (56 Kbps). For example, both *Quake III* and *World of Warcraft* can function over dial-up in most cases.

Consumer-grade DVEs are characterized by low packet inter-arrival times (responsiveness), and very small packets (low resource requirement). [81] found *Quake III* has typical packet sizes of 70-90 bytes, and typical inter-arrival times between 10 and 50 ms. [126] found that *World of Warcraft* sends frequent, small packets, typically with little or no payload. Typical IP packet size ranges between 50 and 70 bytes (my measurement), with 220 ms mean inter-arrival time (their measurement).

There is one notable exception to this behavior in popular DVEs: *Second Life*. This DVE is a cyberspace simulation, not a game. It emphasizes user-created content. This feature consumes significantly more bandwidth, between 10 and 1164 Kbps mean download bandwidth consumption, and between 13 and 74 Kbps mean upload bandwidth consumption [78].

5.1.4.2 Characteristic values: a World of Warcraft-style P2P game

To help put auditing overhead in perspective, I hypothesize basic requirements of a peer-to-peer version of *World of Warcraft* (*WoW*). This model is based upon *WoW* traffic models from Svoboda et al. [126], coupled with my own measurements obtained using WireShark and *WoW* version 3.1.

WoW clients require on average 2.1 Kbps upload and 6.9 Kbps download bandwidth. The current implementation of *WoW* uses a command / state response model for propagating state changes. I verified this by measuring bandwidth consumed by both an actor node and an observer node, noting that each observable action taken at the actor resulted in a small transmission to the server, followed by update packets being transmitted to both the actor and observer. The update packets were identical in size, and typically larger than the command packet transmitted to the server.

Most game DVEs send frequent, very small packets, often with little or no payload. For example, in *WoW* 57% of download packets have an empty payload. Analyzing my own packet trace, I found that in 821 seconds of activity in a popular end-game zone, 3,881 ethernet packets were received, wi1th a total size of 284,246 bytes. My per-packet transport overhead for Ethernet and IP framing was 54 bytes per packet, which is 209,574 bytes of overhead. My measurement shows 74% packet framing overhead, leaving 26% actual event data (i.e. $\bar{F}(E_i) = 1.79$ Kbps if $\bar{R}(E_i) = 6.9$ Kbps) in communications. This tells me the numbers from Svoboda et al. [126] provide conservative values, and are therefore a good benchmark for my purposes.

Average inter-arrival time for state update packets is 220ms, giving $E_i = 4.545$. The size of a full state snapshot as received from the server on initialization depends heavily upon avatar location within the DVE. I measured values between 26 KB for a quiet area to 125 KB for a busy one. I choose $|S_i|$ = 62.5 KB, roughly halfway between the two extremes. I assume Q is 10, a compromise between very busy areas with tens of mutually interacting avatars, and very quiet areas with only enough connections to maintain P2P topology. I assume the P2P version of World of Warcraft at a minimum needs to propagate state updates to affected parties, which is the basis of my traffic analysis.

5.2 Threat model

Carbon is an auditing system intended to provide DVEs a way to identify cheaters (nodes violating DVE software rules). To help evaluate its efficacy, this section contains a variety of threats derived from game DVEs such as *WoW* and *Quake*, typical deployed DVEs. These threats are described in terms of the STRIDE [62] [65] model and the relevant categories in Yan and Randell's cheat taxonomy [134]. I believe these represent a significant category of threats, and serve to provide an illustration of strengths and weaknesses in the Carbon approach. Many of the example threats outlined below are detailed in Hogland and McGraw's book on cheating in MMOGs [64] and a "state of the art" description of cheating by Schloss-Griffin in [119].

Threat 1: *Network packet modification for performance enhancement*

STRIDE: Tampering

Category: Cheating by exploiting lack of secrecy

Description: State updates are modified in flight, circumventing user input and local client consistency checks. Aimbots use this technique, via a network proxy which modifies "shoot" packets in flight, changing the aim-point to always hit the opponent, instead of reflecting the manual aiming normally required.

Threat 2: *Local DVE rule circumvention*

STRIDE: Tampering

Category: Cheating by exploiting misplaced trust

Description: Avatar breaks DVE state update rules, for example by casting spells, moving, or attacking more quickly than allowed, or without the required skill or materials. This is done by either modifying local client code to disable certain checks, or by directly modifying system memory holding relevant values.

Threat 3: *Collusion for rule circumvention*

STRIDE: Tampering

Category: Cheating by collusion. Cheating by exploiting misplaced trust

Description: For DVEs where avatar-related state is owned by third parties, colluding with those third parties to modify avatar state in a fashion contrary to DVE rules. This can be for the advantage of an attacker avatar, or to the detriment of a target avatar. One example would be editing an avatar's current or maximum health to be higher or lower than it actually is.

Threat 4: *Time stamp forgery*

STRIDE: Tampering

Category: Timing cheating

Description: When receiving an update which a client wishes to prevent, they can issue a conflicting update with an earlier timestamp. This typically results in rollback of the undesirable state, and revision of state to reflect the forged message as preceding and possibly invalidating the prevented update. Alternatively, a client can execute the "delayed update" attack to choose the most advantageous action in relation to the received update, and send their update message out time stamped earlier than the actual time to exploit this knowledge.

Threat 5: *DVE message forgery*

STRIDE: Spoofing identity

Category: Cheating by exploiting lack of secrecy

Description: The attacker can forge messages, making a node appear to request or authorize a state change which it did not. Examples include forging messages indicating an enemy avatar dropped its weapons, took off its armor, ran off a cliff, etc.

Threat 6: *Inconsistent state representation*

STRIDE: Tampering

Category: (none)

Description: The attacker provides a different value for their current state to different nodes. For example, the attacker could tell every other avatar that they are located close to that avatar so the attacker's node will be notified of all avatar movement and positions, rather than just those near the avatar.

Threat 7: *Theft of virtual assets*

STRIDE: Spoofing

Category: Cheating related to virtual assets. Cheating by compromising passwords

Description: This includes transfer of gold or items in a game such as *WoW* without the authorization of the rightful owner. The most common attack vector is a compromised account name and password, coupled with valid in-game mechanisms to move the assets to a new account.

Threat 8: *Automating behaviors in violation of DVE rules*

STRIDE: (none)

Category: Cheating by exploiting machine intelligence. Cheating by modifying client infrastructure

Description: Scripting avatar behavior so that it automatically acquires resources and performs without the user having to be present. Farm bots and fishing bots are two examples from *WoW*. WowGlider [88] (now defunct) was a commercially available cheat utility which implemented this attack.

Threat 9: *Unauthorized information access*

STRIDE: Information disclosure

Category: Cheating by exploiting misplaced trust. Cheating by modifying client infrastructure

Description: Obtaining DVE state the attacker is not entitled to know. For example, position and distance of victim avatars outside of avatar's area of interest, or whose position should be obscured (e.g. by a wall using wall hacks), configuration of dynamic resources and territories on unexplored portions of a map, or the contents of unopened objects such as chests.

Many of these threats are mitigated or partially mitigated in a client-server implementation, but new mitigations would need to be enacted for a distributed – e.g. hybrid or P2P – implementation of a DVE.

Impact of Carbon in addressing these threats will be discussed at the end of this chapter.

5.3 Carbon

Distributed virtual environments (DVEs) are large collections of state, and rules for modifying that state. Carbon is an auditing system allowing DVEs to detect illegal state changes.

DVEs must meet certain prerequisites in order to use Carbon. Section 5.3.1 spells out those requirements, and introduces nomenclature for discussing how Carbon interacts with eligible DVEs.

Carbon consists of two modules: an audit client embedded in each DVE client node, and an auditor embedded in DVE code running on one or more trusted nodes. The auditor evaluates recorded DVE state, verifying legality of the simulation run as viewed at a given participant node.

Carbon is DVE-agnostic. It does not understand the intricacies of how a given DVE operates. Instead, it provides a set of basic services DVEs use to organize auditable information. In most cases, adopting Carbon requires little modification of the DVE.

The remainder of section 5.3 is divided into four parts. Section 5.3.1 outlines DVE requirements and introduces nomenclature. Section 5.3.2 describes Carbon audit client requirements, and the client *Reporter* component in detail. Section 5.3.3 describes the Carbon *Auditor* component. Section 5.3.4 provides an example illustrating behavior of a DVE using Carbon. **Note**: the reader may wish to skim section 5.3.4 before reading further, to help motivate nomenclature and design.

5.3.1 Nomenclature and DVE requirements

The diversity of DVE types has led to an explosion of different nomenclature. This section provides my preferred nomenclature (listed in Table 13), and lists a set of requirements a DVE must meet to adopt Carbon.

Symbol	Meaning
S_i^t	State at node i at time t
E^t	Event received at time t
ID_i	DVE identity for node i
$L_i^{t1,t2}$	Audit log for node i from $t1$ to $t2$
$M_{i\to j}$	Message from node i to node j

Table 13: DVE nomenclature

A node is a DVE instance running on a computing resource, typically providing the view and interaction point for a single avatar. State is the collection of all local authoritative and shadow state. An event is defined by the DVE itself, but is typically anything except a node state snapshot: it may be a state change, a user command, or anything else. A message is a container for DVE or Carbon information. It can contain state snapshots, events, audit log extracts, etc.

Undecorated numeric or variable subscripts refer to a specific node. Subscripts prefixed with c refer to Carbon auditors. For example, $M_{i\to cj}$ describes a message from participant node i to Carbon auditor cj. Superscripts refer to a time or time interval. For example, $L_i^{t1,t2}$ refers to an interval of node i's log from time $t1$ to time $t2$, inclusive.

In order for Carbon to operate, a DVE should be able to simulate forward from state snapshots using events, to compare states for similarity, and to determine whether a given event is legal to apply to a given state. DVEs which offer recording and replay of games – such as *Quake III* – already meet these requirements.

Formally, a DVE is a collection of state for the N active nodes $S = \bigcup_{i=0}^{N} S_i$ and a set of rules for changing that state. The overall DVE state S is a union of individual node state S_i. Individual nodes may have overlapping DVE state. Ideally one copy of a given state variable is authoritative and the rest are shadow (non-authoritative) copies, but this is not required.

An event E is a state change or command which can result in state change for a given node's state, i.e. $S_i^{t2+\epsilon} = ProcessEvent(S_i^{t1}, E_i^{t2})$, where $t1 < t2 <= \epsilon$. Given the state of a node at any two times in the DVE, it should always be possible to reach the successor state by taking the predecessor state and applying a series of *ProcessEvent* operations with the appropriate events.

The DVE must be able to communicate state between nodes via messages. A node must be able to initialize itself based upon a combination of local state, and received state and event messages. Again, these requirements are already met by most DVEs.

DVE state changes must be deterministic. This does not rule out *choosing* state changes randomly, but a given random choice must induce a deterministic change, and both the choice and change are events which should be logged.

5.3.2 Carbon audit client: "Reporter"

The Carbon audit client is a small module implementing the DVE participant node portion of Carbon. I refer to this code module as the *reporter*, since it is responsible for taking notes on the client's behavior and interaction, and reporting these to the auditor. The reporter is implemented as a library, invoked as required by DVE client code.

The reporter provides information the auditor needs to perform audits. It is a store for client state snapshots and events both generated at and received by the DVE node. Events consist of any information material for determining state changes. This typically consists of outgoing and incoming DVE messages, but may include other information such as mouse moves and key clicks, depending upon the DVE's needs.

From the reporter's perspective, state snapshots and events are opaque data blobs, stored as simple byte arrays. Neither the reporter nor the auditor components have any knowledge of how the DVE operates, or what event and state data mean.

The reporter exposes the interfaces described in Table 14 for the exclusive use of the DVE node.

Function	Description
Startup	Initialize the reporter
Shutdown	Shut down the reporter
Log	Add an event or state to the log
Commit	Commit to the auditor
RequestAudit	Request an audit
ProcessMessage	Process a Carbon message
RetrieveNotice	Retrieve a Carbon notification
ReleaseNotice	Release a retrieved notification

Table 14: Reporter functions

The reporter does not have a thread, and does not directly transmit network messages. The DVE node calls *Startup* to initialize the reporter. It provides the local node ID and a trusted auditor ID. It calls *Shutdown* to release any transient reporter data and flush data to storage.

The DVE node submits auditable events and state to the reporter via the *Log* function. *Log* takes the log data type (event / state), DVE time, and byte array as parameters. The DVE node periodically calls *Commit* to submit its most recent state snapshot to the auditor.

Upon receiving a remote message intended for the reporter – typically from a trusted auditor – the DVE node calls *ProcessMessage* with the sender ID and message payload. The two cases where this happens today are:

1. Requesting an audit log extract.
2. Supplying audit results.

If the DVE node wants a remote node audited, it calls *RequestAudit* with that node's ID, the minimum DVE time range to audit, and an optional state snapshot. This function would typically be called when a DVE node is informed of state which it doubts, for example with the arrival of a new avatar.

The reporter uses *notices* to communicate relevant information to the DVE node. The DVE node calls *RetrieveNotice* to retrieve a notice whenever a reporter call indicates a notice is waiting, and *ReleaseNotice* to free it.

There are two reasons the reporter returns data to the DVE node via a notice:

1. To request message transmission, for example in response to a received message, or as the result of a call to *Commit*.
2. To provide audit results to the DVE node.

5.3.3 Carbon auditor: "Auditor"

The *auditor* is provided as a small library used by the DVE. It runs as a trusted DVE system component with the primary purpose of accepting state snapshots and performing DVE audits.

The auditor is an advisory component. It does not directly make decisions. It provides a framework for collecting information the DVE can use to make audit decisions, and for disseminating the results to reporters. The DVE controls when an audit should be performed, audit success evaluation, and what action to take upon a successful or failed audit.

The auditor can be embedded within an existing DVE server component. Or, a new purpose-built code base can exchange messages and perform audits on behalf of the auditor. Like the reporter, the auditor has no thread of its own.

Function	Description
Startup	Initialize the auditor
Shutdown	Shut down the auditor
RequestAudit	Require an audit
CompleteAudit	Provide audit results
ProcessMessage	Process a Carbon message
RetrieveNotice	Retrieve a Carbon notification
ReleaseNotice	Release a retrieved notification

Table 15: Auditor functions

The table above lists auditor library functions. Most fulfill the same function as their namesakes in Section 5.3.2. The only new interface is *CompleteAudit*, used by the DVE server to return audit results to the Carbon auditor, along with a list of IDs to notify.

ProcessMessage can receive three different messages, each of which raises a new notice the auditor must retrieve via a call to *RetrieveNotice*.

1. *StateNotice*, a state snapshot message.

2. *AuditRequestNotice*, an audit request message.
3. *AuditResultNotice*, an audit result message, for example a local audit result.
4. *LogNotice*, a requested log excerpt.

Each time the auditor receives a new state snapshot message, it persists the state, and retrieves the immediate predecessor and successor state snapshots for that participant – if any. This provides the basis of a state notice the DVE server can use to determine if it should perform an audit. For example, if the magnitude of changes between subsequent state snapshots seems very unlikely, it may trigger an audit.

If an audit is required, the DVE server calls *RequestAudit*. This call is authoritative since it is made by an auditor, and results in a log excerpt request for the audited node.

When the log excerpt is received and raised as a notice, the DVE server retrieves it, and performs an audit based on the earlier state notice and the log excerpt. It notifies the auditor of the result by calling *CompleteAudit* with the final state and the audit success or failure. The auditor sends a corresponding audit result notification to the audited party, and to any other participant listed in the optional audit notification list, including potentially itself.

The example in Section 5.3.4 below illustrates the system.

5.3.4 Carbon system operation

This section provides a detailed example of a DVE using the Carbon auditing system. I assume a P2P DVE with unique participant identities. Participants can connect and disconnect from the DVE at will, resuming their activities whenever they have time. For my example, suppose the DVE requires a state snapshot every 15 minutes.

As a reminder, the system has four types of actors: A *DVE node* is a client instance. It contains a Carbon *reporter*, responsible for Carbon client activities. The DVE *server* is a trusted DVE component. It contains a Carbon *auditor*, which coordinates auditing. This is illustrated in Figure 9, with two client nodes, Alice and Bob.

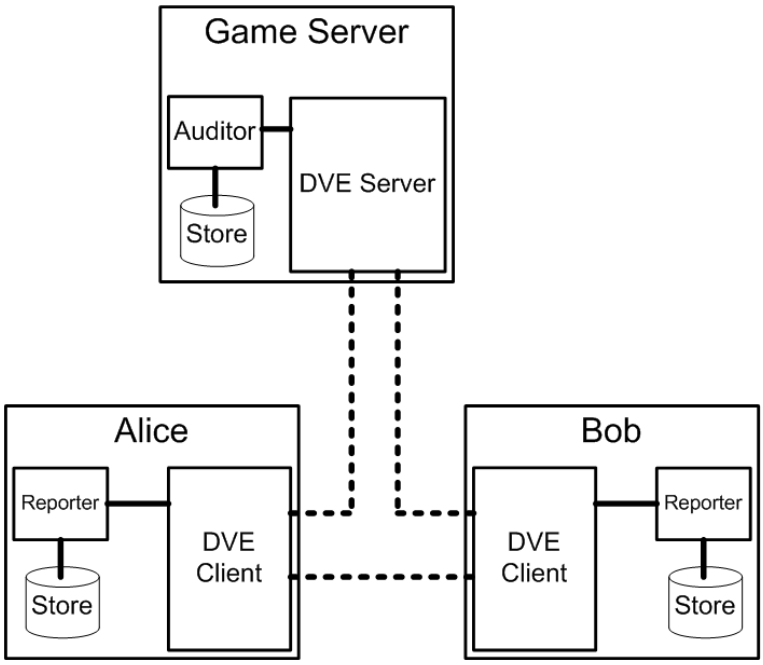

Figure 9: Carbon system

Alice wishes to continue her avatar's virtual life. She starts up her DVE node. As part of initialization, the DVE node code calls *Startup(ID_A , ID_{C1}),* initializing the Carbon reporter with her ID and the ID of a trusted auditor.

The DVE node loads Alice's avatar and finishes integrating it into the DVE. The DVE node serializes a copy of Alice's avatar state and invokes *Log($t0$, S_A^{t0}),* which stores the state snapshot to the local audit log. Then it invokes *Commit()* which packages the latest state snapshot into a message for the auditor. *Commit()* signals the DVE node that a new notice is available for retrieval from the reporter. A call to *RetrieveNotice()* retrieves the message $M_{A\rightarrow C1}$ to send to the auditor. The DVE node connects to the appropriate DVE server, transmits the message, and calls *ReleaseNotice($M_{A\rightarrow C1}$)* to release its copy of the network message.

The DVE server receives the message, and ensures the sender matches the message source ID. It notes that the message target ID belongs to its hosted auditor, and invokes *ProcessMessage($M_{A\rightarrow C1}$).* The auditor deserializes the message, and saves the received state snapshot S_A^{t0} into its state snapshot table for Alice. The auditor constructs a *StateNotice* notification triple $SN = (S_A^{t0-\epsilon}, S_A^{t0}, \emptyset)$, and notifies the DVE server. The DVE server invokes *RetrieveNotice()* and receives the *StateNotice*. It evaluates the state snapshots, determines no audit is needed, and calls *ReleaseNotice*(SN) to return the resources back to the auditor.

Alice participates in the DVE, with her DVE node sending and receiving network messages with state changes. Her DVE node also accepts and processes her local input. Each inbound and outbound network message – with the exclusion of Carbon messages – is considered an *event*, and its payload is logged to the reporter log via a call to *Log(t,E_A^t).* The DVE can optionally record Alice's inputs for auditing. Input events can be stored in a local event queue. Each time a network message is sent or received, the DVE empties the local event buffer contents into a new "user input" event message, and logs it. The reporter and auditor do not differentiate between these two categories of events, though the DVE server does.

During Alice's session, her avatar encounters a new avatar Bob. When Alice's DVE node receives the message describing Bob's avatar state $SB^{t1} \subset S_B^{t1}$, it decides to request Bob be audited. Alice's DVE node invokes *RequestAudit($t1$, ID_B, SB^{t1}),* which creates a new audit request message, which her node retrieves from the reporter and sends to the auditor on the DVE server.

The auditor looks up Bob's state snapshots which fall within or immediately precede the audit interval. In this case, suppose there is a single previous snapshot $S_B^{t1-\epsilon}$. An *AuditRequestNotice* ($S_B^{t1-\epsilon}, SB^{t1}, t1-\epsilon, t1$) is created by the auditor and retrieved by the DVE server. The DVE server compares the states and timespan, and determines an audit is warranted. The DVE server invokes *RequestAudit(ID_B, $t1-\epsilon$, $t1$)* specifying who and over what interval to audit. The auditor constructs a log excerpt request *(ID_B, $t1-\epsilon$, $t1$)* and transmits it to Bob's reporter via a notice and DVE-transmitted message, as explained above.

Bob's reporter constructs a serialized log excerpt $L_B^{t1-\epsilon,\ t1}$ of Bob's events between $t1-\epsilon$ and $t1$, then sends the auditor the excerpt as above.

The auditor extracts the message and embeds the excerpt in a *LogNotice*. The DVE server pairs this excerpt with the state snapshots it already had, and forward simulates from $t1-\epsilon$ to $t1$ checking the legality of each event as it is processed. Once the simulation time reaches $t1$, the DVE server compares SB^{t1} with its calculated version in S_B^{t1}. If the state variables specified in SB^{t1}match the value of the same state variables in S_B^{t1} then the audit passes. Otherwise it fails.

If the audit was successful, the DVE server makes a list with Alice and Bob's IDs. If the audit failed, it makes a list which includes Alice, Bob, and any other participants the DVE server wishes to notify of the audit failure, such as Bob's neighbors.

The auditor constructs a series of audit result messages containing notification of audit results, one per recipient in its list, and transmits the notifications to recipients as above.

When a reporter receives the audit result, it creates an *AuditResultNotice* $(ID_B, t1 - \epsilon, t1, RESULT)$ and passes it to its DVE node. The DVE node code is responsible for performing an appropriate action, such as continuing simulation, or disconnecting from Bob.

5.4 Analysis

P2P DVEs have more security vulnerabilities than client-server DVEs. State storage and modification is performed on untrusted peers, increasing their vulnerability to client misbehavior, collusion, and inconsistency attacks. There is no guarantee peers responsible for these activities are executing the prescribed code base and obeying DVE rules. A P2P DVE system requires means for correct nodes to ensure correctness of other nodes behavior, whether directly or indirectly.

Participants in large-scale P2P DVEs typically possess only a fragment of the overall DVE state, some authoritative, and some cached shadow state. They rely upon other DVE nodes to provide them with shadow state updates at appropriate times, for example when another participant moves within their area of interest.

Given a set of trusted audit servers, Carbon allows a DVE to mitigate vulnerabilities related to misrepresentation of state, and to detect illegal state modification. More specifically, while the system cannot guarantee the represented state is correct, it can at least guarantee that the represented state is reachable from an earlier (trusted) state, and that the avatar presenting the state can produce an event sequence which reaches the represented state. This provides partial mitigation for client misbehavior, collusion, and inconsistency attacks.

The remainder of this section analyzes the behavior and overhead of Carbon. Since PeerReview is closely related to Carbon in terms of intent and behavior, I compare and contrast Carbon's overhead with that of PeerReview.

Section 5.4.1 discusses Carbon efficacy in terms of the threats outlined in section 5.2. Section 5.4.2 provides some details on PeerReview's behavior, and high-level comparison with Carbon. Section 5.4.3 provides finer grained details about Carbon and PeerReview's client resource consumption in the DVE scenario. Finally, section 5.4.4 discusses Carbon auditor resource consumption.

5.4.1 Audit coverage

DVEs implement deterministic state machines. Given access to a state snapshot and an event, any node can determine the resulting state. This principle provides the basis of my auditing solution.

P2P DVE nodes usually perform similar activities to one another, with similar levels of trust, ideally none. In some architectures a subset of nodes are granted additional responsibilities for coordinating DVE activities, but such responsibilities are typically based upon node resources rather than trustworthiness.

Carbon provides an auditing framework for detecting invalid state transitions within the DVE. The DVE can use Carbon-provided data to perform audits, or more complex analysis, such as detecting illegal input devices.

Carbon's goal is to enable a DVE to ensure avatar integrity and correctness. It is impossible to verify avatar state integrity in isolation: the avatar's state is affected by its environment. A system examining only avatar state lacks context to verify it. For example, suppose avatars have a "money carried" property, and Alice violates DVE rules by modifying her avatar to have a million dollars. If the auditing system evaluates only avatars, and if there is any non-avatar source of money – for example money lying on the street – Alice could claim upon audit that her million dollars was found on the street, with no way to disprove her claim. By increasing the audit scope to include Alice's entire DVE node state, the auditor has access to context which can help validate or refute Alice's avatar state: it can review her node's simulation to learn about any money on the street, and can confirm the amount is appropriate. If the provenance of the money is suspect, its source in the DVE can also be audited, and so on.

Section 5.2 presented nine threats against DVE integrity. These threats are listed again below, with a brief discussion of Carbon's efficacy against them for distributed scenarios.

Threat 1, *Network packet modification for performance enhancement*.

Mitigated. Carbon can prevent attacks which work by externally modifying events, such as a network proxy-based aimbot. If messages are modified outside the client, the client is unaware of the value of those messages, and cannot adjust its logged message copy or state accordingly. This leads to state inconsistency between nodes, and inconsistent messages, both of which are detectable by Carbon auditing.

Threat 2, *Local DVE rule circumvention.*

Partially mitigated. Carbon cannot detect modification or replacement of remote client software, or the replacement of values in system memory. However, Carbon does ensure externally observed client behavior respects DVE rules. These rules are typically related to faithfully executing network protocol, and state storage and transition. Carbon partially mitigates this attack. If the network protocol is violated, then clients participating in the protocol will see unexpected messages. These messages are logged at both the sender and receiver, and the auditor can detect an inconsistency when either party is audited. This in turn can trigger an audit of the compromised node, which reveals its misbehavior. Carbon also detects illegal state transitions. Audits evaluate state transitions at each node, ensuring they follow DVE rules. If they do not, then the auditor can detect this, and flag the node as violating DVE rules.

Threat 3, *Collusion for rule circumvention.*

Partially mitigated. The collusion attacks Carbon is concerned with involve nodes collaborating to selectively subvert or ignore DVE state transitions. If any DVE state is changed without adhering to externally observable DVE rules and both parties log their transactions, auditing can detect and expose the parties who illegally changed that state, regardless of how they are colluding.

Specific categories of collusion require additional checks by DVE clients to detect. For example, suppose Alice transfers a million dollars to Bob, then clears that transaction from her state and audit log. If Bob retains this transaction, then individually each node's log would appear correct, but a million dollars would have been duplicated. Such attacks can be detected by comparing state snapshots of interacting clients during audits. For example, Bob's auditor could flag the transaction with Alice as significant, and from its own audit session request an audit of Alice for the same interval, providing its calculated state for Alice's avatar as the state snapshot to compare against. Alice's audit would fail, because the state provided by Bob's auditor would differ significantly from the state calculated by Alice's auditor. Note that questions of message authenticity (for example, did Bob forge the transaction from Alice, or is Alice lying?) needs to be addressed by the DVE. The PeerReview auditing system integrates this protection globally via authenticators, but at a dramatic cost as discussed in sections 5.4.3 and 5.4.4. I believe selectively protecting against this attack is a better compromise for most DVEs in terms of cost vs. benefit.

Threat 4, *Time stamp forgery.*

Partially mitigated. Timing attacks involve forging timestamps on packets. Carbon can defend against a subset of this attack. To circumvent Carbon auditing, an attacker would create a message, forge the timestamp, and rewrite their local audit log to place the message in the correct location in the log. So long as the log was not committed to the auditor in the interim, this attack is undetectable. However, modifications which cross a commit boundary could be detected. It is worth noting that PeerReview

provides stronger protection against this attack, so long as the attack results in a reordering of transmitted messages.

Threat 5, *DVE message forgery.*

Partially mitigated. Carbon uses a unique numeric identity for each node, but relies upon the DVE to ensure those identities are sufficiently strong, and that transactions between DVE nodes are suitably identified and secured. While Carbon does not directly address inter-node communications and therefore Sybil attacks, meeting its identity requirements in a reasonable fashion provides a basis for the DVE to implement standard Sybil attack defenses.

Threat 6, *Inconsistent state representation.*

Mitigated. Nodes periodically commit their logs to the auditor. A state commit can only match a single sequence of events. If a cheating node sends inconsistent state representations to two nodes, only one of those representations can possibly match the auditable event stream the attacker commits to. Each node receiving state from the cheating node can request an audit, including the state supplied to them by the cheater. If the resulting audit does not match the provided state – which MUST be the case for all but one of the receivers of different state – then the audit fails and the cheater is exposed. Note that signing transmitted state snapshots may be required to make assertions of received state sufficiently strong.

Threat 7, *Theft of virtual assets.*

Not mitigated. The virtual asset theft scenario involves compromise of the system outside of the DVE rule set, and does not violate any DVE state change rules. Carbon can be used to enforce internal consistency of the DVE, not player compliance.

Threat 8, *Automating behaviors in violation of DVE rules.*

Not mitigated. Most automation attacks involve forging user input to the DVE by modifying the surrounding operating environment. For example, using WowGlider to inject windows events into the Windows version of *WoW* using the Windows API. It may be possible to log additional auditing information which would allow such attacks to be detected, but such defenses are outside the default scope of Carbon auditing.

Threat 9, *Unauthorized information access.*

Partially mitigated. Mitigation depends upon the attack vector. If the unauthorized information is obtained by modifying the execution environment, for example replacing system rendering libraries and causing the environment to be rendered differently than specified by the DVE, then this threat is not mitigated. Examples of this sort of attack include changing the names of model files to cause interesting objects to be easier to see, and using wall hacks to allow rendering of objects which should be obscured.

If, on the other hand, the attack is mounted by requesting and obtaining information the client is not entitled to, then Carbon auditing can detect the unexpected request / response and report the violation, mitigating the threat.

Exhaustive auditing via Carbon reliably detects most of the threats specified in section 5.2 with reasonable overhead, as discussed in section 5.4.3 and 5.4.4. If less expensive auditing is desired, DVEs can leverage participant affinity for their avatar, assigning punishment sufficient to deter

cheaters who believe they will be caught, and reducing the fraction of total transactions audited. The DVE should adjust punishment severity and audit probability until an appropriate deterrence is achieved. For example, a 50% chance of having their avatar's lifespan cut short may be sufficient to deter most cheating.

As mentioned earlier, Carbon does not directly evaluate DVE state correctness. Instead, it collects and organizes information for the DVE to determine when an audit should be performed. Likewise, performing the actual audit is left up to the DVE code itself. An audit can be as straightforward as verifying successive state transitions are legal, or as complex as correlating state transitions between multiple DVE views from multiple participants, or performing deep data mining to uncover more elusive DVE violations such as account sharing or theft [36] and dependency hacks such as wall hacks [82].

5.4.2 Comparison with PeerReview

PeerReview is a decentralized auditing system with good scalability and correctness guarantees. State changes and local transactions relevant to state calculations are stored in certified append-only local logs. Log contents are committed by peer exchange of signed log digests, and periodically uploaded to untrusted auditors called *witnesses*. *Witnesses* simulate forward from a state snapshot through a series of logged events to ensure simulation correctness. Based on audit results and general behavior, nodes are labeled as trusted, suspected, or exposed (bad).

Carbon and PeerReview have many similarities.

- Both are auditing systems, focused on error detection, not error prevention.
- Both are asynchronous, with auditing performed out of band.
- Both systems require client nodes to capture system snapshots and events, and to provide those events to auditors.
- Both use third party auditors who, given a state snapshot and a related log excerpt, forward simulate through the log excerpt to ensure behavior over the simulated interval is correct.

Carbon and PeerReview have a number of differences, as well.

- **Auditor trust**. PeerReview can operate even when some auditors misbehave. Carbon trusts auditors to be correct.
    - PeerReview: Does not trust individual auditors, and is resilient to a limited number of auditor errors.
    - Carbon: Trusts the auditor, and requires that operating auditors be correct.
- **Client log security**. PeerReview constructs tamper-evident logs with per-entry tamper protection. Carbon achieves tamper evidence by submitting hashes over large sets of log entries to the trusted auditor.
    - PeerReview: Allows log validation between any two events, and auditing forward from any state snapshot.
    - Carbon: Allows log validation between any two submitted hashes, and auditing forward from any state snapshot.
- **Message security**. PeerReview provides a mechanism for securing actual message exchanges between clients. It effectively creates a signature ("authenticator") for each message, and requires a signed response. This imposes a specific structure on transactions, but provides non-repudiation, tamper detection on transmit, and many other desirable properties.
    - PeerReview. Message signing and positive acknowledgement of every packet provides message non-repudiation, and enables easy detection of certain classes of

cheat such as omitting key messages from sender logs, but still reporting them on the receiver.

- Carbon. No network message security is provided. Instead, it is expected the DVE will have its own transport security, and if desired, perform message validation as part of event auditing.

- **Completeness**. PeerReview typically audits all events by all clients in the system. Carbon uses application feedback to selectively audit suspected intervals.
  - PeerReview: Exhaustive validation of all system behavior. Probabilistic interval selection proposed but not described.
  - Carbon: Selective validation. Validation explicitly guided by the application and its clients.
- **Inter-client impact**. PeerReview affects every inter-client interaction to provide message delivery guarantees. Carbon does not, but I believe it provides sufficient security for DVEs.
  - PeerReview: Requires a signed log commit (authenticator) for each transmitted message, and a similar receipt acknowledgement.
  - Carbon: Does not affect inter-client traffic.
- **Implementation**. PeerReview's implementation uses sub-classing from auditor defined classes, with an audit event engine driven by PeerReview. Carbon is implemented as an API with audit event engine driven by the application.
  - PeerReview: Has total control over audit process, but significant structural and integration requirements.
  - Carbon: Application links to Carbon libraries and calls APIs as it sees fit. Carbon has no execution threads of its own.

The remainder of this section provides quantification of some of the differences between Carbon and PeerReview, as they pertain to performance. These sections rely heavily upon my description of typical DVE characteristics and my *WoW*-like example in section 5.1.4. Section 5.4.3 describes client resource impact of adopting Carbon, and compares it to the overhead required to adopt PeerReview. Section 5.4.4 provides a similar analysis for auditor resource requirements.

5.4.3 Carbon and PeerReview client overhead analysis

Adopting an auditing or security scheme can have a significant impact on DVE resource consumption. Game DVEs are performance-hungry, and many consumers buy high-powered computers specifically to improve their DVE performance. Any DVE considering a security model needs to pay careful attention to the overhead that model induces.

This section describes the client overhead associated with Carbon, and contrasts it to the overhead for adopting PeerReview for this scenario. It is worth mentioning that DVE traffic patterns – small, frequently sent packets – are a worst-case scenario for PeerReview, and a best-case scenario for Carbon. The former is a side effect of the broad applicability of PeerReview to distributed network protocols, while the latter is an intentional design criteria for Carbon.

I divide my analysis of client overhead into four categories: network bandwidth, message latency, client persistence workload, and CPU overhead.

5.4.3.1 Bandwidth

Auditing typically involves transmission of information to auditing parties, and possibly to other clients. The most obvious impact of this activity is on client bandwidth consumption.

Carbon induces additional client network load consisting of:

- Copying the payload for non-carbon messages sent or received to the client's auditor, when requested by the auditor. If 10% of all transactions are audited, then 10% of payloads will be copied to the auditor.
- Periodic state commits, uploaded regularly to the auditor

For DVEs which exchange large amounts of state information, full auditing with Carbon increases overall client traffic to roughly double its original amount. For more typical DVEs which have small, frequent exchanges of state, Carbon overhead is partially mitigated by the more efficient framing of data sent to the auditor: Whereas small payloads are frequently transmitted for normal DVE interaction, auditing can bundle up many payloads before transmitting them to the auditor, resulting in reduced packet framing costs relative to the payload size. For my example of a *WoW*-style P2P game, this reduces additional bandwidth consumed for propagating events from 100% additional overhead to 27% additional overhead.

A subset of PeerReview overhead is similar to that for Carbon, as it must propagate events and state snapshots to auditors. However, PeerReview typically has a quorum of auditors for each client, so each client transmits these messages several times. In addition, PeerReview has several additional sources of overhead, as described below.

- A PeerReview *authenticator* must be included in each DVE message. The authenticator is a sequence number / SHA-1 hash pair signed with a 1024-bit RSA key, with a total size of 156 bytes. This is required for each outgoing event message. The authenticator is acknowledged by the recipient with a new authenticator roughly the same size, but which needs no additional acknowledgement.
- Each received authenticator is transmitted to every auditor responsible for the authenticator creator.

Suppose both auditing systems have a state snapshot interval Z, and a probability p of auditing the interval between any two subsequent state snapshots. For subsequent analysis, let me assume $Z = 900$ seconds, and $p = 1$.

In the terms outlined earlier, Carbon bandwidth overhead can be characterized as $(|S_i|/Z) + p \times \overline{F}(E_i) \times (2Q)$. PeerReview overhead, assuming a single trusted witness per client rather than a quorum, would be $(|S_i|/Z) + (3 * E_i * 156 * 8/1024 + p \times \overline{F}(E_i)) \times 2Q$ Kbps, or $(3.744 \times E_i \times 2Q)$ Kbps more than Carbon. Plugging my numbers from the *WoW*-style game above, this gives me Carbon auditing overhead of 37.4 Kbps, and PeerReview auditing overhead of 340 Kbps, nearly an order of magnitude difference.

It is worth noting that auditability of an interval for both Carbon and PeerReview relies upon having an initial state snapshot, plus a contiguous series of audit log entries through the audited interval. Ideally the audit should include a final state snapshot to compare against audit results, although this is not strictly necessary.

This has significant implications for PeerReview auditing interval selection. Even though PeerReview provides a tamper-evident log which can be examined between any two intervals, the reality is that other than verifying the sequence of messages matches the authenticators for any given sub-interval, no auditing can be performed except for an interval starting at a state snapshot. In my example above, this means auditing must be performed continuously from the most recent state snapshot preceding the interval of interest, providing no state auditing advantage over Carbon.

On the other hand, the extra overhead of PeerReview authenticators enables certain protections which are weaker or not present in Carbon.

5.4.3.2 Latency

Carbon introduces no latency into most client interactions, as it does not modify existing message exchange. The only place where latency may be introduced is when the packets for additional

exchanges with the auditor collide with other DVE traffic. Since auditor traffic is infrequent – on the order of an exchange once every several minutes – the overall effect is negligible. Further, the Carbon model allows the application to choose when audit traffic is generated and transmitted, so audit transmissions can be delayed until a time the application finds convenient.

PeerReview adds an authenticator to every inter-client event message, resulting in negligible but non-zero delivery latency. Specifically, each packet has an extra 156 bytes of data, which on a 1 Mbps circuit requires just over a millisecond of extra transit time before the packet is fully received and can be decoded. The authenticator also requires cryptography to create, which according to PeerReview's analysis, can induce another 1.5 ms of latency in circa 2007 hardware. This results in a minimum latency impact of 2.5 ms, negligible in most WAN scenarios.

5.4.3.3 Storage

Both Carbon and PeerReview induce client storage overhead for client logs, roughly equal to the node's network traffic, plus auditing overhead. In my example, a typical week of *WoW* usage (614 minutes) [106] for a Carbon DVE client would consume approximately (614 * 60) s * 37.4 Kbps = 172 MB of audit client storage per client. A PeerReview audit log for the same time period would consume 1.57 GB of audit client storage, a larger but still acceptable amount. The difference in log size is because of the authenticator required for PeerReview. Each authenticator is relatively small, but with an average event payload measured in tens of bytes, two 156-byte authenticators per payload represents significant transfer and storage overhead.

5.4.3.4 CPU

Client CPU impact is not significant for Carbon, but may be an issue for PeerReview.

For Carbon, audit entries are retained in the local reporter log, but no cryptographic operations are performed upon them, except for a hash every snapshot interval (once per fifteen minutes in my example above).

For PeerReview, each client event message sent or received results in calculation of two non-signature hashes, an RSA signature, and an RSA signature validation. While hashing is relatively quick, signing time can quickly add up, especially for periods with high event rates.

PeerReview quotes a signing and validation time of 1.5 ms. Each client must sign and verify $E_i \times (Q + 1)$ messages per second. This consists of the E_i event messages it originates and transmits to subscribers, and the $E_i \times Q$ event messages it receives from other clients and for which it must verify the authenticator and create its own signed acknowledgement. In my example above, this is on average 50 messages per second, for a total CPU overhead of 75 ms per second. While not prohibitive on average, DVE client load has spikes which can easily be an order of magnitude larger than average, for example during a *WoW* battleground melee. This would raise CPU overhead to 750ms per second, or 75% of one core, enough to significantly impact DVE client performance.

5.4.4 Carbon and PeerReview auditors and overhead

The Carbon auditor is a trusted component which should be encapsulated within a trusted DVE server. If no such component (such as a rendezvous server) exists, then one may need to be built. At a minimum, the trusted DVE server needs to be able to exchange network messages with DVE clients on behalf of the Carbon auditor. It should also be able to authenticate connections and correlate the remote party with Carbon identifiers.

The PeerReview auditor (Witness) is an untrusted component. Faults are detected by statistical guarantees based upon the probability of a given auditor being faulty, e.g. cheating. The quorum size of auditors per client is determined by the probability of an auditor being faulty and the risk tolerance of the system.

For my PeerReview analysis, I assume no PeerReview auditors are faulty, and so a single auditor is sufficient for a given client. This is the best case for PeerReview overhead.

Auditor overhead can be described in terms of bandwidth, storage, and CPU. I do not consider latency, as auditors do not directly participate in DVE transactions, instead auditing out-of-band. Any latency introduced between auditors is largely irrelevant to client and overall DVE performance.

For both systems, the DVE provisions a number of auditors sufficient for its needs, ranging anywhere from a single auditor auditing all clients to one auditor per client. Each audit begins with a state snapshot, and forward simulates through a series of logged events, checking each state transition for correctness. The audit can optionally conclude by comparing resulting state to another state snapshot from the audited client.

Since the client node was able to simulate its portion of DVE state based upon the event stream captured by auditing, a single auditor can replay that client's view of state and state changes. This implies each audit can typically be done without regards to any other audits, and so are fully decomposable from one another. In some cases it may be desirable to cross-check a subset of audits, which may reduce decomposability.

In addition to the checks above, PeerReview checks received authenticators to make sure messages are accurately logged by both sides of each exchange. This defense prevents attacks where one side or the other omits a message from their audit log, but not attacks where both do. Carbon does not provide this protection. Instead, Carbon relies upon the DVE to take any action it deems necessary to validate message exchange, for example auditing nodes with significant interactions on the same auditor, and comparing intermediate states to ensure no significant differences.

The remainder of this section describes the bandwidth, storage, and CPU requirements for an auditor. For my examples below, evaluation is based on the P2P *World of Warcraft* example. Analysis is done for a DVE population $W = 10{,}000{,}000$ total players, with a steady-state population of $X = 100{,}000$ active client nodes. For my analysis I specify $Y = 100$ auditors are active, for a ratio of auditors to active clients of 1 : 1,000.

5.4.4.1 Bandwidth

All Carbon client transactions are exchanges between clients and auditors, and so auditor bandwidth originating at clients is the aggregate of reporter bandwidth requirements. At a minimum this consists of commits, but may also include client audit requests, submission of requested log excerpts to auditors, and notification of audit results to clients. This traffic is detailed in section 5.4.3.1. Average auditor server bandwidth is (X/Y) times the average reporter client bandwidth. Using the example values above, average per-auditor bandwidth required would be 3.14 Mbps to have a 99.9% chance of detecting cheaters sometime in their avatar lifetime, or 37.4 Mbps for exhaustive auditing, both easily within the capabilities of a low end server.

PeerReview traffic analysis is more complicated in the general case, but somewhat simplified by my assumption of fully trusted, non-redundant auditors. Client message authenticators are created, written into logs, and then transmitted to a remote party. Both the sender and receiver log the authenticator, and the receiver transmits the received authenticator to the sender's auditor. Whenever an auditor performs an audit, it retrieves the log from the audited party, which contains similar data to that used by Carbon, plus the associated authenticators. Upon receiving a log excerpt, the auditor forwards any authenticators contained within the log – but not created by the log creator – to the auditor for the party who created the log excerpt. In addition, auditors periodically request contiguous sequences of authenticators from clients to verify log continuity and commitment. There is also a challenge protocol to verify nodes are live and responding to messages.

Since the continuity check and challenge protocol provide functionality not in Carbon, they are ignored for the purpose of comparative bandwidth estimation. State snapshot, log excerpt, and authenticator exchanges are the remaining traffic sources for auditors. Calculated traffic numbers can be divided into two categories: client/auditor exchanges, and auditor/auditor exchanges.

Using the examples above, calculations from section 5.4.3.1, and assuming exhaustive auditing, the per-client contribution to each auditor is 340 Kbps. An additional copy of each authenticator is

exchanged between auditors, for an additional per-client overhead of $(2.496 \times E_i \times 2Q) = 227$ Kbps. Total bandwidth required per auditor is 567 Mbps for my scenario, approximately 15 times as much as Carbon. As is the case for clients, the vast majority of this traffic overhead consists of authenticators.

5.4.4.2 Storage

Carbon auditors are responsible for storing state snapshots S_i on behalf of each client i, submitted as determined by the DVE client, but typically with a frequency Z. Based on the example DVE characteristics and steady-state population, this would result in an average of $(7 \times 24 \times 60 \times 60) \times (X/Y) \times |S_i| / Z = 42$ GB of data stored per server per week, a trivial amount of storage for modern servers. Audit log excerpts sent from reporters to satisfy audit requests are temporarily retained, and then discarded after the audit is completed, so they are not factored into this total.

PeerReview auditors typically retain a copy of logs submitted to them by clients, and must check any received authenticators against those logs. Only one copy of each authenticator needs to be retained by each auditor, but it appears twice: once in the sender's log, and once in the receiver's log. Total storage required per server per week would be $(7 \times 24 \times 60 \times 60) \times (X/Y) \times 226.5\ kbps/8 = 17.1$ Terabytes, more than 400 times as much data as Carbon. This makes retaining full client logs at each of the PeerReview auditors impractical. Indeed there is little value in retaining them after a successful audit.

5.4.4.3 CPU

Both Carbon and PeerReview auditors would consume CPU performing forward simulation in the course of auditing. This workload varies from DVE to DVE, but would be similar between both systems. Most DVEs consist of state changes which, given a base state and an event, are straightforward to calculate. Most CPU on DVE nodes is applied instead to rendering the DVE scene for the client, and as mentioned earlier rendering is irrelevant to auditing.

For exhaustive auditing, each audit server needs to be able to simulate client state changes at the same rate it receives new state snapshots. In other words, for each state snapshot that represents Z seconds of changes, the auditor would need to retrieve the previous state snapshot for that node, request and receive the audit log excerpt for that period from the node. It would need to verify the audit log hash matches the committed hash for that interval, and then forward simulate through the events, checking each event for DVE legality, and comparing the submitted final state with the locally simulated final state. If the steady state ratio of clients to servers is (X / Y), then each server needs to be able to simulate at least (X / Y) DVE instances in real-time.

PeerReview auditors have an additional CPU overhead introduced by the requirement to verify authenticators. Based on the figures quoted in section 5.4.3.4, real-time verification of authenticators contained in the logs received from 14 nodes – at 7.5% CPU per log – would saturate a single core. This is probably the biggest single obstacle to assigning multiple clients per auditor in the PeerReview scheme. PeerReview mitigates this by not requiring auditors be trusted, hence enabling (for example) every client to also be an auditor. Unfortunately, creating a quorum of auditors per client multiplies both auditor and client resource requirements.

In general, CPU will probably be the limiting factor for audit servers for both Carbon and PeerReview. Carbon can compensate for this by reducing audit frequency from exhaustive to guided, based on the relative difference between state snapshots, and on audit requests submitted by detectors at client nodes.

5.5 Conclusions

Carbon is a trusted auditing system designed specifically for P2P DVEs. Running the Carbon auditor as a trusted rather than untrusted entity allows a Carbon-enabled system to distribute most operations amongst DVE participants, while retaining good error detection guarantees.

Carbon is able to detect many significant forms of cheating in DVEs, allowing it to serve as part of a P2P DVE's security system. Auditing can detect a variety of common attacks, include illegal state transitions, inconsistent representation of state to peers, and tools such as aimbot proxies which modify inter-node messages. It also obviates collusion as a way of modifying DVE state without detection.

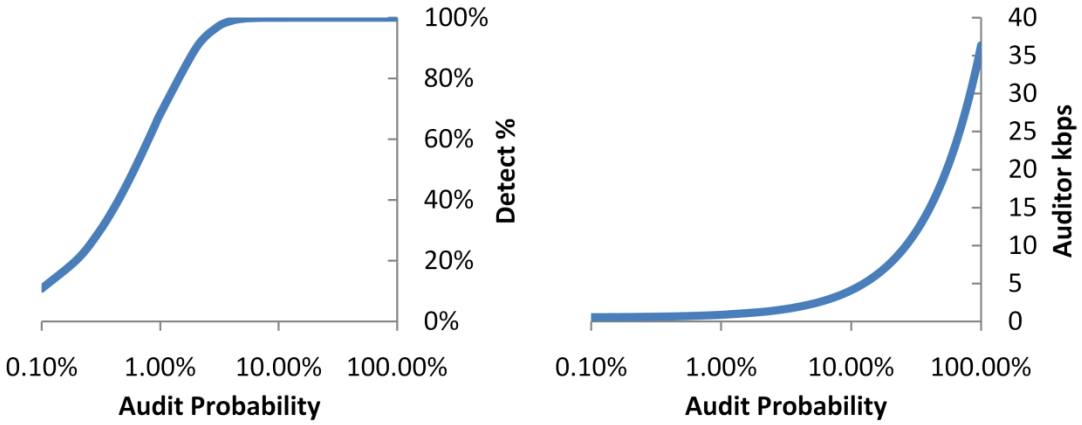

Figure 10: Detection probability and audit bandwidth

Carbon achieves this with a modest amount of client overhead. For example, a P2P DVE which followed eventing and player patterns similar to *World of Warcraft* can have a 99.9% chance of catching an occasional cheater sometime in their lifetime with 7% auditing, assuming a player lifetime of 1,000 hours and cheating once per hour. This level of protection would induce a bandwidth overhead of 3.14 Kbps on total non-audit traffic of 138 Kbps, or 2.3%. Exhaustive auditing would require 37.4 Kbps, a bandwidth overhead of 27%, still a reasonable margin. Figure 10 shows the relationship between audit probability and chance of catching a cheater, and the auditor bandwidth impact of choosing that percentage.

Client CPU overhead would be negligible, and client latency would be unaffected, as Carbon does not modify client-to-client interactions and auditing is performed out of band.

Client storage overhead is also reasonable. Considering a P2P version of *World of Warcraft* as a usage model, Carbon requires 172 MB per week to store a typical players audit logs. The system could be configured with an audit log horizon (such as a week) to minimize client-side storage. Given that a game like *World of Warcraft* can require 5 GB of storage for basic installation, this is less than 4% overhead compared to static installation files.

Carbon performance overhead and security was compared to that provided by PeerReview. In the domain of DVEs, Carbon provides similar protection with less than a tenth of the network traffic, less than a hundredth of the storage requirements, and significantly less CPU than PeerReview requires. This suggests that Carbon is a better choice for auditing DVEs, as it was designed to be.

6 Untrusted collaboration

The final contribution this dissertation makes for DVE security is a straight-forward extension of secure coin flipping, and was presented at NOSSDAV 2009 under the title "Probabilistic event resolution with the pairwise random protocol" [97].

This algorithm is light-weight to implement, and can be used in any scenario where adversaries wish to fairly resolve probabilistic events.

6.1 Introduction

Distributed virtual environments (DVEs) are virtual environment (VE) simulations run on two or more nodes. Nodes are defined as individual software instances contributing to the DVE, usually running on separate computers connected by a network. DVEs usually follow one of two models: client-server or peer-to-peer. Client-server DVEs perform important operations on trusted nodes, and so can typically trust state representation and state transition calculations. Peer-to-peer DVEs, however, distribute more of the state-keeping and transition work to untrusted nodes, requiring additional steps to secure the DVE.

Some DVEs combine deterministic and probabilistic state changes. For example, avatars in *World of Warcraft* (*WoW*) have an arsenal of skills, spells, weapons, and armor at their disposal. The game is balanced so each set of choices – each configuration - is strong against some adversary configurations, and weak against others. Further, the selection of the strategy to compete with opponents, depending upon those opponents' configurations and strategy choices, can make a huge difference in the outcome of battles. Analysis of these combinations and configurations has spawned an entire arm of investigation for *WoW*, called *theorycraft*. The damage dealt by a sensible configuration vs. that dealt by an optimal configuration can easily differ by a factor of two or more. The choice of avatar skills, spells, weapons, armor, and strategy is the deterministic part of competition, and like chess, broad strategies and short-term choices both have significant impact on outcome.

The probabilistic attributes of the DVE also impacts combat outcome. For example, a particular attack from an avatar may be assigned a 4 in 5 chance of successfully hitting an opponent. The damage inflicted per successful strike may be evenly distributed between 100 and 500 points. Averaged over a long combat, the expected value of damage dealt by this attack would be $(4/5) * (100 + (500 - 100)/2) = 240$ points per attack. In the shorter term, however, a player may be extremely unlucky. It is possible – though unlikely - for five attacks to miss in a row, or for attacks which hit to do minimum damage, underperforming expected values.

How can such DVE transactions be successfully resolved between a pair of competitors, when both are incented to cheat, for example to claim they always hit, and they always do maximum damage, or that their opponent always misses? There are a variety of solutions which address resolving the deterministic aspects of competition, but none which address the probabilistic portions.

This chapter outlines a pairwise random protocol (PRP) for untrusted nodes to fairly generate random bit sequences which can be used to resolve probabilistic events. PRP allows adversaries to fairly resolve sequences of actions without requiring intervention from a third party, trusted or otherwise.

The remainder of this chapter presents PRP and analyzes its benefits. Section 6.2 provides a brief overview of DVE security research, and the foundation of bit commitment and secure coin flipping. Section 6.3 presents two variations of PRP. Section 6.4 discusses PRP's attributes and performance compared to a trusted third party (TTP). Section 6.5 presents a final summary.

6.2 Related work

Relevant related work falls into two categories: DVE security research, and secure multi-party computation.

DVE security research covers a variety of different aspects of DVE correctness, but does not generally address fair resolution of probabilistic events without a trusted third party. Details on DVE security research can be found in section 5.1.

Secure multi-party computation is a set of techniques allowing adversaries to calculate a function's value without revealing their contributions to that function. This makes it particularly well suited for adversaries to create a pseudo random bit sequence: the adversaries want to ensure neither can control the values determined by the calculation. If either was able to learn the input of the other during calculation, they could modify their own input to create the desired sequence. By keeping the inputs secret – at least until after the sequence is calculated - we ensure neither party can control the output of a well-formed pseudo-random number generation function.

6.2.1 Secure multi-party computation

Secure multi-party computation (SMC) is a field of cryptography allowing two or more parties to calculate the value of a function of one or more variables without revealing each of their inputs to that function.

One early example of SMC is secure coin flipping, introduced by Blum in [24]. In essence, Blum proposes using a secure one-way function $F(x)$ to enable *Alice* and *Bob* to verifiably flip a fair coin, even though they are adversaries. In the simplest case, this is a three step process, where *Bob* tries to guess if a bit sequence *R* chosen by *Alice* is even or odd. If he is correct, he wins the coin toss. Otherwise he loses.

1. *Alice* chooses a bit vector x, then tells *Bob* $F(x)$.
2. *Bob* tells *Alice* his guess as to whether x is even or odd.
3. *Alice* reveals x to *Bob*.

At the end of the exchange, *Bob* can calculate $F(x)$ to ensure *Alice* did not change x after learning *Bob*'s guess.

Another example of SMC is Yao's solution to the Millionaires' problem [135]. In this example, two millionaires *Alice* and *Bob* each have between 1 and *k* million dollars. They want to find out which is richer, without either revealing the actual amount of their wealth. Yao provides a general solution for comparing two integer values in a limited domain (e.g. *k* is an integer between 1 and 10), and determining which is greater. Yao's solution is complex and expensive, requiring a mutually agreed table of *k* integers, calculation and transformation of that table by each participant in the calculation, and transmission of a derived table by one of the participants. Using such a protocol to create a random bit sequence (for example, a 32-bit sequence which has more than 4 billion unique values) would be expensive and impractical in the scenario of a DVE.

Oblivious transfer (OT) [116] allows *Alice* to send two messages to *Bob*, and be assured that *Bob* will only receive one of them, although *Alice* will not know which one. A refinement to the original OT protocol called one-to-two oblivious transfer [49] is a common component to many SMC implementations, as it can be used to determine a random bit between adversaries. Like Yao's solution to the Millionaires' problem, oblivious transfer is expensive in terms of computation and information transferred.

Of these techniques, secure coin flipping is the lightest weight, and so the one chosen as the basis of PRP.

6.3 Pairwise Random Protocol (PRP)

The pairwise random protocol (PRP) provides a way for two competing nodes in a DVE to fairly resolve probabilistic events.

Consider a DVE with nodes, *Alice* and *Bob*. Each node controls an avatar, and those avatars are interacting. Given a consistent, verifiable view of the simulation state, the goal is to enable *Alice* and

Bob to fairly resolve a set of probabilistic events. For example, *Alice* and *Bob* are engaged in combat, with a certain probability of each successfully attacking their opponent, and a variable amount of damage inflicted per successful attack.

Each node is incented to cheat to resolve actions in their favor. *Alice* wants all of her attacks to succeed, and all of *Bob*'s attacks to fail. *Alice* wants each of her hits to inflict maximum damage, and each of *Bob*'s hits - should he manage to get any - to inflict minimum damage. PRP ensures that - given consistent views of world state - *Alice* and *Bob* can fairly resolve probabilistic interactions such as determining attack success and selecting the amount of damage inflicted within the specified range.

As *Alice* and *Bob* are participating in the same DVE, several simplifying assumptions can be made.

- *Alice* and *Bob* each know the correct DVE rules. Even if *Alice* is running a modified version of the DVE software, she has the unmodified code at her disposal for verifying validity of *Bob*'s activities.
- *Alice* and *Bob* have access to identical pseudo-random number generators, and these generators provide "suitably random" sequences for the DVE to resolve probabilistic sequences of activities.
- *Alice* and *Bob* can communicate with each other.

Given these assumptions, any probabilistic activity which affects either party can be defined as an adversarial activity.

Before resolving the success or failure of an adversarial activity, *Alice* and *Bob* must specify the activity to be decided. For example, *Alice* and *Bob* must agree that they are performing PRP to calculate whether or not *Alice* succeeds in attacking *Bob*. This has two benefits:

1. It ensures that the losing party in a PRP exchange cannot claim the exchange was intended to determine outcome of a different activity, e.g. whether *Alice* gets crumbs on her jacket from eating a donut, rather than success in combat.
2. It allows a cryptographic proof of participation in the activity to be generated. This reduces the utility of the loser refusing to continue the exchange.

This binding can preface the PRP exchange, or be performed as part of it. Discussions of methods for doing this are out of scope of this dissertation.

Section 6.3.1 describes the core PRP protocol to resolve a single probabilistic event. Section 6.3.2 proposes a refinement for generating a pseudo-random sequence without either adversary controlling the sequence.

6.3.1 Resolving a single action

Probabilistic actions can be resolved by a series of secure coin flips with a pre-agreed interpretation. For example, *Alice* and *Bob* can agree that *Alice* has a 5 in 8 chance of successfully attacking *Bob*. *Alice* therefore needs to generate a random number between 1 and 8, and if it is 5 or less, her attack succeeds. *Alice* and *Bob* can generate this number by flipping a fair coin three times to generate a 3-digit binary number, with heads being a 1 and tails a 0. As long as correct sequencing of flip results used as bits is guaranteed, resolving a single arbitrarily scaled probabilistic event – such as this one - can be reduced to ensuring a single coin can be fairly flipped.

The basic protocol for *Alice* and *Bob* to determine a random bit without requiring a trusted third party is described below, and illustrated in Figure 11. Note this exchange is roughly equivalent to Blum's secure coin flip protocol [24].

1. *Alice* and *Bob* each privately choose a bit vector of length 1, B_A and B_B respectively.
2. *Alice* generates a (possibly zero-length) nonce N_A known only to her, and uses a cryptographic hash $H(x)$ to generate a digest $D = H(N_A, B_A)$. She sends D to *Bob*.

3. *Bob* makes a note of *Alice*'s digest D, and sends his bit vector B_B to *Alice*.
4. Upon receipt of *Bob*'s bit vector, *Alice* transmits her nonce N_A and bit vector B_A to *Bob*. *Bob* verifies that the hash of these values $H(N_A, B_A)$ matches the previously received digest D.
5. *Alice* and *Bob* XOR their own bit vector with their adversary's bit vector to determine the outcome of the exchange. In the case of a single-bit bit vector, if $B_A = B_B$ then the result is 0. Otherwise it is 1.

As long as each message is eventually received, and *Alice* chooses a nonce of sufficient size to diversify values for D, *Alice* and *Bob* can be assured that the binary result is fairly determined. It does not matter whether *Alice* and *Bob* randomly or deliberately select their bit vectors. As long as *Alice* and *Bob* are not collaborating, there is a 50% chance of the bit being 1, and a 50% chance it is 0.

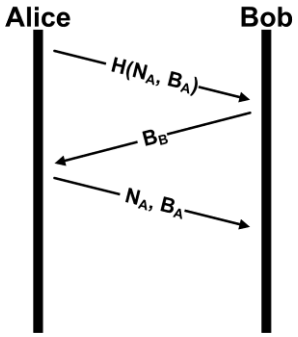

Figure 11: Single bit PRP exchange

Barring retransmissions, a minimum of three messages comprising one-and-a-half round trips are required to complete a single PRP exchange, as shown in Figure 11. If low latency is more important than a low message count, latency can be reduced to a single round trip by adding a message and making the exchange symmetric, as shown in Figure 12. Note that this optimization may open additional attack vectors.

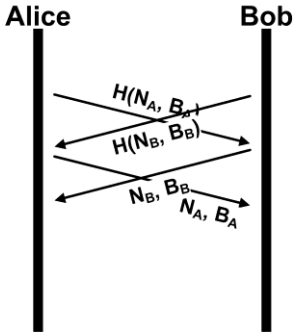

Figure 12: Symmetric single bit PRP exchange

This protocol can be trivially extended to provide an arbitrarily long random bit vector by changing the number of bits in B_A and B_B. For example, rather than performing three sets of exchanges for *Alice* to generate her three-bit random number, she can simply replace B_A with a 3-bit bit vector, and instruct *Bob* to do the same with B_B.

This version of PRP is secure, but requires several network messages for each random value provided. Depending upon the security requirements of the DVE, it is possible to obtain acceptable results with less overhead, as detailed below.

6.3.2 Resolving an unbounded random sequence

Interactions in DVEs are often comprised of long sequences of actions. Requiring a three or four message exchange for each action by each participant is secure and fair, but slow and expensive.

An alternative is to resolve more bits than are required for the current event, and to use the next sequence of unused bits for each subsequent activity. While efficient from a protocol perspective, this extension suffers from a look-ahead vulnerability in terms of consumption. Once *Alice* and *Bob* finish the exchange and determine the bit sequence, neither can change the bits. However, they can modify their behavior to consume the bits in an advantageous way.

For example, suppose *Alice* can execute any of four actions interchangeably: she can tie her shoes (random chance of failure), skip a rock (random number of skips), pick a flower (random length of stem), or build a house (random number of rooms). Each action has a different cost and benefit for *Alice*. If *Alice* knows the sequence of bits which will be consumed to determine the outcome of her probabilistic actions, she can "look ahead" to determine the most favorable sequence to execute. For example, she can pick flowers to consume undesirable bits, waiting to build a house until the next set of bits guarantee she builds a house with the maximum number of rooms.

Another alternative to provide random values for a series of activities is to use PRP to determine a random seed for a pseudo-random generator. *Alice* and *Bob* agree on how the pseudo-random stream will be used, then use PRP to create a bit vector of an appropriate size to seed the generator. Since both *Alice* and *Bob* have copies of the random number generator, they can each validate the sequence generated using the resolved bits seed, and the subsequent results. Note that the idea of using a pseudo-random generator to create a sequence of random numbers which can be verified by all participants is suggested in [79].

6.4 Results

PRP as described in Section 6.3.1 provides a reliable but relatively expensive source of bits to fairly resolve adversarial probabilistic events. Section 6.3.2 describes a less expensive variant of PRP, but at the cost of enabling look-ahead cheats, and allowing participants to unfairly optimize the order of events which consume those bits.

DVE authors should carefully examine impact of look-ahead exploitation before using the random seed or pre-generation approaches to generating bit sequences. Real-time interactive DVEs such as network games may be so dynamic that the look-ahead vulnerability is of no practical concern, especially if the bit stream is refreshed every few seconds. For example, *Alice* may have only a small number of action choices at any given time, and attempting to bias her choice according to attributes of the random bit stream may provide less value – even when done via an automatic enhancement hack - than selecting the most appropriate action at the time.

While PRP performance overhead is greater than the overhead of working directly with a TTP, the cause is not solely the algorithm itself. Distributing activities normally performed by a TTP to unreliable, untrusted nodes can introduce significant overheads to DVE activities, as noted in several of the previously cited DVE security works. Still, informed choices can minimized this overhead. The performance analysis below is provided to help DVE authors understand trade-offs in different PRP usage scenarios.

DVEs often rely upon congruent random generators running on a TTP to determine the outcome of probabilistic actions. For example, *Quake III* uses calls to their random generator to determine variations in projectile direction due to weapon recoil. The random number generator typically used in deployed network games is from the 'C' standard library, and typically provides a two byte random number.

In a game with trusted third parties, the TTP can often produce and consume random numbers locally on behalf of a given client. Alternatively, it can provide the random number to the client for the client to consume in its local activities. In peer-to-peer DVEs, the system should not rely upon a TTP for

common activities. The client should ideally be able to resolve probabilistic events without TTP intervention. For the DVE to be fair, this must be done in a way which does not allow the value to be chosen by the same client consuming it.

The following two subsections discuss PRPs security and performance properties, using this scenario to illustrate those properties.

6.4.1 Security

Most deployed DVEs are implemented as client-server applications. From the client perspective, the server acts as a trusted third party (TTP). The server is explicitly trusted to fairly resolve probabilistic events on behalf of its clients. In other words, over the course of many trials, the client expects the distribution of results to roughly match the probability of each outcome.

Even in the TTP case, outcome can be biased by many factors, such as the source of random numbers for event resolution. Methods for generating suitably random numbers are out of scope of this section. Instead, the goal is to support the proposition that given two adversaries, neither adversary can predictably bias the resulting random bit vector. As long as appropriate precautions are taken, this should provide probabilistic event resolution of a quality no worse than that available from a TTP.

Let *Alice* be a node undertaking PRP to create a random bit vector for her consumption, and *Bob* an adversary participating in that PRP exchange. The goal is to prove that so long as *Alice* and *Bob* cannot predict the value of their adversary's bit vector B, neither can bias the result of the PRP exchange.

To do this, four properties must be proven:

1. Once *Alice* commits to a choice for B_A by transmitting a digest D to *Bob*, she cannot change her choice without detection.
2. *Bob* cannot ascertain *Alice*'s choice of B_A from the digest D.
3. If *Bob* has no knowledge of *Alice*'s choice for B_A, then *Bob* cannot choose a B_B which will bias the result.
4. Given the sequence of messages exchanged in PRP, neither *Alice* nor *Bob* can dispute the value of the resulting bit vector $B_A \oplus B_B$.

P1: *Alice* transmits the digest D, a SHA-256 hash of an input of length at least 256 bits. In this, case input is a 255-bit nonce N_A and *Alice*'s 1-bit bit vector B_A. In order for *Alice* to change her selection of B_A to B'_A after transmitting D to *Bob*, she must find a new nonce N'_A such that $SHA_{256}(N'_A, B'_A) = SHA_{256}(N_A, B_A)$. Since SHA_{256} is not broken, this would require a brute force attack, on average $2^{256}/2$ attempts, which means trying every value for the 255-bit nonce. This is computationally infeasible. Even if every computer on earth were employed and each was capable of testing a million candidates per second, more than 10^{56} years would be required. Alternatively *Alice* could try to find two 256-bit vectors with a different last bit whose hashes collide, but even this would require $\boldsymbol{O}(\sqrt{2^{256}}) = \boldsymbol{O}(2^{128})$ attempts.

P2. Since SHA-256 is an unbroken cryptographic one-way function, and since *Alice* has given it an input of at least 256 bits, there is no way for *Bob* to predict the value of the input solely based upon its output, or to limit that input to a specific candidate pool other than brute-force attack, which as shown above is computationally infeasible.

P3. For each bit in B_B, a 1 will invert *Alice*'s choice for the same bit in the result bit vector, while a 0 will leave *Alice*'s choice intact. Since *Bob* cannot determine the bit chosen by *Alice* for each position in B_A at the time he must commit to B_B, he has no way of choosing a value for B_B to maximize chances of a specific outcome.

P4. For a given input, XOR is a deterministic operation, so $B_A \oplus B_B$ is deterministic for given B_A and B_B. By the time $B_A \oplus B_B$ can be calculated by either *Alice* or *Bob*, both are committed to their bit vector values, and cannot change that commitment without detection from their adversary.

Like most protocols, PRP in its basic form is vulnerable to abort attacks, e.g. *Bob* refusing to acknowledge receipt of *Alice*'s final PRP message after he determines the resulting bit vector does not yield his desired outcome. This can be mitigated by standard cryptographic techniques such as signing each message in the protocol, and using anti-replay and sequencing protections to prove message sequence order and contents. Such mitigations are especially important if the PRP variation in Figure 12 is used.

6.4.2 Performance

Suppose *Alice* wishes to generate a single 16-bit random number for consumption for a pre-agreed purpose. Suppose as well that *Alice* has 100 ms round trip time (RTT) to *Bob* on the network.

Table 16 compares the network latency and traffic required for *Alice* to obtain this random bit vector. Assume IPv4 UDP on Ethernet as the transport medium, inducing transport overhead of 42 bytes per packet. PRP uses SHA-256 as the one-way hash, and a nonce size equal to (hash length – target bit vector size) for bit vectors smaller than the hash value size.

PRP requires one and a half round trips, with each packet containing 42 bytes of headers. The first packet contains the SHA-256 hash of *Alice*'s 240-bit nonce and 16-bit bit vector. The second packet contains *Bob*'s 16-bit bit vector. The final packet contains *Alice*'s nonce and bit vector, and completes the PRP protocol transaction.

	TTP	**Adversary**	**Additional cost**
Latency	50 ms	150 ms	300%
Network Bytes	44 bytes	192 bytes	436%

Table 16: Random number generation cost

The cost in terms of latency and network bytes for the PRP protocol version described in section 6.3.1 is significant compared to obtaining the random bit vector directly from a TTP. Fortunately there are a few ways overhead can be decreased without realistically compromising security.

First, reduce the number of bytes transmitted in payloads by reducing the size of the transmitted hash, and of the nonce itself. PRP uses SHA-256 because SHA-256 is not yet broken, rather than because 256 bits of protection are required. Secrets in a PRP exchange are short-lived – less than a second in the example above – so the hash value only requires enough bits to prevent an attacker from determining *Alice*'s bit vector before it is revealed in *Alice*'s second message. The most significant threat is a dictionary attack, because of its short execution time.

In the 16-bit bit vector case, it would be trivial for *Bob* to create a dictionary with the SHA-256 hash values for the 2^{16} possible values for *Alice*'s bit vector. To prevent this, include a large nonce in the hash to make lookup impractical for *Bob*. Establish a size of lookup table to defeat – for example one petabyte – and choose a hash and nonce size to enable that level of protection. A petabyte is approximately 2^{58} bits. Each entry in a sparse lookup table would include the lookup hash value and the expected 16-bit bit vector. If the hash is truncated to 64-bits, then each lookup table entry would consume 80 bits, resulting in a table capacity of about 2^{52} entries. With a nonce size of 48 bits (and a 16-bit bit vector), this would give *Bob* a probability of about $2^{52} / 2^{64}$ = 0.02% of successfully looking up *Alice*'s bit vector from the hash in her first PRP packet with a petabyte index. This optimization reduces the network bytes required for a 16-bit bit vector PRP exchange from 192 bytes to 144 bytes, dropping the network cost from 436% of TTP transaction cost to 327%.

Another way to improve both latency and network overhead associated with PRP – though at the cost of some security - is to pre-calculate a large bit vector for consumption, and then use successive parts of that vector for the next *k* random contests. For example, suppose *Alice* needs an average of ten 16-bit random values per second. She can request a bit vector with enough bits to satisfy five seconds of her requirements, or $50 * 16 = 800$ random bits. For a request this large, assuming a sufficiently random input bit vector on *Alice*'s side, a nonce is no longer needed. Pre-calculating a series of random values amortizes PRP latency across several seconds of bit consumption, reducing its effective performance impact. It also reduces the relative overhead of generating the 800 random bits. Total PRP network byte cost – assuming 64-bit truncated hash - is 340 bytes, which compared with a TTP-sent packet size of $44 + (800/8) = 144$ bytes is 236% more, less overhead than the previously listed optimizations.

A slightly weaker choice would be to use PRP to create the 32-bit seed for *Alice*'s DVE random number generator, and have *Alice* use the resulting pseudo-random sequence for a set interval or number of operations. This approach would consume 46 bytes to obtain the seed from a TTP, or 146 bytes using PRP. While the relative overhead in this case is still more than 300% greater than obtaining the seed from a TTP, the absolute cost to the DVE for generating e.g. 800 pseudo-random bits is quite low.

6.5 Conclusions

This section presented the Pairwise Random Protocol (PRP), based on secure coin flipping. Using PRP, adversaries can fairly determine and agree upon the outcome of probabilistic actions. Three different variations of PRP were presented, along with high-level performance analysis of the algorithms. The variations range from a perfectly fair approach which requires a three-way handshake per random event, to creating arbitrarily long pseudo-random sequences using a fairly determined random seed, up to the tolerance of the DVE.

PRP makes it possible for adversaries to fairly determine the results of probabilistic events in a DVE with the same security a trusted third party – such as a game server – could provide. For DVEs which do not frequently need random numbers, or which are tolerant of the 2 to 4 times overhead required for the most secure versions of PRP, this can be done without loss of fairness or security. If the DVE is performance-sensitive, then compromises can be used such as pre-generating a set of random bits to use over time, or seeding a random number generator, which allow reasonable security without significant performance impact.

7 Conclusions and future work

As computers increase in power and the world becomes better connected, Distributed virtual environments (DVEs) have the opportunity to reach a larger audience. Cyberspace as proposed in the 80s and 90s is still a long ways away, but the potential is clear in today's online network games and social experience such as *Second Life*.

Tens of millions of people use DVEs, but the scope of the experience those DVEs offer has technical limitations. The most popular DVEs – such as *World of Warcraft* and *Second Life* – limit mutual interaction to less than a thousand of their millions of users. Experiences like full-scale virtual battles and large-scale performances remain far out of reach.

Enabling such experiences requires DVEs to scale beyond their current limitations. Decentralizing DVE execution – beyond the degree it already has been – is a promising direction for investigation. This dissertation provides techniques useful for decentralizing DVE execution, and also provides guidance on techniques which are not feasible in today's Internet.

In preparing for this dissertation, I investigated the state of the art in DVE scalability research, especially in terms of decentralizing DVE operation. I found many promising approaches, none of which have been adopted by broadly distributed DVEs. As I examined the research and its evaluations, I identified three possibilities as to why these approaches are not in use today:

1. **Applicability of evaluation**. Most of the research proposals include evaluation against researcher-proposed workloads, without any clear link to broadly adopted real-world systems. Could proof of applicability be the missing key preventing DVE operators from investing in developing commercial DVEs based upon the research?
2. **Deployability**. Algorithms which work well in numerical simulations and in well-provisioned computer labs perform differently in the real world, often failing catastrophically when deployed. Does today's Internet infrastructure allow the possibility of deployment to consumers?
3. **Security**. Many of the decentralization proposals make the simplifying assumption that security will be handled separately. However, there is no obvious system available to meet security needs, even for something as simple as fair resolution of probabilistic events between adversaries. And, as online game providers have found, security is paramount for DVE adoption and longevity. Are there tractable mechanisms for securing decentralized DVEs?

This dissertation provides answers for each of these three questions.

In chapter 3, I analyzed network traffic and avatar movement in a subset of the world's most popular DVE, *World of Warcraft* (*WoW*). I characterized the movement of player avatars in a highly interactive scenario, player-vs-player battles in battlegrounds. I compared my findings to assumptions earlier research has made about avatar movement in DVEs, and found those assumptions to be inconsistent with observed avatar behavior. I found that a waypoint model is not sufficient to characterize avatar movement. To my surprise and despite clear incentives for players to group when moving, I found they did not generally do so. Finally, I found that hotspots are a factor in avatar movement and clustering, though additional research is required to employ this as a basis for DVE evaluation.

In chapter 4, I extended my instrumentation and data processing, and captured data for the five main scenarios in *WoW*. I wrote a simulator which honored upload and download bandwidth limitations and node latency. I modeled TCP windowing behavior between nodes to take into account stream contention and latency impact on data throughput. I parsed the *WoW* captures, attributed the majority of messages in my captures to avatars, and determined the positions of those avatars. I used the avatar positions and their messages to drive simulation of an area of interest message propagation scheme for each of the five *WoW* play scenarios. I created node bandwidth and latency models based upon published real-world measurements of ISPs serving 90% of the UK's population.

I evaluated two models for message propagation: the existing client-server model, and an ideal P2P model with zero overhead and perfect knowledge. Simulated client-server traffic was consistent with my in-game measurements, as expected. I found to my dismay that despite the relative efficiency of the *WoW* network protocol (less than 10 Kbps up and down on average for a client in the client-server architecture), even an idealized P2P implementation could not keep up with peak – and in some cases, average – bandwidth requirements for most scenarios. I concluded that to be deployable in today's Internet, decentralized DVE solutions must implement either a client-server or hybrid model, not a pure peer-to-peer model.

In chapter 5, I examined the security needs of DVEs, and proposed the auditing system Carbon to meet those needs. Carbon is predicated on the existence of trusted auditor nodes within the DVE, but requires no trust between participant (player) DVE nodes. Transactions between DVE participants can be handled efficiently, without being gated on real-time trusted third party interaction. Participants locally store a log of their activities, periodically submitting log summaries to trusted auditors. They can request audits of other participants whom they suspect of cheating, and must honor audit requests made by the trusted auditor by providing requested log excerpts.

Carbon's architecture is comparable to the more general PeerReview system (the two are contrasted in section 5.4.2), but has been tuned using assumptions relevant to DVEs. This tuning reduces overall security in ways I believe are not significant for the DVE scenario, and dramatically reduces overhead requirements. When analyzed against *WoW's* attributes, Carbon is able to offer exhaustive auditing with only 27% client bandwidth overhead, and 4% client storage overhead. If it is acceptable to have a 99.9% chance of catching an occasional cheater *sometime* in their avatar's lifetime rather than on a single instance of cheating, Carbon can operate with 7% client bandwidth overhead. Carbon's overall resource requirements are at least an order of magnitude less than those required by PeerReview.

Finally, chapter 6 proposes a "Pairwise Random Protocol" (PRP) to allow untrusted adversaries to resolve probabilistic events fairly between themselves. Most DVE auditing and security proposals assume deterministic behaviors in the DVE. For example, behavior similar to chess where the same move always yields the same immediate result. These systems do not support transactions where the outcome is probabilistic, e.g. a 25% chance of one avatar successfully tickling another. PRP is a simple extension of an established technology called secure coin flipping. PRP allows adversaries to fairly determine probabilistic events by generating a reproducible pseudo-random bit stream. I provide analysis showing different ways PRP can be used, from a three-way handshake to determine each single bit, to generating a random seed which both sides will use in a pre-agreed manner.

These four contributions provide insight into each of the three questions posited earlier as follows:

1. **Applicability of evaluation**. Previous evaluation models do not correlate well with the DVE I examined, *WoW*. I believe *WoW* is generalizable to most broadly deployed MMOGs. I recommend using real-world data captures, or proving a given evaluation model is consistent with these real-world captures.
2. **Deployability**. DVE architectures can be divided into client-server, peer-to-peer, and hybrid (client-server with some peer contribution). Peer-to-peer is not a deployable architecture for DVEs in 2010s Internet. DVE providers developing DVEs for release in the near future should continue to focus on client-server architectures, or investigate hybrid architectures.
3. **Security**. I provide two security contributions which I believe can enhance security and fairness of decentralized DVE solutions.

In reviewing the contributions of this dissertation, I believe I have substantially added to the community's knowledge of DVE scalability, and to the options available for securing DVEs. Three of my four chapters have been peer-reviewed and published. The fourth – Carbon – has been available as a technical report for more than a year.

There are many directions which should be explored for future work.

World of Warcraft has been used as the exemplar for much of this research. It would be interesting to evaluate other DVEs to see whether findings hold true for all goal-oriented DVEs. For example, do

distributed military simulations follow similar avatar movement patterns? I expect not, but then, results from *WoW* were surprising as well. What about the latest generation of games, such as *Aion Online*? These appear similar to *WoW*, but are newer with higher system-wide demands, and may have different play characteristics as well.

For decentralized systems, understanding the state of residential broadband worldwide, and then identifying trends in improvement and potential growth would be useful for determining when we might hope to see P2P DVEs be deployed on a broad scale. Mitigations for dealing with peak demands in avatar density could bring that time closer to the present. Creating "peer assisted" hybrid solutions where centralized resources can opportunistically shift load to peers would allow an architecture to be deployed and slowly "turned on" as network resources improve, and seems a very promising and relevant direction for research.

For auditing, a deeper security analysis would be welcome. Can automated examination of audit results guide audit targets, providing a way to enable partial auditing to provide security guarantees similar to those exhaustive auditing provides? Can social networking and player reputations suggest ways to allow players to request audits without creating a new attack vector? What is the social and commercial tolerance for time and correctness in identifying and punishing cheating?

Distributed virtual environments enable productivity and provide enjoyment to millions of people worldwide. I hope this dissertation contributes to bridging the gap between research results and the lives of those people, and that research continues so we can do bigger and better things moving forward!

Bibliography

1 ACTIVISION. Call of Duty(R): Modern Warfare(R) 2 Sets All-Time Entertainment Industry Record Grossing an Estimated $550 Million Worldwide in First Five Days. *Activision Press Release*, http://investor.activision.com/releasedetail.cfm?releaseid=425018 (Nov. 18, 2009).

2 ACTIVISION. *Call of Duty: Black Ops*. http://www.callofduty.com/blackops.

3 Aggarwal, Sudhir, Christofoli, Justin, Mukherjee, Sarit, and Rangarajan, Sampath. Authority assignment in distributed multi-player proxy-based games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (2006), ACM.

4 AKAMAI TECHNOLOGIES. *Akamai: The Leader in Web Application Accelleration and Performance Management, Streaming Media Services and Content Delivery*. http://www.akamai.com/.

5 AKAMAI TECHNOLOGIES. *The State of the Internet*. Akamai Technologies, 2010.

6 ANONYMOUS AUTHOR. *Unreal*. http://en.wikipedia.org/wiki/Unreal.

7 Anthes, Christoph, Heinzlreiter, Paul, and Volkert, Jens. An adaptive network architecture for close-coupled collaboration in distributed virtual environments. In *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry* (2004), ACM, 382-385.

8 Armitage, Grenville. An experimental estimation of latency sensitivity in multiplayer Quake 3. In *The 11th IEEE International Conference on Networks, 2003* (2003), IEEE, 137-141.

9 Armitage, Grenville, Claypool, Mark, and Branch, Philip. *Network and Online Games: Understanding and Engineering Multiplayer*. Wiley, 2006.

10 Backhaus, Helge and Krause, Stefan. Voronoi-based adaptive scalable transfer revisited: gain and loss of a Voronoi-based peer-to-peer approach for MMOG. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (2007), ACM, 49-54.

11 Bauer, Daniel, Rooney, Sean, and Scotton, Paolo. Network infrastructure for massively distributed games. In *Proceedings of the 1st workshop on Network and system support for games* (2002), ACM, 36-43.

12 Baughman, Nathaniel E. and Levine, Brian Neil. Cheat-proof playout for centralized and distributed online games. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies* (2001), IEEE, 104-113.

13 Baughman, Nathaniel E., Liberatore, Marc, and Levine, Brian Neil. Cheat-proof Playout for Centralized and Peer-to-Peer Gaming. *IEEE/ACM Transactions on Networking*, vol. 15, iss. 1 (2006), 1-13.

14 Beeharee, Ashweeni Kumar, West, Adrian J., and Hubbold, Roger. Visual attention based information culling for Distributed Virtual Environments. In *Proceedings of the ACM symposium on Virtual reality software and technology* (2003), ACM, 213-222.

15 BEEPA PTY LTD. *FRAPS show fps, record video game movies, screen capture software*. http://www.fraps.com/.

16 Beigbeder, Tom, Coughlan, Rory, Lusher, Corey, Plunkett, John, Agu, Emmanuel, and Claypool, Mark. The effects of loss and latency on user performance in unreal tournament 2003. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games* (2004), ACM, 144-151.

17 Bharambe, Ashwin, Douceur, John R., Lorch, Jacob R., Moscibroda, Thomas, Pang, Jeffrey, Seshan, Srinivasan, and Zhuang, Xinyu. Donnybrook: Enabling Large-Scale, High-Speed. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication* (2008), ACM, 389-400.

18 Bharambe, Ashwin, Pang, Jeffrey, and Seshan, Srinivasan. Colyseus: a distributed architecture for online multiplayer games. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation* (2006), USENIX Association.

19 BLIZZARD ENTERTAINMENT. *Blizzard Entertainment: Warcraft III*. http://us.blizzard.com/en-us/games/war3/.

20 BLIZZARD ENTERTAINMENT. *Starcraft*. http://us.blizzard.com/en-us/games/sc/.

21 BLIZZARD ENTERTAINMENT. *World of Warcraft Community Site*. http://us.battle.net/wow/en/.

22 BLIZZARD ENTERTAINMENT. World of Warcraft Subscriber Base Reaches 12 Million Worldwide. *Blizzard.Com*, http://us.blizzard.com/en-us/company/press/pressreleases.html?101007 (Oct. 21, 2010).

23 BLIZZARD GAMES. *Diablo 2*. http://us.blizzard.com/en-us/games/d2/.

24 Blum, Manuel. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, vol. 15, iss. 1 (1983), 23-27.

25 Boulanger, Jean-Sébastien, Kienzle, Jörg, and Verbrugge, Clark. Comparing interest management algorithms for massively multiplayer games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (2006), ACM.

26 Bowery, Jim. *Spasim (1974) The First First-Person-Shooter 3D Multiplayer Networked Game*. http://web.archive.org/web/20010410145350/http://www.geocities.com/jim_bowery/spasim.html. 2001.

27 Brun, Jeremy, Safaei, Farzad, and Boustead, Paul. Managing latency and fairness in networked games. *Communications of the ACM*, vol. 49, iss. 11 (November 2006), 46-51.

28 Buro, Michael. ORTS: A Hack-Free RTS Game Environment. In *Proceedings of the Third International Conference on Computers and Games* (2003), ACM, 156-161.

29 BUTTERFLY. *Digital Media: Massively Multiplayer Online Gaming (MMOG)*. Intel, Butterfly.net. 2003.

30 Castro, M., Druschel, P., Kermarrec, A., and Rowstron, A. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *Journal on Selected Areas in communications*, vol. 20, iss. 8 (2002), 1489-1499.

31 Castro, Miguel and Liskov, Barbara. Practical Byzantine fault tolerance. In *Proceedings of the third symposium on Operating systems design and implementation* (1999), USENIX Association, 173-186.

32 Cecin, F. R., Real, R., Oliveira, R. de, Resin, C. F., Martins, M. G., and Victoria, J. L. A Scalable and Cheat-Resistant Distribution Model for Internet Games. In *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications* (2004), IEEE, 83-90.

33 Chambers, Chris, Feng, Wu chang, and Feng, Wu chi. Towards public server MMOs. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (2006), ACM.

34 Chambers, Chris, Feng, Wu-chang, Feng, Wu-chi, and Saha, Debanjan. Mitigating information exposure to cheaters in real-time strategy games. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video* (2005), ACM, 7-12.

35 Chan, Luther, Yong, James, Bai, Jiaqiang, Leong, Ben, and Tan, Raymond. Hydra: A Massively-Multiplayer Peer-to-Peer Architecture for the Game Developer. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (2007), ACM, 37-42.

36 Chen, Kuan-Ta and Hong, Li-Wen. User identification based on game-play activity patterns. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (), ACM, 7-12.

37 Chen, Kuan-Ta, Huang, Polly, and Lei, Chin-Laung. How Sensitive are Online Gamers to Network Quality? *Communications of the ACM*, vol. 49, iss. 11 (November 2006), 34-38.

38 Chen, Kuan-Ta, Pao, Hsing-Kuo Kenneth, and Chang, Hong-Chung. Game bot identification based on manifold learning. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games* (2008), ACM, 21-26.

39 Claypool, Mark and Claypool, Kajal. Latency and Player Actions in Online Games. *Communications of the ACM*, vol. 49, iss. 11 (November 2006), 40-45.

40 Dahmann, Judith S., Fujimoto, Richard M., and Weatherly, Richard M. The Department of Defense High Level Architecture. In *Proceedings of the 29th conference on Winter simulation* (1997), IEEE Computer Society, 142-149.

41 DeLap, Margaret, Knutsson, Björn, Lu, Honghui, Sokolsky, Oleg, Sammapun, Usa, Lee, Insup, and Tsarouchis, Christos. Is runtime verification applicable to cheat detection? In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games* (2004), ACM, 134-138.

42 Dibbell, Julian. The Decline and Fall of an Ultra Rich Online Gaming Empire. *Wired Magazine*, vol. 16, iss. 12 (November 2008).

43 Dick, Matthias, Wellnitz, Oliver, and Wolf, Lars. Analysis of factors affecting players' performance and perception in multiplayer games. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games* (2005), ACM, 1-7.

44 Ding, Dawei and Zhu, Miaoling. A model of dynamic interest management: interaction analysis in collaborative virtual environment. In *Proceedings of the ACM symposium on Virtual reality software and technology* (2003), ACM, 223-230.

45 Douglas, D. H. and Peucker, T. K. Algorithms for the Reduction of the Number of Points Required to Represent a Line or its Caricature. *The Canadian Cartographer*, vol. 10, iss. 2 (1973), 112-122.

46 Duong, Ta Nguyen Binh and Zhou, Suiping. A dynamic load sharing algorithm for massively multiplayer online games. In *The 11th IEEE International Conference on Networks* (2003), IEEE, 131-136.

47 EA GAMES. *Medal of Honor: Allied Assault*. http://www.ea.com/uk/game/medal-of-honor-allied-assault.

48 ELECTRONIC ARTS, INC. *Medal of Honor*. http://www.medalofhonor.com/.

49 Even, Shimon, Goldreich, Oded, and Lempel, Abraham. A randomized protocol for signing contracts. *Communications of the ACM*, vol. 28, iss. 6 (June 1985), 637-647.

50 Fan, Lu, Taylor, Hamish, and Trinder, Phil. Mediator: A Design Framework for P2P MMOGs. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (2007), ACM, 43-48.

51 Feng, Wu-chang, Kaiser, Ed, and Schluessler, Travis. Stealth measurements for cheat detection in on-line games. In *Proceedings of the 7th ACM SIGCOMM workshop on Network and system support for games* (2008), ACM, 15-20.

52 Ferretti, Stefano and Roccetti, Marco. Game time modelling for cheating detection in P2P MOGs: a case study with a fast rate cheat. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (2006), ACM.

53 FIRAXIS GAMES. *Sid Meier's Civilization - Official Site*. http://www.civilization.com/.

54 FLAGSHIP INDUSTRIES, INC. *Ventrilo - Surround Sound Voice Communication Software*. http://www.ventrilo.com/.

55 Fritsch, Tobias, Ritter, Hartmut, and Schiller, Jochen. The effect of latency and network limitations on MMORPGs: a field study of everquest2. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games* (2005), ACM, 1-9.

56 Fung, Yeung Siu. Hack-proof synchronization protocol for multi-player online games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (2006), ACM.

57 GauthierDickey, Chris, Zappala, Daniel, Lo, Virginia, and Marr, James. Low latency and cheat-proof event ordering for peer-to-peer games. In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video* (2004), ACM, 134-139.

58 Goodman, Josh and Verbrugge, Clark. A peer auditing scheme for cheat elimination in MMOGs. In *Proceedings of the 7th ACM SIGCOMM workshop on Network and system support for games* (2008), ACM, 9-14.

59 Gorawski, Marcin and Stachurski, Karol. A Secure Event Agreement (SEA) protocol for peer-to-peer games. In *Proceedings of the First International Conference on Availability, Reliability and Security* (2006), IEEE Computer Society, 34-41.

60 Haeberlen, Andreas, Kouznetsov, Petr, and Druschel, Peter. PeerReview: practical accountability for distributed systems. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles* (2007), ACM, 175-188.

61 Henderson, Tristan and Bhatti, Saleem. Networked games: a QoS-sensitive application for QoS-insensitive users? In *Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS: What have we learned, why do we care?* (2003), ACM, 141-147.

62 Hernan, Shawn, Lambert, Scott, Ostwald, Tomasz, and Shostack, Adam. Uncover Security Design Flaws Using The STRIDE Approach. *MSDN Magazine* (November 2006).

63 Hikichi, Kenji, Yasuda, Yasuhiko, Fukuda, Akifumi, and Sezaki, Kaoru. The effect of network delay on remote calligraphic teaching with haptic interfaces. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (2006), ACM.

64 Hoglund, Greg and McGraw, Gary. *Exploiting Online Games: Cheating Massively Distributed Systems*. Addison-Wesley Professional, 2007.

65 Howard, Michael and Leblanc, David E. *Writing Secure Code*. Microsoft Press, 2002.

66 Hu, Shun-Yun and Liao, Guan-Ming. Scalable peer-to-peer networked virtual environment. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games* (2004), ACM, 129-133.

67 ID SOFTWARE. id History. *id Games Website*, http://www.idsoftware.com/business/history/ (Sep. 22, 2010).

68 ID SOFTWARE. *Quake*. http://www.idsoftware.com/games/quake/quake/.

69 IEEE standard for distributed interactive simulation - application protocols. *IEEE Std 1278.1-1995* (1996).

70 IEEE standard for distributed interactive simulation - application protocols. *IEEE Std 1278.1a-1998* (1998).

71 IEEE Standard for Information Technology - Protocols for Distributed Interactive Simulations Applications. Entity Information and Interaction. *IEEE Std 1283-1993* (1993).

72 Iimura, Takuji, Hazeyama, Hiroaki, and Kadobayashi, Youki. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games* (2004), ACM, 116-120.

73 INTERNATIONAL GAME DEVELOPERS ASSOCIATION. *2004 Persistent Worlds Whitepaper*. http://www.igda.org/online/IGDA_PSW_Whitepaper_2004.pdf. 2004.

74 Jardine, Jared and Zappala, Daniel. A hybrid architecture for massively multiplayer online games. In *Proceedings of the 7th ACM SIGCOMM workshop on Network and system support for games* (2004), ACM, 60-65.

75 Kabus, Patric, Terpstra, Wesley W., Cilia, Mariano, and Buchmann, Alejandro P. Addressing cheating in distributed MMOGs. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games* (2005), ACM, 1-6.

76 Keller, Joaquín and Simon, Gwendal. Solipsis: a massively multi-participant virtual world. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications* (2003), CSREA Press, 262-268.

77 Ki Junbaek, Cheon, Hee Jung, Kang, Jeong-Uk, and Kim, Dogyun. Taxonomy of online game security. *Electronic Library, The*, vol. 22, iss. 1 (2004), 65-73.

78 Kinicki, James and Claypool, Mark. Traffic analysis of avatars in Second Life. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (2008), ACM, 69-74.

79 Knutsson, Björn, Lu, Honghui, Xu, Wei, and Hopkins, Bryan. Peer-to-Peer Support for Massively Multiplayer Games. In *INFOCOM 2004: Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies* (2004), IEEE.

80 Krause, Stephan. 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* (2008), ACM, 28-33.

81 Lang, Tanja, Branch, Philip, and Armitage, Grenville. A synthetic traffic model for Quake3. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology* (2004), ACM, 233-238.

82 Laurens, Peter, Paige, Richard F., Brooke, Phillip J., and Chivers, Howard. A Novel Approach to the Detection of Cheating in Multiplayer Online Games. In *Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems* (2007), IEEE Computer Society, 97-106.

83 Lee, Kyungmin and Lee, Dongman. A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution. In *Proceedings of the ACM symposium on Virtual reality software and technology* (2003), ACM, 160-168.

84 Lee, Dongman, Lim, Mingyu, and Han, Seunghyun. ATLAS: a scalable network framework for distributed virtual environments. In *Proceedings of the 4th international conference on Collaborative virtual environments* (2002), ACM, 47-54.

85 LINDEN LABS. *Land - Second Life Wiki*. http://wiki.secondlife.com/wiki/Land.

86 LINDEN LABS. *Second Life: Official site of the 3D online virtual world*. http://secondlife.com/.

87 Lin, Shiding, Pan, Aimin, Guo, Rui, and Zhang, Zheng. Simulating Large-Scale P2P Systems with the WiDS Toolkit. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (2005), IEEE Computer Society, 415-424.

88 LLC MDY INDUSTRIES. *Glider*. 2008.

89 Lui, John C.S. and Chan, M. F. An efficient partitioning algorithm for distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, iss. 3 (2002), 193-211.

90 Matsumoto, Nobutaka, Kawahara, Yoshihiro, Morikawa, Hiroyuki, and Aoyama, Tomonori. A scalable and low delay communication scheme for networked virtual environments. In *IEEE Global Telecommunications Conference Workshops* (2004), IEEE, 529-535.

91 Mauve, Martin. How to Keep a Dead Man from Shooting. (Enschede 2000), Springer, 199-204.

92 Mauve, Martin, Vogel, Jürgen, Hilt, Volker, and Effelsberg, Wolfgang. Local-lag and timewarp: providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia*, vol. 6, iss. 1 (Feb. 2004), 47-57.

93 McLeroy, Carrie. History of Military gaming. *Soldiers Magazine*, vol. 63, iss. 9 (August 2008), 4-6.

94 MICROSOFT. *Gears of War*. http://gearsofwar.xbox.com/en/.

95 Miller, John L. and Crowcroft, Jon. Avatar movement in World of Warcraft battlegrounds. In *Proceedings of the 8th annual workshop on Network and systems support for games* (2009), IEEE, 1-6.

96 Miller, John L. and Crowcroft, Jon. *Carbon: trusted auditing for P2P distributed virtual environments*. Technical Report UCAM-CL-TR-753, University of Cambridge, Cambridge, 2009.

97 Miller, John L. and Crowcroft, Jon. Probabilistic event resolution with the pairwise random protocol. In *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video* (2009), ACM, 67-72.

98 Miller, Duncan C. and Thorpe, Jack A. SIMNET: the advent of simulator networking. *Proceedings of the IEEE*, vol. 83, iss. 8 (August 1995), 1114-1123.

99 Mitterhofer, Stefan, Kruegel, Christopher, Kirda, Engin, and Platzer, Christian. Server-Side Bot Detection in Massively Multiplayer Online Games. *IEEE Security and Privacy*, vol. 7, iss. 3 (May 2009), 29 - 36.

100 Mogaki, Shunsuke, Kamada, Masuru, Yonekura, Tatsuhiro, Okamoto, Shusuke, Ohtaki, Yasuhiro, and Reaz, Mamun Bin Ibne. Time-stamp service makes real-time gaming cheat-free. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (2007), ACM, 135-138.

101 Mönch, Christian, Grimen, Gisle, and Midtstraum, Roger. Protecting online games against cheating. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (2006), ACM.

102 Morillo, Pedro, Orduna, Juan M., Fernandez, Marcos, and Duato, Jose. Improving the Performance of Distributed Virtual Environment Systems. *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, iss. 7 (2005), 637-649.

103 Morse, Katherine L. *Interest Management in Large-Scale Distributed Simulations*. Technical Report ICS-TR-96-27, University of California, Irvine, 1996.

104 Muffeeehhh. *Anti-Cheat Reloaded - an Anti-Cheat Revolution*. http://www.counter-hack.net/content.php?page=article_reloaded. 2005.

105 MYTHIC ENTERTAINMENT. *Ultima Online*, http://www.uoherald.com.

106 NIELSEN COMPANY. Nielsen Insights - Video Games. *Nielsen | Video Games*, http://en-us.nielsen.com/content/nielsen/en_us/insights/rankings/video_games.html (June 2010).

107 OFCOM. *UK Broadband Speeds 2009*. http://stakeholders.ofcom.org.uk/market-data-research/telecoms-research/broadband-speeds/broadband_speeds/, 2009.

108 Pagdin, Frances A. and Taylor, Ian C. *Virtual Reality - a new therapeutic medium*. http://members.kabsi.at/t01/twa/article.html. 2008.

109 Pang, Jeffrey, Uyeda, Frank, and Lorch, Jacob R. Scaling Peer-to-Peer Games in Low-Bandwidth Environments. In *Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS)* (2007), USENIX.

110 Pittman, Daniel and GauthierDickey, Chris. A measurement study of virtual populations in massively multiplayer online games. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (2007), ACM, 25-30.

111 Ploss, Alexander, Wichmann, Stefan, Glinka, Frank, and Gorlatch, Sergei. From a single- to multi-server online game: a Quake 3 case study using RTF. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology* (2008), ACM, 83-90.

112 Pope, Arthur. *The SIMNET Network and Protocols*. BBN Report No. 7102, BBN Systems and Technologies Advanced Simulation Division, Cambridge, 1989.

113 Pritchard, Matt. How to Hurt the Hackers: The Scoop on Internet Cheating and How You Can Combat It. *Gamasutra*, http://www.gamasutra.com/features/20000724/pritchard_pfv.htm (July 2000).

114 Quax, Peter, Dierckx, Jeroen, Cornelissen, Bart, Vansichem, Gert, and Lamotte, Wim. Dynamic server allocation in a real-life deployable communications architecture for networked games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games* (2008), ACM, 66-71.

115 Quax, Peter, Monsieurs, Patrick, Lamotte, Wim, De Vleeschauwer, Danny, and Degrande, Natalie. Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games* (2004), ACM, 152-156.

116 Rabin, Michael O. *How to exchange secrets with oblivious transfer*. Technical Report TR-81, Aiken Computation Lab, Harvard University, Cambridge, 1981.

117 Rowstron, Antony and Druschel, Peter. 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* (2001), Springer-Verlag, 329-350.

118 Rueda, Silvia, Morillo, Pedro, and Orduna, Juan M. A Saturation Avoidance Technique for Peer-to-Peer Distributed Virtual Environments. In *Proceedings of the 2007 International Conference on Cyberworlds* (2007), IEEE Computer Society, 171-178.

119 Schloss-Griffin, Helen. *The State of Cheating in Online Multiplayer Games*. http://articles.gameapex.com/gaming/article_the_state_of_cheating_in_online_multiplay.php. 2006.

120 Schluessler, Travis, Goglin, Stephen, and Johnson, Erik. Is a bot at the controls?: Detecting input data attacks. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (2007), ACM Press, 1-6.

121 Singhal, Sandeep and Zyda, Michael. *Networked virtual environments : design and implementation*. ACM Press SIGGRAPH Series, New York, 1999.

122 SONY ONLINE ENTERTAINMENT. *Everquest II MMORPG - Official Game Site with free sign-up*. http://www.everquest2.com/.

123 St. John, Aaron and Levine, Brian Neil. Supporting P2P gaming when players have heterogeneous resources. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video* (2005), ACM, 1-6.

124 Suznjevic, Mirko, Dobrijevic, Ognjen, and Matijasevic, Maja. MMORPG Player actions: Network performance, session patterns and latency requirements analysis. *Multimedia Tools and Applications* , vol. 45, iss. 1-3 (2009), 191-214.

125 Suznjevic, Mirko, Matijasevic, Maja, and Dobrijevic, Ognjen. Action specific Massive Multiplayer Online Role Playing Games traffic analysis: Case study of World of Warcraft. In *Proceedings of the 7th ACM SIGCOMM workshop on Network and system support for games* (2008), ACM, 106-107.

126 Svoboda, Philipp, Karner, Wolfgang, and Rupp, Markus. Traffic Analysis and Modeling for World of Warcraft. In *IEEE International Conference on Communications, 2007.* (2007), IEEE, 1612-1617.

127 Thawonmas, Ruck, Kurashige, Masoyoshi, and Chen, Kuan-Ta. Detection of landmarks for clustering of online-game players. *The International Journal of Virtual Reality*, vol. 6, no. 3 (2007), 11-16.

128 Varvello, Matteo, Biersack, Ernst, and Diot, Christophe. Dynamic clustering in delaunay-based P2P networked virtual environments. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (2007), ACM, 105-110.

129 Wang, Weihua, Lin, Qingping, Ng, Jim Mee, and Low, Chor Ping. SmartCU3D: a collaborative virtual environment system with behavior based interaction management. In *Proceedings of the ACM symposium on Virtual reality software and technology* (2001), ACM, 25-32.

130 WARCRAFTREALMS.COM. *Weekly Realm Activity, All Realms*. http://www.warcraftrealms.com/weeklyfactionactivity.php?serverid=-1.

131 Webb, Steven Daniel and Soh, Sieteng. Cheating in networked computer games: a review. In *Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts* (2007), ACM, 105-112.

132 Woodcock, Bruce Sterling. *MMOG Subscriptions Market Share - April 2008*, http://www.mmogchart.com/Chart7.html.

133 Yamamoto, Shinya, Murata, Yoshihiro, Yasumoto, Keiichi, and Ito, Minoru. A distributed event delivery method with load balancing for MMORPG. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games* (2005), ACM, 1-8.

134 Yan, Jeff and Randell, Brian. A systematic classification of cheating in online games. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games* (2005), ACM, 1-9.

135 Yao, Andrew C. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science* (1982), IEEE Computer Society, 160-164.

136 Yee, George, Korba, Larry, Song, Ronggong, and Chen, Ying-Chieh. Towards designing secure online games. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 02* (2006), IEEE Computer Society, 44-48.

137 Yonekura, Tatsuhiro, Kawano, Yoshihiro, and Hanawa, Dai. Peer-to-peer networked field-type virtual environment by using AtoZ. In *Proceedings of the 2004 International Conference on Cyberworlds* (2004), IEEE Computer Society, 241-248.