1-1-2012

Accurate Player Modeling And Cheat-Proof Gameplay In Peer-To-Peer Based Multiplayer Online Games

Daniel Everett Pittman
University of Denver, DEPittman@gmail.com

ACCURATE PLAYER MODELING AND CHEAT-PROOF GAMEPLAY IN PEER-TO-PEER BASED MULTIPLAYER ONLINE GAMES

A Dissertation

Presented to

the Faculty of Engineering and Computer Science

University of Denver

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

Daniel Everett Pittman Jr.

June 8, 2012

Advisor: Dr. Chris GauthierDickey

Author: Daniel Everett Pittman Jr.
Title: ACCURATE PLAYER MODELING AND CHEAT-PROOF GAMEPLAY IN PEER-TO-PEER BASED MULTIPLAYER ONLINE GAMES
Advisor: Dr. Chris GauthierDickey
Degree Date: June 2012

Abstract

We present the first detailed measurement study and models of the virtual populations in popular Massively Multiplayer Online Role-Playing Games (MMORPGs). Our results show that, amongst several MMORPGs with very different play styles, the patterns of behaviors are consistent and can be described using a common set of models.

In addition, we break down actions common to Trading Card Games (TCGs) and explain how they can be executed between players without the need for a third party referee. In each action, the player is either prevented from cheating, or if they do cheat, the opponent will be able to prove they have done so. We show these methods are secure and may be used in many various styles of TCGs. We measure moves in a real TCG to compare to our implementation of Match+Guardian (M+G), our secure *Peer-to-Peer* (P2P) protocol for implementing online TCGs. Our results, based on an evaluation of M+G's performance on the Android^{TM} platform, show that M+G can be used in a P2P fashion on mobile devices.

Finally, we introduce and outline a HYbrid P2P ARchitecture for Trading Card Games, HYPAR-TCG. The system utilizes Distributed Hash Tables (DHTs) and other P2P overlays to store cached game data and to perform game matchmaking. This helps reduce the network and computational load to the central servers. We describe how a centralized server authority can work in concert with a P2P gameplay protocol, while still allowing for reputation and authoritative account management.

Acknowledgements

I would like to thank my advisor Dr. Chris GauthierDickey for the support and guidance he's given me over the years. The work he's done to further research in networks and games is inspiring. There's no doubt in my mind that I would not have finished this thesis without his support.

I also would like to thank my committee members, Dr. Scott Leutenegger, Dr. Mario Lopez, and Dr. Nathan Sturtevant for taking the time to review my dissertation and take part in my defense. I could not have asked for a more knowledgeable set of people in my field to review my work.

To my friends Mohammed Al-Bow and Dr. Jeff Edgington, thank you for serving as a sounding board to numerous crazy ideas over the years. I feel as though I learned just as much from the projects we pursued together, that are not represented in this thesis, as I did from the projects that are.

Finally, thank you to my parents Keith and Dorinda Lech for their unwavering support through the years in every aspect of my life. Their motivation was a main driving factor for me to go to college in the first place, and I know I wouldn't be where I am today without them.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction and Overview

Massively Multiplayer Online Games (MMOGs) are a well established, profitable business. Most major MMOGs available today, such as World of Warcraft (WoW) [Ent] and Warhammer Online (WAR) [Bio], use a centralized server environment as the cornerstone of their architecture. While this model certainly is sufficient, as is shown by the large user base for these games, it suffers from a problem of *over-provisioning*. Specifically, over-provisioning is the problem that although, any given time, active user population may not be very large, there are peak times when it is. In order to support this surge in user base, game providers must allocate servers and bandwidth necessary to support peak anticipated load. An alternative to this architecture is a Peer-to-Peer (P2P) based system, in which gameplay mechanics and game state are maintained by clients participating in the game, instead of the central server.

In the field today, there exists a great deal of research in (P2P) based MMOGs. However, not a great deal of work has been done in terms of trying to understand how the player

Behavior Aspect	**Description**
Player Distributions	How players are distributed amongst different areas of the virtual world
Player Sessions	How long players play the game
Player Movements	How often players move between different areas of the game, and how long they stay in an area after they move into it

Table 1.1: Key Aspects of Player Behavior in MMOGs*: A listing and description of critical player behavior aspects to consider when designing a P2P based MMOG*

behaves *inside* the virtual world. Table 1.1 describes the key aspects of player behavior that need to be considered when creating a P2P based Massively Multiplayer Online Game (MMOG).

Accurate models, based on empirical evidence, of the player behaviors mentioned above allow for researchers to simulate new architectures using representative player behavior models. This significantly improves the ability of researchers to show that one particular architecture is better than another. We provide a set of models based on observed behavior in some of the largest Massively Multiplayer Online Role Playing Games (MMO-RPG) games active today, with the goal of helping researchers gain a better understanding of the behavior of players inside MMOGs.

In online Trading Card Games (TCGs), similar to MMOGs, online gameplay exists but typically relies on a client/server architecture. This requires that clients must be connected to the server at all times. However, unlike MMOGs, there exists little work on moving this gameplay style into a Peer-to-Peer environment.

In order to advance work in the emerging field of P2P based TCG gameplay, we propose, analyze, and evaluate a P2P based protocol, Match+Guardian (M+G), to decentralize gameplay while preventing cheating. Furthermore, in order to solve the *over-provisioning* problems that current game designers have to face, we propose a HYbrid Peer-to-Peer (P2P) Architecture for hosting TCGs, HYPAR-TCG. This system offloads much of the computationally heavy and network intensive operations onto the clients themselves. This allows for a lightweight server presence without introducing unacceptable delay to the gameplay clients.

1.1 Importance of Topic

Existing work in MMOG modeling uses very naive assumptions about the behavior of populations in virtual worlds, such as uniform distribution of players, which can lead to incorrect observations about the performance of their presented architecture. Our models provide the first realistic representation of user behavior *inside* an MMORPG.

In TCGs, there is no comprehensive P2P based protocol for handling all aspects of gameplay in mobile and possibly disconnected environments. The work that has been done to date contains severe limitations that make them unusable in those environments. Specifically, the solutions presented either involve expensive commutative encryption, such is the case with the solution to the Mental Poker [Blu83] problem, or require several disinterested peers to be available, as is proposed in the P2P TCG schemes presented in [WK05] and [Yeh08]. Our protocol, M+G, allows for many types of TCGs to be implemented in a P2P manner using lighter weight encryption schemes and no need for a disinterested third party peer. This allows for more flexibility in how the user plays the game while maintaining an acceptable level of responsiveness to user actions.

One of the largest barriers that prevents independent game developers from being successful on a large scale is the financial barrier associated with standing up a large, centralized server environment. HYPAR-TCG is a comprehensive architecture for hosting TCGs with a minimalistic centralized server presence. The benefit from this architecture is realized in the scalability of the system. Without as large of a demand for server side resources, the number of servers required to manage a large user population can be reduced. This allows for supporting a larger user base with a smaller operating budget.

1.2 State of the Field

While prior measurement research has been done on MMOs, it has primarily focused on traffic modeling and characterization. Though traffic measurement is important and useful for research in MMOs, it represents only one aspect of modeling player behavior and cannot help with identifying player actions *inside* the game. Our models compliment the traffic modeling work by adding another dimension of understanding to the actions performed by people inside a virtual world. This allows for more complete representations of players during simulations, which helps with the accuracy in evaluations of prospective architectures.

The work done to date in creating P2P based TCG protocols [WK05] [Yeh08] has focused primarily on a solution utilizing the idea of shared secrets, a concept in which individual nodes know a piece of the information being kept private but cannot discern what the hidden information is. Only when collecting all the secrets from every player and putting them together is the secret revealed. While this mechanism is effective at preventing players from knowing hidden information in a game, its main limitation is that a trusted peer must be used when creating the initial set of secrets to distribute. This adds a requirement to the protocol that nodes, other than the ones involved in the game must be available to serve as arbiters, and precludes people from being able to play in situations where such resources are not available.

Almost all existing work in P2P game architectures has been in the area of handling messaging and state management in MMORPGs [YV05, SSB09, MTKE11, JZ08, AS09, BCG08, JPNF10, LZXC06, CRJ$^+$04, BDL$^+$08, AS08, CM06, MTKE11]. Unfortunately, MMORPGs have a very different set of problems that need to be solved, such as efficient messaging to a group of hundreds of people and managing a persistent game state across an entire virtual world. A P2P TCG architecture will contain many players but all will be playing small games of usually no more than a dozen members. Furthermore, the game state is only persistent for the length of the individual match. Our P2P TCG architecture, HYPAR-TCG, allow us to take advantage of the large user base when caching player data and performing matchmaking. Our P2P TCG protocol, M+G, lets us take advantage of the small game size when designing a message passing protocol for playing.

1.3 Goals

Our goal is to help expand the understanding of player behavior inside Multiplayer Online Role-Playing Games, and to introduce mechanisms for supporting Trading Card Games inside a P2P environment.

Specifically, there are three main problems we wish to address with our work:

- Gaining a better understanding of player behavior within MMORPGs
- Creating a cheat-proof P2P based Trading Card Game (TCG) protocol
- Outlining a comprehensive, hybrid P2P architecture for TCGs using M+G

It is our hope that, in so doing, we will help advance the area of games research by providing accurate data and robust game architectures upon which the community can build.

1.4 Dissertation Outline

A description of relevant background material is provided in chapter 2. We begin by describing the state of the field in terms of MMO and multimedia modeling. In addition, we outline the challenges that face a Peer-to-Peer based game, including cheating and network connectivity issues. Finally, we describe many popular P2P network topologies in use today, comparing and contrasting each one in terms of key capabilities.

In chapters 3 and 4, we describe the observation process for, and the generated models of, the following MMORPG player behaviors: movements, zone transitions, and total play time. Our data collection methodology, along with a characterization of the data we gathered, is outlined in Chapter 3. We describe the process by which we created an add-on for two highly popular MMOs, World of Warcraft (WoW) and Warhammer Online (WAR). We also show the general trends of the data, including player rankings in term of chosen race and class. In chapter 4 we introduce the results and models derived from our observations. We show that existing models and assumptions regarding these behaviors are incorrect, and need to be revised.

In Chapter 5 we present our work on Match-Guardian (M+G), a P2P protocol for TCGs and other MOGs, that allows for decentralized gameplay while preventing cheating. We begin by outlining common forms of TCG play, and the required protocol steps necessary to support those play styles. We then describe a cheat-proof algorithm for implementing those steps. Finally, we show that, although P2P gameplay incurs an overhead on client machines in terms of extra computation cost, that cost does not introduce a perceptible delay to players.

We outline a HYbrid P2P ARchitecture for Trading Card Games, HYPAR-TCG, in Chapter 6. The goal of this architecture is to allow for decentralized gameplay and information lookup while maintaining a trusted central authority for signatures and billing. The chapter opens by providing a description of the measurable aspects of architecture performance. A breakdown of central server roles in a hybrid P2P architecture is then presented. After that, we describe how a P2P overlay can be used as both a matchmaking tool and content cache. This helps offload some of the server responsibilities to the client. A comparison of the performance of the central server, in both a traditional client/server architecture, and HYPAR-TCG follows. We show that HYPAR-TCG performs much better than a centralized server architecture in real-world usage scenarios.

Chapter 2

Background

Our work will focus on three main areas: MMOG Player Modeling, securing gameplay in P2P based TCGs, and outlining a hybrid-P2P architecture for hosting TCGs. First, in order to thoroughly understand the state of the industry concerning MMO modeling, we will detail existing work in that area plus work in the adjacent area of Multimedia Modeling (including traffic modeling, video coding, and video modeling). Second, we will be providing background on the components necessary to implement a cheat-proof TCG protocol, as well as a taxonomy of the kinds of cheats we intend to prevent. Finally, we will review Distributed Hash Tables (DHT) and other P2P overlays, summarizing the strengths and weaknesses with each. This will allow us to make an informed decision when choosing which overlays to propose for use in HYPAR-TCG.

2.1 MMO Player Behavior Modeling

Over the last several years, a significant number of measurement studies have been performed on MMOs, though most of these have focused on traffic patterns and network characteristics. Chambers et al. studied network patterns related to players and the server of small networked multiplayer games [CFSS05]. Their measurements show diurnal patterns of game populations. Kim et al. measured network patterns on Lineage II, a popular MMO in Korea [KCC$^+$05]. Their work focused on network packet sizes, Round Trip Times (RTTs), session times and inter-session arrival times. The data they recorded shows a power-law distribution for session times. This is similar to the the times we show in Chapter 4.2.3. Ye et al. devised a set of performance models for MMORPG servers and networks based on concurrent player population [YC06]. Chen et al. profiled packet inter-arrival times, packet load distribution, and bandwidth utilization of *ShenZhou Online*, a popular MMORPG [CHHL05]. Svoboda et al. modeled traffic patterns and sessions lengths for players of WoW using both wireless and wired Internet connections [SKR07].

Beyond network traffic measurement, some research has looked at traffic patterns, session lengths, and latency when measured with respect to players and user behaviors.

Tarng et al. performed a long term study of WoW in order to see if it was possible to model subscription lengths of players based on how much a user plays a MMO [TCH08]. They showed that while it is possible to predict short term behavior, long term prediction is much more difficult. Claypool and Claypool characterized latency requirements of various online games in terms of the deadline in which a user command must be processed and the precision of the commands the user is issuing [CC06]. Traffic patterns and session lengths of WoW were profiled with respect to different player action categories by Suznjevic et al. [SDM09]. They hypothesized that mobile devices could be used for some of the less traffic intensive player activities. Fernandes et al. characterized traffic patterns in Second Life during different player activities in the world [FKS$^+$07]. Kinicki et al. expanded on the work of Fernandes et al. by considering object and avatar interactions of the player in the virtual world when modeling traffic characteristics [KC08]. Finally, Szabó et al. provided a model from which you can detect the activity of a user within a MMORPG by correlating the traffic patterns observed through passive monitoring and packet level introspection [SVM09].

Chen et al. looked at how player interactions and virtual location relates to network traffic patterns and physical location [CL06]. Similar to our recent work, they found that player locations in the virtual world follow a power law distribution. They also observed that there is a correlation between continuing social interaction in a MMO and physical topology closeness with the player's IP address. The categorization of player interactions was achieved by identifying packet traces and observing common patterns. Kawale et al. created a churn model for players in MMOs that took into account the social aspect of gameplay [KPS09]. They showed that how engaged a player is with a MMO can be used to accurately model churn amongst players. Liang et al. studied movement patterns in a Networked Virtual Environment, Second Life [LTN$^+$08]. They collected metrics on how often a player moves between zones, which paths the player takes to get to new zones, and how total population changes over time. Their results for player population are very different from those observed in our work with MMORPGs. Instead of a diurnal pattern to player arrivals and departures, the authors observed a fairly constant amount of churn throughout the day. They suggested a hybrid mobility model for player movement that consists of both a random waypoint model for outdoor movement and a pathway model for indoor movement. Miller et al. studied player movement in World of Warcraft battlegrounds, a subset of the WoW game world [MC09]. They observed 13 battles using observers in the game, and concluded that players who participate in battles are likely to remain throughout the entire battle. Also, they hypothesized that a waypoint model for player movement in a battleground may be useful, but would not represent more than 50% of all movement patterns. Nae et al. provided a dynamic resource allocation model for MMOs based on user traces [NIP10]. The goal of providing this model was to move away from the centralized over provisioning of servers used to handle peak load and instead request resources from external data centers on demand. Seay et al. looked at how players in MMOs communicate with each other and group together. They also measured through survey how "committed" players felt towards their in-game guild [SJLK04].

In order to better demonstrate where our contribution lies, within the current bodies of work concerning MMOs, we have broken down the relevant publications into a table listing the research categories for each paper. Table 2.1 lists those categories.

While all of the related work provides important contributions towards modeling MMOs, especially in terms of traffic behavior, our work is the first to provide details of the MMO-RPG virtual world, its population distributions, player movements, and player interactions. Our literature search indicates that this is not a widely explored area of research.

Category	Paper(s)
Bandwidth Utilization	[CHHL05]
Churn Prediction	[KPS09]
Dynamic Resource Allocation	[NIP10]
Latency Requirements	[CC06]
MMO Resource Utilization	[NIP10]
Network Characteristics	[CFSS05]
Networked Virtual Environments	[LTN+08]
Performance Modeling	[YC06]
Player Commitment	[SJLK04]
Player Engagement	[KPS09]
Player Interactions/Movement	[CL06] [KC08] [LTN+08] [MC09] [SDM09] [SVM09]
Session Lengths	[KCC+05] [SDM09] [TCH08]
Social Interactions	[SJLK04]
Subscription Lengths	[TCH08]
Traffic Categorization	[CL06]
Traffic Modeling	[KC08]
Traffic Patterns	[SKR07] [SDM09] [FKS+07] [KCC+05] [YC06] [CFSS05] [CHHL05] [SVM09]
Virtual Interactions	[FKS+07]
Virtual Player Distributions	[CL06]

Table 2.1: Categorization of Previous Work in MMOs*: A Categorization of Previous Work in MMOs broken down by focus area of the research*

2.2 Multimedia Modeling

In order to more completely understand the bodies of work that are related to MMO modeling we will now look at a related field, multimedia modeling. Within this field we will be concentrating on three subcategories: traffic modeling, video coding and video modeling.

2.2.1 Traffic Modeling

There have been many papers written in the area of multimedia traffic modeling. Golaup, et al. described a set of traffic patterns for complex multimedia applications, and created a simulation framework to facilitate future studies [GA06]. Lazaar, et al. produced a simplified model of real-time video sources based on two different time scales, the frame and slice level, that accurately modeled the real time video sources they were using. They also assessed how much of an advantage their models could provide in an ATM network [LPP94]. Liang et al. demonstrated how fuzzy classification techniques could be used to classify MPEG video traffic into either *movie* or *sports* categories. Their algorithm performed correct classifications with a fairly low percentage of false positives [LM01]. Neame et al. proposed using the M/Pareto process, a process that represents periods of overlapping long transmission bursts, to model video traffic over packet switched networks. They showed that if constants are defined correctly, this algorithm was effective in predicting sample traffic captures. The authors were unable to provide a method for determining the constants in real time, however, which presents a limitation of the algorithm's effectiveness [NZA99]. Shah-Heydari et al. provided a method of estimating parameters for a Markov-Modulated Poisson Process (MMPP) used to model multimedia traffic on an Asynchronous Transfer

Mode (ATM) network. They showed that, using their technique, MMPP could reasonably approximate general traffic behavior [SHLN00].

Pantel et al. assessed the impact that network delay would have on real-time multiplayer games. They showed that for a racing game, a genre they classified as a worst case scenario in terms of latency requirements, a delay of up to 50ms is acceptable [PW02a]. Diot et al. provided an evaluation of the bucket synchronization mechanism of MiMaze, a multiplayer online game. They showed that the impact of network delay could be minimized by synchronizing game state at fixed intervals, and that users were satisfied with play experience even when player position was accurate only 65% of the time. The scale of the authors' experiment was small however, and they note that the scalability of their algorithm could be a problem [DG99]. Bettner et al. provided a description of the network design choices made in the development of the Age of Empires series. Most notably, the authors go over the P2P design principles that the games utilize, as well as the motivation for those designs. In order to minimize update communications, simultaneous simulations on all clients are performed. Various optimizations are also presented that further reduce the network overhead [BT01].

Pantel et al. studied the applicability of dead reckoning schemes for games. They analyzed seven different prediction mechanisms, and discussed the suitability of each mechanism in different types of games. Their conclusion was that dead reckoning helps mask some of the artifacts introduced in games by network delay [PW02b]. Aggarwal et al. attempted to reduce the negative effects of network latency in dead reckoning algorithms. In order to do this, the authors created a scheduling algorithm that delayed updates to players with low latency in order to accommodate players with high latency. In an attempt to mitigate the delays introduced by this algorithm, the authors also provided a budget algorithm that allowed for the overall update interval to be reduced. This reduction comes at the expense of not every player getting an update each round [ABMR05]. Mauve proposed a consistency mechanism for dead reckoning games that involves recomputing events in the past when new information is received. Mauve believes that this "timewarp" algorithm will prevent a scenario when an unrecoverable event (such as Player A shoots Player B) is incorrectly predicted due to dead reckoning [Mau00]. Cai et al. created an adaptive dead reckoning algorithm based on area of interest, or AOI. When a player is outside of another player's AOI, the amount of updates sent between them will be reduced. The authors show that, using this system, the number of updates needing to be sent to all players is reduced without a large sacrifice in correctness [CLC99].

This group of works' focus is very different from that of player behavior modeling. The focus of these traffic level papers are to:

- Predict traffic patters for real time resource allocation
- Categorize at a high level the genre of videos being transmitted
- Characterize the impact of delay to a multiplayer game
- Mask network jitter by efficiently sending status updates to interested players and, for short periods, predicting player movement based on prior action when no updates are available

2.2.2 Video Coding

Video coding deals with how video data can best be represented. There have been several papers written in this area. Aizawa et al. discussed methods of encoding 3-D representations of human faces in very low bit-rate applications [AH95]. Han et al. proposed an object based encoding scheme for motion video in low bit rate environments that utilizes a Markov Random Field (MRF) model based on color intensity information. They showed that their model could be a viable alternative to the traditional block based encoding standard at low bit rates [cHW98]. Torres summarized the evolution of second generation video coding schemes [TKP96]. Specifically mentioned was the MORPHECO project, which focused on segmentation based encoding. Neff et al. discussed a very low bit rate encoding scheme using the concept of matching pursuits. In this algorithm, the video's signal is passed through a multi-stage dictionary of functions. At each stage the best approximation of the signal is chosen, the energy of that approximation is subtracted from the original video. The remaining energy of the signal is passed on to the next stage as a residual [NZV94]. The authors showed that while this method of video encoding can produce better results at low bit rates, than other algorithms, there is a significant increase in computational cost.

These papers' focus is exclusively on how to best encode video for transmission across the network. Although the encoding scheme might take into account the motion of objects within the video, no attempt is made to more broadly categorize the motion into models of behavior. In addition, there is not enough information provided by the coding schemes to derive such information.

2.2.3 Video Modeling

The area of video modeling is focused on how best to index videos stored in a database, so that content can be retrieved based on some set of meaningful metadata.

Day et al. presented a model for automatically indexing videos based on both spatial and temporal events using a common set of n-ary operators. The proposed system is hierarchical in nature and allows for multi-level indexing of information [DDI$^+$95]. Hjelsvold et al. provided a generic data model for querying video that was adapted to television news domain [HM94]. Li et al. provided a model of the spatial relationship of objects in a video that supports a comprehensive set of queries [LOS96]. Petkovic et al. created a framework for video modeling that focused on automatically mapping features into an internal representation that incorporates the semantics of the video [PJ00]. Vasconcelos et al. presented a system, called BMoViES, that infers semantic content from the visual patterns in the video [VL98]. The authors are able to show that, with a 90% level of accuracy, they are able to categorize time-lines in movies according to semantic attributes such as:

- Action
- Close-up
- Crowd
- Natural set

While this group of work does not specifically address the issue of modeling behavior, it would be conceivable given a large enough body of videos involving MMOs to be able to query for interesting statistics utilizing the indexing mechanisms provided. Unfortunately, there is not nearly enough raw video material in the genre of MMO games to allow for this avenue of research.

2.3 Trading Card Game Protocols

Most modern trading card games have their roots in Magic: The Gathering$^{\text{TM}}$, which was released in 1993. The game consists of a complete library of cards where players build their own collection by purchasing packs of cards. Each individual card has some level of rarity such that the more rare a card is, the fewer copies are produced. Each card pack has some guarantee of containing a set ratio of very rare, rare, and common cards. For example, a card pack may be guaranteed to contain at least one very rare card, three rare cards, with the remainder being common cards. As one may guess, the rarest cards tend to be the most valuable and are often the most *powerful* in terms of gameplay, thereby creating an economy around collecting the rarest cards. Other well-known examples of trading card games include Pokémon$^{\text{TM}}$, the World of Warcraft Trading Card Game$^{\text{TM}}$, and the Yu-gi-oh! Trading Card Game$^{\text{TM}}$.

2.3.1 Mental Poker and Distributed Random Number Generation

The idea of playing cards in a distributed fashion without cheating, termed *mental poker*, was first discussed by Shamir et al. [SRA81]. The authors show that its impossible to deal from a *shared deck* of cards without one player knowing what card another player received. They then described a protocol that relies on commutative encryption to solve this problem. Note that their impossibility result still holds: if one could break the encryption in a reasonable amount of time, dealing from a shared deck of cards without revealing who received which card to the other player is still impossible. The primary difference between this problem and our problem is that in our case, the deck of cards between players *are not shared*. Without a shared deck, our solution can avoid using commutable encryption. As with their solution, we rely on the encryption not being easily breakable, where *easy* means within the span of time it takes to complete a game.

The protocol described in Chapter 5 relies on being able to generate a random number between two or more players fairly. In this case, we define fairly as either player not being able to influence the outcome of the generated random number. This is solvable by the well-known coin flipping by telephone protocol [Blu83]. With this method, Alice picks a random number r_A, cryptographically hashes it and sends the hash, $H(r_A)$ to Bob. Bob picks a random number, r_B, signs it and sends it to Alice. Alice then XORs r_A and r_B to determine the final value. Alice can then reveal the result later by giving Bob her private number allowing Bob to compute the same value. Note that since Alice has no idea what number Bob will pick, she cannot influence the final random value. Furthermore, Bob cannot influence the final value since he has no idea what initial value Alice chose. Expanding this to n players requires that each player generate a private number and one public number to share with the other $n-1$ players. Each player then XORs their private number with the $n-1$ other public numbers.

2.3.2 P2P Turn-Based Gameplay

An alternative to solving the shared resource problem in a P2P environment was presented by Wierzbicki et al. in [WK04] and improved on in [WK05]. Specifically, the authors were trying to solve playing Scrabble$^{\text{TM}}$, a word reconstruction game using a shared set of letters, in a P2P environment. Their solution involved the use of several disinterested players to act as arbiters for actions such as shuffling the letters and choosing a shuffled letter from the set of available resources. Message secrecy was handled through constructing shared secrets that would be distributed between the arbiters. While the authors do show that their protocol is effective at preventing cheating, it requires several players to be in a P2P overlay. Also, if the node that creates the shared secret to distribute is colluding with another player, it is possible to know all the shared secrets for a given letter.

While the author's protocol could be adapted for our needs, it is, like the mental poker solution, far more complicated than is needed to handle a non-shared resource game such as TCGs. It is interesting to note, however, that if the rules of Scrabble$^{\text{TM}}$ were slightly modified to allow for generating letters based on their probability of appearance, rather than drawing from a fixed set of pre-generated resources, the distributed random number generator we describe in Chapter 5.2.3 could be used. This would reduce the number of required players to only be those involved in the game and would prevent collusion between a subset of the peers.

Another P2P protocol for card games is described by Yeh in [Yeh08]. Similar to [WK05], the author proposes a system based around shared secrets for hiding card selection from other players. Their improvement is based around the improving the run-time complexity of shared secret maintenance. Unfortunately, this protocol still has the same limitations as stated earlier. Namely, collusion with the shared secret constructor is still possible, and at least one player not involved in the game must be present.

In the ad-hoc P2P networks that our protocol is designed for, we cannot count on relying on several disinterested players. Also, we would like to eliminate the trusted third-party peer from the design so that collusion with that player is not possible. Thus, neither of these schemes for message security are viable solutions for our problem.

2.3.3 Cheating in Games

We define cheating as: any action by a player that circumvents the normal course of actions in an game that gives the player an unfair advantage in the game or over another player. Cheats are primarily possible due to security flaws in an application, protocol, or network. We can create a taxonomy of cheats based on the *layer* in which they occur [GZLM04], which is useful when we are considering what we can and cannot prevent at a particular layer. We note that Yan and Randell [YR05], as well as Kabus and Buchmann [KB07], have also created classifications of cheating in games. Their techniques differ from the one we will be using. Table 2.2 lists several common types of cheats and the layers on which they occur.

Cheats occurring in the first category, the network level, allow players to gain an advantage in the game by exploiting security flaws in network and routing protocols. Cheats occurring at the application level originate from applications modified to act differently then their original intent. Typically network and application level cheats both occur through modifying the application, though network cheats specifically target security flaws with the network protocols. Last, cheats occurring at the game level are cheats that occur in the game by breaking game rules (possibly by exploiting bugs or sidestepping rules in some

Cheat	Level	P2P	Client/Server
Denial of Service	Network	✓	⋆
Fixed Delay	Protocol	✓	⋆
Timestamp	Protocol	✓	⋆
Suppressed Update	Protocol	✓	
Inconsistency	Protocol	✓	
Collusion	Protocol & App	✓	✓
Secret revealing	App	✓	⋆
Bots/reflex enhancers	App	✓	✓
Breaking game rules	Game	✓	⋆

Table 2.2: A Taxonomy of Cheating*: A List of the common forms of cheats performed. Check marks indicate whether this type of cheat is possible under the listed architecture. Stars indicate cheats which are partially possible*

way). These may also occur by modifying the application, such as adding new beneficial cards to a deck during play.

Much of the research in cheat prevention in games has looked at preventing protocol or application level cheats, i.e., those cheats which occur by modifying network protocol behavior or altering the application in order to gain an unfair advantage. For example, Baughman and Levine developed *asynchronous synchronization*, one of the first protocols that prevented some network level cheats [BLL07]. Cronin et al. added pipelining to the simple lockstep protocol and secured it from several types of cheats introduced by adding a pipeline to actions [CFJ03]. GauthierDickey et al. developed the NEO protocol as a replacement for a simple lockstep protocol that further prevented network level cheats [GZLM04].

At the application layer, Li et al. described information dissemination strategies that limit state information exposure to clients [LDM04]. Chambers et al. showed how to prevent *maphacks*, a type of information exposure cheat [CFFS05] in real-time strategy games. Schluessler et al. showed how to detect when players were using bots, which falls under application-level cheats [SGJ07].

In this paper, we focus specifically on avoiding or detecting *game level* cheats in Online TCGs, which are those cheats that occur by breaking the rules of the game, specifically centered around preventing players from inserting/removing cards, seeing the opponent's cards, or not fairly shuffling a deck of cards. The original protocol for detecting and preventing cheating was described in [PG11]. An expanded description of the protocol is also described in [PG12]. We detail this further in Section 5.2.

2.3.4 Network Address Translation and P2P Protocols

Peer-to-peer protocol users invariably have to deal with the problem of Network Address Translation (NAT). When a client is connecting to a remote endpoint, depending on the network configuration between the client and the endpoint, the IP address and port number used by the client can be changed, or *translated*. The remote endpoint would then respond to the translated address, which in turn would be translated back to the originating IP and port.

While this system is fine for user initiated requests, it restricts the ability for remote peers to initiate unsolicited connections. In a P2P environment, this means that a peer that is behind a network using NAT would be unavailable for connection. This severely limits how the peer can interact and participate in the P2P overlay.

There are multiple ways to overcome this problem, such as consistent endpoint translation in the NAT and *TCP hole-punching*. In consistent endpoint translation, a predicable NAT device can be probed to see what translation scheme is being used. This would allow a client to "guess" what translated address it will be allocated and have clients connect to it using the predicted address. In TCP hole-punching, a proxy endpoint outside of the NAT can serve as a connection arbiter for peers. If peers A and B were tying to connect to each other, but they were behind a NAT, they could first contact an arbiter C, who would then know the translated addresses of both A and B. C could then send that information to A and B so that a direct connection between the peers could be established.

Using these approaches, 80-90% of the peers can successfully connect to each other [BSK05]. However, this does not account for the case where two users are behind the same NAT trying to connect to other users. In this case, a third party would need to be used for relaying port numbers prior to TCP hole-punching.

In M+G, peers are either playing on a disconnected, ad-hoc P2P connection or in a connected, structured P2P connection. In the first scenario players are directly connected to each other and do not have to worry about NAT. The second scenario is addressed by our architecture presented in Chapter 6.

2.4 Scalable Peer-to-Peer Overlay Networks

In order to choose overlays suitable for our hybrid P2P TCG architecture, it is important to understand the performance characteristics of each type. Included in this section is a summary of the most popular types of Distributed Hash Tables (DHT) and other forms of overlay networks available today. Table 2.3 describes the key capabilities of each overlay that is described in this chapter.

Name	**Key Space**	**Query Hops**	**Range**	**Multi-Attribute**	**Locality**
CAN	DHT	$O(logN)$	NO	NO	YES
Chord	DHT	$O(logN)$	NO	NO	NO
Kademlia	DHT	$O(log_{2^b}N)$	NO	NO	YES
Pastry	DHT	$O(log_B N)$	NO	NO	YES
SkipNet	Skip Lists	$O(logN)$	YES	NO	YES
Mercury	Load Balancing	$O(1/k * log^2 N)$	YES	YES	YES

Table 2.3: Summary of P2P Overlay Capabilities*: This table details key information about common P2P overlays. Information is provided in terms of: key space maintenance, number of hops worst case when performing a query, whether the overlay supports range queries, if the overlay supports multi-attribute queries, and if the overlay can be made content or locality aware*

2.4.1 Overview of Peer-to-Peer Networks

A Peer-to-Peer (P2P) network is a collection of client machines that act as both a client and a server to other clients in the network. Each machine, or *node*, in the P2P network have equal responsibility. One of the first uses of a P2P network can be found with Napster ™ [nap], a file-sharing network that focused primarily on sharing music. Another application can be found with GridCast [CLZ+08], a P2P based video-on-demand system. Although Napster and GridCast have different goals in terms of functionality, the core goal of the network is the same: Group nodes into an *overlay* so that useful operations can be performed.

2.4.1.1 Peer-to-Peer Overlays

In a naive implementation, a P2P network could be constructed so that every node is connected to every other participating node. This design has an advantage in that no node is further than one message, or *hop* away from any other node. The disadvantage to this design, however, is found in the number of connections that must be maintained. For each node in the system, connection information about every other node must be kept. This means that each node maintains N connections, and there are N^2 connections total in the system. There are two main problems with this design: First, the number of connections required for each node cannot scale. It is impossible for a client workstation to participate in thousands of active network connections. Operating system limitations would prevent such a system from being possible. Second, whenever a node leaves or joins, up to N connection updates must be performed. Given that, as is shown by Gummadi et. al [GDS$^+$03], the average time a node is connected is ≈ 2.5 minutes, the number of connection updates that would be performed in a hour is far too great for clients to handle.

To solve this problem, most P2P systems organize nodes into an *overlay*, or logical topology of nodes. In a P2P overlay, the interconnections between nodes are maintained in such a way that different properties of the connectivity graph between them can be used to optimize different problem sets. As stated above, the optimal number of hops a P2P overlay could hope for is one. Since that is not possible, most P2P overlays try to balance the number of connections per node with hop length such that no more than $O(logN)$ number of hops is required to perform an operation.

2.4.1.2 Distributed Hash Tables

One of the more common classes of P2P overlays are Distributed Hash Tables (DHTs), a system used to store and lookup data in a decentralized manner. Similar to the Hash Table data structure, a consistent hashing function is utilized on the ID of the content being stored, which then provides an indicator as to where to look for the data in the network. All nodes participating in the DHT are responsible for a subset of the hash function key space, and have an associated node ID. The node ID is used to determine the specific subset that the node will take responsibility for. Typically, keys are either 128 or 160 bytes long.

When searching for data, or inserting data, almost all DHTs use a distance function for keys to determine how far away a node is from the data it is looking for. It is also used to determine which neighbor will route the query closer to the goal.

2.4.2 Content Addressable Network

A Content Addressable Network (CAN) is an implementation of a Distributed Hash Table. The basic premise behind this system is that incoming peers will bootstrap to some known active node, acquire a segment of the hash table key space from an existing node based on the incoming node's ID, and then "own" that space for the purposes of storing keys that hash to that location, as well as, responding to queries that terminate there. The authors, in addition to presenting the basic idea of CAN, provide many modifications that seek to improve various aspects of the CAN's performance depending on what it is the end user needs to optimize. Examples of what the authors include are improvements that reduce virtual path length through the CAN, physical path length in the actual network that is being overlaid, and replication methods that help with the fault-tolerant nature of the system.

Each node holds a routing table that contains the next hop in routing between CAN nodes. Entries in the routing table for a node are maintained based on the *neighbor* list of that node. A node is considered to be a neighbor if the key space it is responsible either overlaps or is adjacent to the key space of the CAN node. When a query is routed through a node and that node is not responsible for the key space requested, the query is hashed into the routing table and the IP address stored in that position of the table is the node to which the query is forwarded.

CAN can route based on multiple dimensions of key space. This allows for queries based on different indexed properties of the data being stored. Thus, the average routing length of CAN is $\frac{d}{4} \times n^{\frac{1}{d}}$ where d is the number of dimensions and n is the number of nodes. Average routing length grows as $O(n^{\frac{1}{d}})$.

2.4.3 Chord

Chord [SMK+01][LCP+05], another DHT, works by using a distributed consistent hashing function to guarantee an equal distribution of keys throughout the collection of nodes. This allows for only a small amount of routing entries be maintained at any node (logarithmic on the number of nodes, specifically).

The DHT key space in Chord is ordered and organized into a ring. Each participating node occupies a location in the ring, and holds a finger table that serves as a routing table for queries. Each entry in the finger table is selected so that, when routing a query, large sections of the key space ring can be short circuited to optimize the search. The simple concept of routing is as follows: In order to find the correct node to pass a query to, a node in the Chord ring forwards the request to the successor node (via the finger table) until found. A successor node is defined as a node whose ID is larger than the requesting node, but is smaller than the ID of the item being searched for. Each node is responsible for data whose ID is greater than or equal to the node's ID, but are smaller than the ID of the next node in the ring.

The average cost of a query in Chord is $O(logN)$, which is the path length required to route the query to the correct node. Joining and leaving has a cost of $O(logN)^2$ in order to redistribute keys.

2.4.4 Kademlia

Kademlia [MM02][LCP$^+$05] is a DHT that incorporates XOR operations on node IDs into its routing algorithm in order to determine distance between peers. Nodes are grouped according to the concept of a k-bucket, meaning that any query returns the k closest peers so that the originator can determine a node from which the data will be requested (based on latency, for example).

The authors show that it is able to withstand some aspects of a Denial of Service (DoS) attack since the k-bucket system it utilizes for caching nodes is only updated (when full) after an existing node leaves. Due to this, new nodes cannot flush out existing entries.

Routing is performed in a 160-bit key space with items stored at peers with IDs close to that of the object, for some definition of close. Expected number of hops grows by $log_{2^b}n$ with a routing table size of $2^b log_{2^b}n$ k-buckets per node.

2.4.5 Pastry

Pastry [RD01][LCP$^+$05] is a shortest prefix matching algorithm DHT in which each node is assigned a 128-bit identifier in a circular key space.

Queries are routed from one node to the next by following the rule that the node you forward to should have an ID that shares a common prefix that is at least b bits longer than the current node with the current query. If such a node is not available, the message is forwarded to a node that shares the same prefix match, but has a node ID numerically closer to the ID of the target key.

The average number of steps to route a request grows as $O(log_B N)$ where $B = 2^b$ with b being a tunable parameter. As b increases, the number of steps to route a requests decreases, but the number forwarding table entries required at each node increases by approximately $O(log_{2^B} N) * (2^b - 1)$.

2.4.6 SkipNet

SkipNet [HJS+03] is a P2P overlay based on the skip lists data structure. The basic concept of a skip list is that, if content is inserted into a list ordered by node ID, short-circuit pointers from each node can be maintained that short circuit, or "skip" a range of nodes to provide faster search performance. SkipNet's design can provide content locality as well as path locality. In this context, content locality refers to the ability to specify which node, or groupings of node, content should be stored at. Path locality further enhances the capabilities provided by content locality. This is accomplished by guaranteeing that messages will not be routed outside of a specified grouping of nodes. By combining both content and path locality, SkipNet can guarantee that operations on and storage of messages will remain within a specified grouping of nodes.

In games, there are clear advantages to both content and path locality. As an example, take a game in which there are two factions, A and B. In a P2P overlay such as Chord [SMK+01], a private message between members of faction A could be intercepted by faction B because of the uniform nature of nodes within the overlay coupled with the routing protocol used within it. SkipNet could prevent this scenario by allowing for a path local overlay that would never route the message outside of A's domain.

As another example, take for instance different areas, or zones, within a virtual world. Updates within the zone have no need to be propagated outside of it. In SkipNet, assume each player, upon entering the zone, changed their name prefix to be the current zone's name so that they were placed into the same content-local grouping. It would be a simple matter then for a user to identify which players to contact in order to send a mass update in the zone; just send a wildcard range query for all members with the correct zone prefix.

Experimental and theoretical results, included in the author's work, show that resilience in the face of network failures is far better than that of both Chord and Pastry. This is because of the data locality inherent in the SkipNet architecture.

2.4.7 Mercury

Mercury [BAS04] is a scalable query routing and data storage protocol that supports multiple attribute range style queries. Their implementation is similar in design to a DHT, but does not use a uniform hashing algorithm. Instead, Mercury explicitly load balances content amongst nodes in the overlay. This is required in order to support the multi-attribute query properties of the system. The authors provide many optimizations to their algorithm, including but not limited to:

- query caching, which helps reduce the number of hops for common queries
- random sampling techniques to approximate system load and balance queries accordingly

- explicit load balancing algorithms to counteract areas of popularity in key-space
- query selectivity to help reduce query message complexity

The authors also propose an application of Mercury, a publish/subscribe system, for distributed gaming. The authors show how the built in functionality for Mercury makes this a straightforward extension to implement. The simulation results for the publish/subscribe system show that Mercury provides a underlying framework that performs much better than simple broadcast.

In the context of games, this publish/subscribe systems can have many uses. For example, assume once again we are dealing with a player in a virtual world. That player can subscribe to a global event topic, a topic for the current zone he is in, and even perhaps subscribe to a topic associated with the guild to which he currently belongs. Any state change that occurs in those topics would be populated to the character through Mercury, with the originator of the event only having to send one message.

Chapter 3

MMO Modeling Methodology and Data Characterization

In order to accurately model behavior in a virtual world, it is necessary to study several aspects of a player's behavior. Although there are many individual data points to collect, the individual models can be categorized into three groups:

- Player Movement Models
  - Player movement, both within a zone (intra-zone) and between zones (inter-zone)
  - Player distributions
  - Number of zones visited during a session
- Player Interaction Models
  - Player to player interactions

    - Interactions of players with the virtual world

- Temporal Models for Player Behavior
    - Session lengths
    - Arrival and departure rates
    - Time spent in a zone

3.1 Data Collection Methodology

Two primary methods can be used for measuring virtual populations and behaviors of players in MMORPGs: analyzing logs and data generated from the game server or using probing-based measurements which try to infer properties of the system. In the first case, log and data analysis has the advantage of being accurate and providing information about player behavior that cannot be gathered otherwise. Unfortunately, though, few game companies are willing to share their logs, and even if they do it is not always the case that they log the needed information. In the second method, traffic or in-game measurements are taken to generate the needed data, but this may not be possible if the game client does not support the needed API. Our dataset was generated through the second method, which unfortunately prevented observations in the *Player Interaction Models* category.

3.1.1 Probing-Based Measurements

In order to measure population information in World of Warcraft (WoW) and Warhammer Online (WAR), we designed a set of scripts that run from the game clients using the Lua[1] scripting interface provided by both games[2]. For WoW, we modified the Census+ add-on to collect broad information about all players currently online[3]. WAR's add-on was custom written but was based on functionality of Census+. We also wrote an additional add-on for both MMORPGs to record continuous detailed information about a randomly selected subset of players.

3.1.2 Virtual Population Census

To measure the virtual population, we performed server queries from the clients using the *who* service, which allows a player to search for another player in the game. Using the who service, we performed snapshots of the server population every 15 minutes. This allowed us to record the entire population seen during a measurement.

A 'who' query actually only returned a small subset of players, but allowed us to use expressions to narrow our search. These expressions included level restrictions, race restrictions, specialization restrictions, and name restrictions. For example, we could query for all players that were level 40. If the maximum number of players was returned from the query, we would further restrict the query until a subset fewer than the maximum was returned. Thus, we could systematically query the server for all players currently in the game, though each complete census depended on the current population of the server.

[1]`http://www.lua.org`

[2]Source code available from `http://www.cs.du.edu/~chrisg/measurements`.

[3]`http://www.warcraftrealms.com/censusplus.php`

A full census of the virtual population could take up to 10 minutes when the server load was high. During this time, players could leave or join the game and we might possibly miss them from the census. To understand the amount of population fluctuation we implemented a back-to-back snapshot where a second census was taken immediately following the first one. By examining the two censuses, one could measure the overall churn in the game.

3.1.3 Detailed Movement Measurements

Because we did not want to schedule constant censuses and because snapshots of the population took so long, we did not have the granularity we wanted for modeling player movement (i.e., knowing where a player is every 15 minutes leaves a lot of guess-work for what they did during that long interval). We addressed this issue by using the *friends list*, which allows you to have a continual stream of information about where your 'friends' are currently playing the game. We populated our friends list with a random subset of players from the census snapshots. However, because snapshots were only taken every 15 minutes, we only added friends from the 2nd snapshot who were not in the 1st snapshot since we knew they had recently logged into the game. In WoW, we were able to maintain a maximum list of 50 players while in WAR we could have a maximum of 40 players.

The friends list allowed us to track a small subset of players including when they log on and off and where they are during each query. As 'friends' logged off, they were replaced with new players from the 2nd half of the back-to-back census snapshots, so that we continually observed a subset of players.

The final set of data underwent a minor cleaning process. Because of the way the Lua add-ons worked, we had to log out of the game to record the data, which in turn forced our measurements to stop once a day to log the information. During the period that we logged in and out of the game, we might end up with an incomplete snapshot or only one-half of a back-to-back pair. In our friends list data collection, we might see a player log on, but not log off because we had logged out of the game to record the data. We simply eliminated these suspect snapshots and movement collections from the data set.

Using our methodology, we observed over $125,000$ individual players and tracked player movements on over $75,000$ sessions. The high ratio of random players seen by tracking them via the friends list indicates the success of our methodology in obtaining a reasonable sample of players seen from our census. We note that while all MMOs do not use Lua as a scripting interface for the game client, the who service and friends list tend to be universal and therefore similar techniques could be applied to other MMOs.

WoW was measured over a 4 month period on the Aerie Peak server while WAR was measured over 2 weeks on the Volkmar server. We examined data from other servers and it was similar to the results presented here, thus these two servers are sufficiently general for both games. The results of the analysis and modeling of our dataset were published both in Netgames 2007 [PG07] and in MMM 2010 [PG10], though as noted previously, not all facets of the dataset have been fully explored.

3.2 Data Characterization

Our data is stored in XML using the following schema in Appendix A. Generally, data points are broken into two parts: *snapshots* and *playerObservations*. Each *Snapshots* entry contains a sequence of snapshots, each recording the player name, level, guild, location, specialization, race and faction. The individual snapshot also contains the start time and stop (i.e., completion) time of the snapshot.

PlayerObservations contain two timestamps indicating the starting and stopping of a sequence of observations. Each individual observation consists of a sequence of times and locations of the given player.

The XML data is currently available in a compressed form at: `http://www.cs.du.edu/~chrisg/mmoobservations`.

3.2.1 Dataset Statistics

Faction	**Total Unique Players**
Alliance (WoW)	71,510
Horde (WoW)	38,653
WAR	12,198

Table 3.1: Totals by Faction in Dataset*: A breakdown of the total population by faction*

The dataset contains a total of 122,361 unique players and 3,205 total guilds with the breakout by faction and game listed in Table 3.1. General statistics broken down by race as shown in Figure 3.1. On the Alliance faction, two races were played more prominently than other races, while on the Horde faction, the distribution was more even. A breakdown of user population by specialization is shown in Figure 3.2. Note that the last few boxes in the WAR populations are because WAR has more specializations that WoW. Also note that specializations were more evenly distributed in WAR than in WoW.

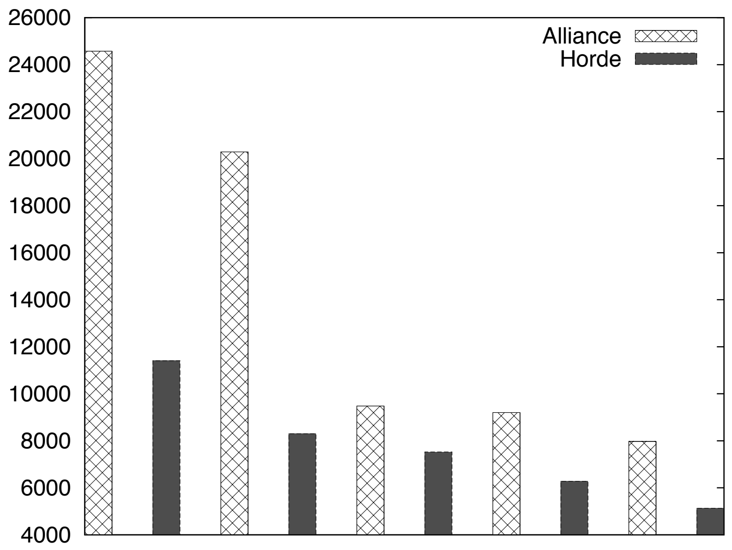

Figure 3.1: Total Unique Players by Race in WoW*: Horde and Alliance populations are sorted from most populated to least populated races*

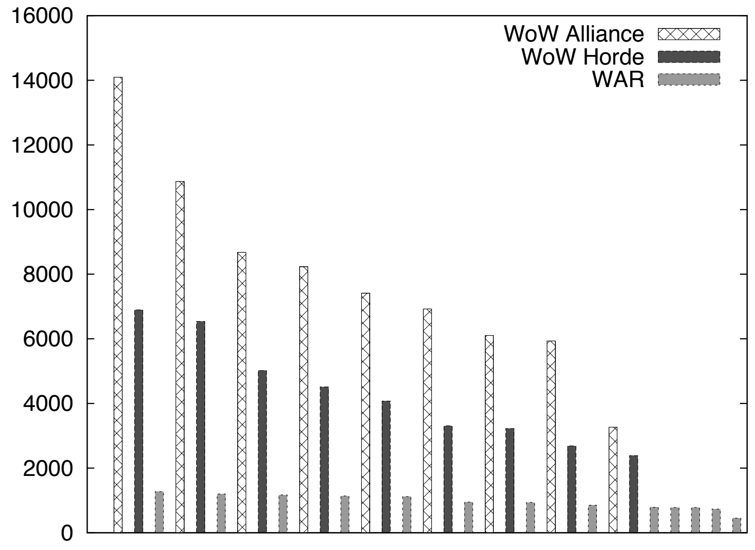

Figure 3.2: Total Unique Players by Specialization in WoW and WAR*: Total populations are sorted from most populated to least populated specialization*

The dataset also contains information about levels and guilds, but because the data collection was taken over several months, a player is seen with multiple levels, depending on how many they played through during the observations. The same issue holds for guilds and as such, a level breakdown or guild breakdown is not easily summarized without modeling.

Chapter 4

MMO Behavior Models

As stated earlier, player behavior in MMORPGs can be broken down into three categories: *Movement*, *Interaction*, and *Temporal*. Our work focuses on providing models for both the *Movement* and *Temporal* categories.

4.1 Player Movement Models

Inside the virtual world in MMORPGs, regions are statically divided into *zones*. From both WoW and WAR we measured the zones each player visited (including the order visited). The goal in modeling this behavior is to see how players move between zones. Some questions that we want to answer include:

- How many zones do players visit during a session?
- Do players move back and forth between the same zone?
- Are some zones visited more often than others?

4.1.1 Player Distributions

We started by measuring the distribution of players in the virtual world of both games. Specifically, we measured how many players are in each zone over the measurement period. After examining the data, we realized that a large percentage of the zones had 0 players in them. To model this correctly, we calculated the quantity of 0 population zones we examined (36.44% in WoW and 78.56% in WAR) and removed these from the data set for the purpose of modeling the remaining data. We then plotted the remaining points to cover the probability from 0 to 1. Using the least-squared method, we fit the data using a Weibull distribution. Figures 4.1 and 4.2 show the measured data and the fitted Weibull distributions of the zone populations. Note that in both games, only a few zones have more than 50 players, while the majority of zones have fewer than 10 players. For WoW, we saw an average of 121 players in a zone, with a minimum of 0 and a maximum of 293 players. On WAR, we saw an average of 74 players in a zone with a maximum of 156 players and a minimum of 0.

Player distributions were modeled very closely using a Weibull distribution with a standard deviation of 0.007 for WoW and 0.008 for WAR of the residuals from the measured data and models. Thus, given a uniformly distributed random number $0 \leq p \leq 1$, we can model WoW and WAR population distributions as follows:

$$Population_{WoW}[p] = \begin{cases} 0 & \text{if } p \leq .3644 \\ \lfloor 1 - e^{-(((p-.3644)/.6356)/12.744)^{0.7822}} \rfloor & \text{otherwise} \end{cases}$$

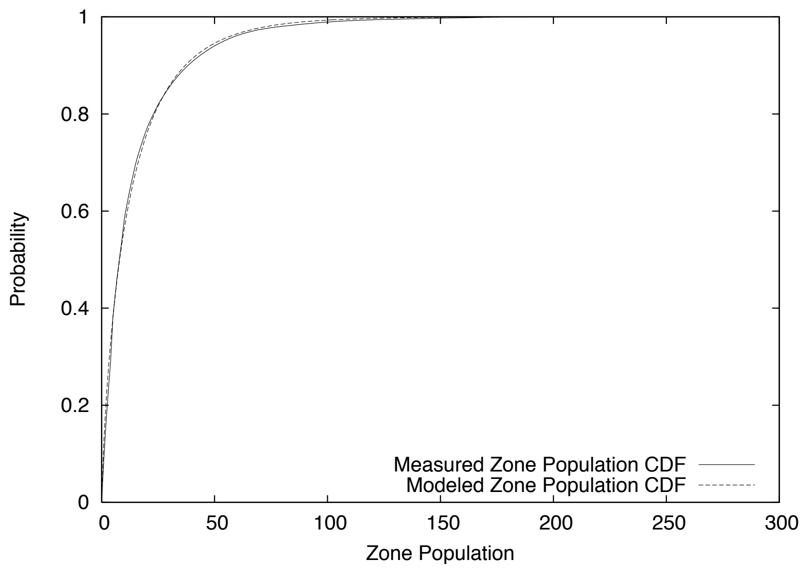

Figure 4.1: Distribution of Players per Zone in WoW*: This figure shows the distribution of players per zone in WoW. There were 36 zones without players that are not included in the CDF.*

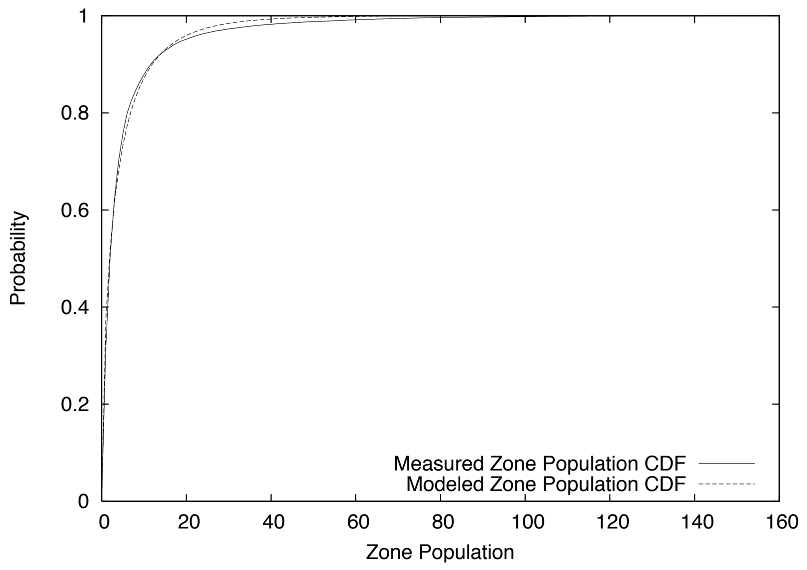

Figure 4.2: Distribution of Players per Zone in WAR*:This figure shows the distribution of players per zone in WAR. There were 78 zones without players that are not included in the CDF.*

$$Population_{WAR}[p] = \begin{cases} 0 & \text{if } p \leq .7856 \\ \lfloor 1 - e^{-(((p-.7856)/.2144)/3.256)^{0.6417}} \rfloor & \text{otherwise} \end{cases}$$

One of the primary uses of the Weibull distribution is failure prediction for devices. As time increases, the probability of a failure increases according to the parameters of the Weibull function. Conversely, the probability of functioning correctly decreases over time. It is this property of decreased likelihood over time that makes Weibull a good fit for this data. A very high percentage of zones do not have a large user population. Thus, as population grows, the probability that given zone will contain that population decreases quickly.

The measurements of player distributions are important because they show that players are *not* uniformly distributed in the virtual world. With the exception of Baughman and Levine, who used real traces from a small networked multiplayer game called XPilot [BL01, XPi], much of the prior research in scalable game architectures has assumed this. Clearly, given a uniform distribution of players, almost any architecture can be reasonably well-balanced so that it scales well. However, a Weibull distribution indicates that players tend to group in large numbers in only a few zones, causing stress on any architecture as it has to handle the increased number of interactions between players. *Therefore, game designers and researchers must consider this Weibull distribution of players in which a few zones contain a large number of players while many zones only have a few (or no) players when characterizing the potential load on a MMO architecture.*

4.1.2 Number of Zones Visited

We hypothesized that a linear relationship exists between the number of zones visited during a session and the session length. To test this hypothesis, we measured how many zones the players traveled to each session in both WoW and WAR and plotted the results in Figures 4.3 and 4.4. The number of zones visited are not unique zones, but the total number of times a player moved from one zone to another.

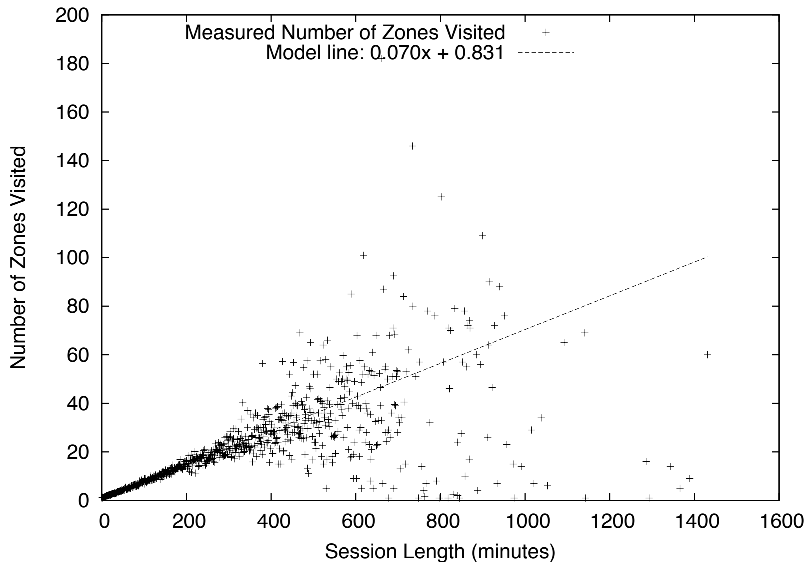

Figure 4.3: Zones Visited vs. Session Lengths in WoW*: This figure shows the number of zones visited plotted against the session time in minutes in WoW. A simple linear equations is used to model the data.*

From these figures, we see that our hypothesis held for the 80% of session times in both games, i.e., those 200 minutes and below in WoW and those 100 minutes and below in WAR. On both graphs, the hypothesis no longer seemed to hold for the highest 20% of the sessions. One explanation may be that players who are on for long periods of time behave differently in the game than those on for shorter periods. Note that the game will disconnect

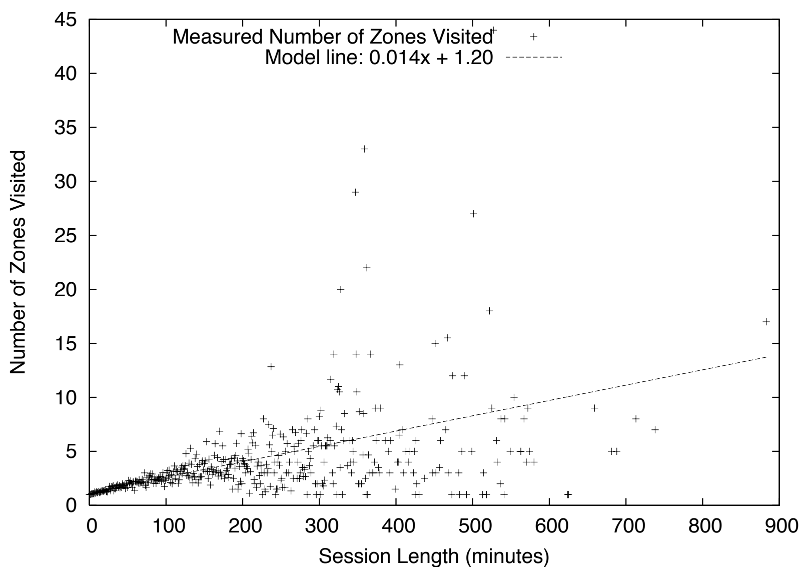

Figure 4.4: Zones Visited vs. Session Lengths in WAR*: This figure shows the number of zones visited plotted against the session time in minutes in WAR. A simple linear equations is used to model the data.*

players who remain idle for longer than 10 minutes. Thus, even these long sessions consist of active players or bots.

In both cases, we model this behavior using a simple linear equation. For WoW, we found that the equation $y = 0.070x + 0.831$ works well while for WAR we found that the line at $y = 0.014x + 1.20$ works well. For future work, we plan on exploring how the longer sessions can be modeled more accurately.

4.1.3 Player Movement

The final aspect we measured with regards to player movement was *how* players moved between zones. Our hypothesis was that the random waypoint model of player movement is not accurate for MMORPGs. Our results indicate that, instead, a log-normal distribution of waypoint choices is more accurate.

To model this type of player movement, we examined the zone locations of all of the players we tracked and recorded where they were from one measurement to the next. We created a Markov chain of zone to zone player movement from this data. A Markov chain is represented by a two dimensional matrix where each row represents the probability of a transition from the row header (or zone in this case) to a given column header (a destination zone). To visualize the matrix, we created a square image where each pixel represents a cell in the matrix and is colored gray according to its probability in the table, with black being a probability of 1 and white a probability of 0. Figure 4.5 and 4.6 shows the result of this visualization.

In order to create a player movement model, we also wanted to be able to randomly generate Markov chains that had the same properties as the measured Markov chains. We found that a log-normal CDF worked well in modeling these probabilities. The results are seen in Figure 4.7 and 4.8.

The modeled CDFs are defined as follows, where given a uniformly distributed number $0 \leq p \leq 1$ and where *erf* is the Gauss error function:

$$Choice_{WoW}[p] = \frac{1}{2}(1 + \mathrm{erf}(\frac{15.45 + \log(p)}{6.399\sqrt{2}}))$$

$$Choice_{WAR}[p] = \frac{1}{2}(1 + \mathrm{erf}(\frac{12.23 + \log(p)}{5.235\sqrt{2}}))$$

Note that in using this function to create the Markov table would require each line to be adjusted so that it summed to 1.

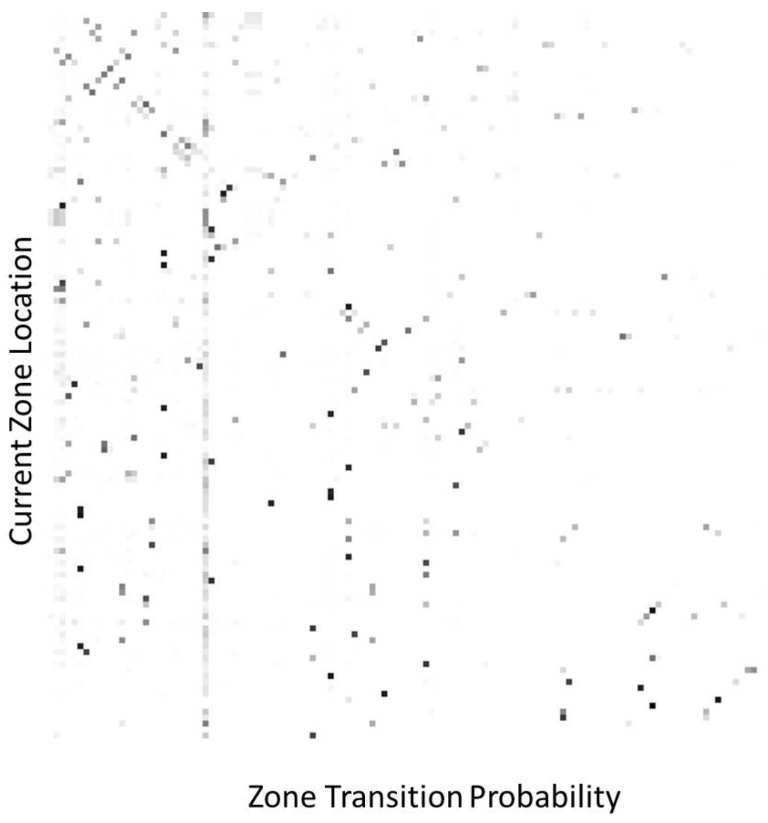

Figure 4.5: Visualization of Markov Chain Probabilities in WoW*: This figure is a visualizations of the Markov chain generated from the measured player movements between zones in WoW. Higher transition probabilities are shaded darker*

The log-normal distribution is a good fit for data that is clustered at the beginning of the range of possible values, as the function is biased to the right. In the case of zone choice probability, a vast majority of the choices will always be zero, which makes log-normal model the data well.

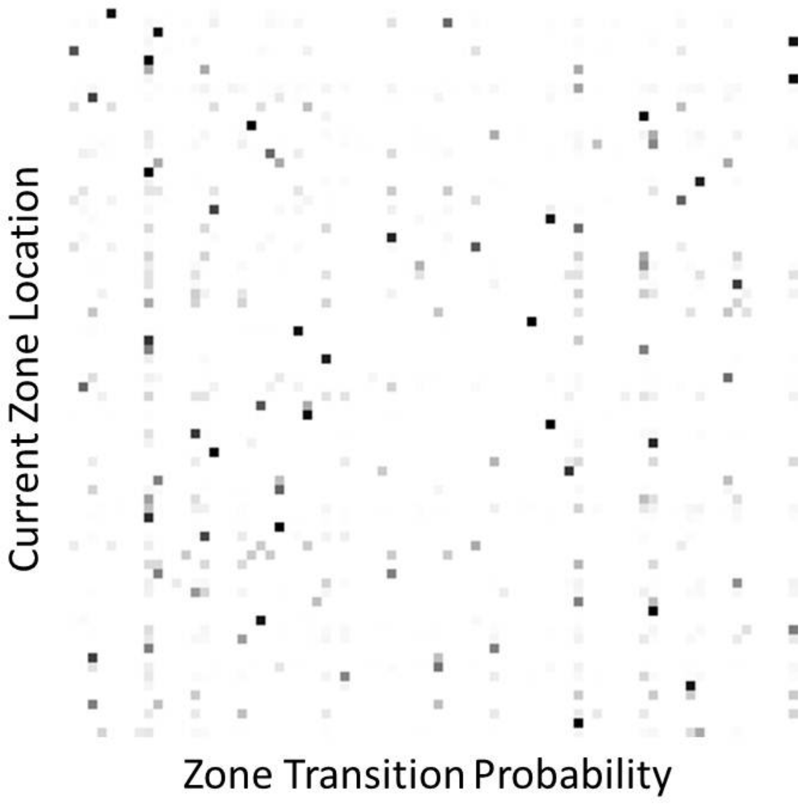

Figure 4.6: Zones Visited vs. Session Lengths in WAR*: This figure is a visualizations of the Markov chain generated from the measured player movements between zones in WAR. Higher transition probabilities are shaded darker*

4.2 Temporal Models for Player Behavior

For both WoW and WAR we measured the total populations over time, the lengths of each session observed, and the time spent in each zone. By looking at this data, we seek to answer the following questions:

- How long does a player stay in a particular zone?
- How long does a player stay in the game?

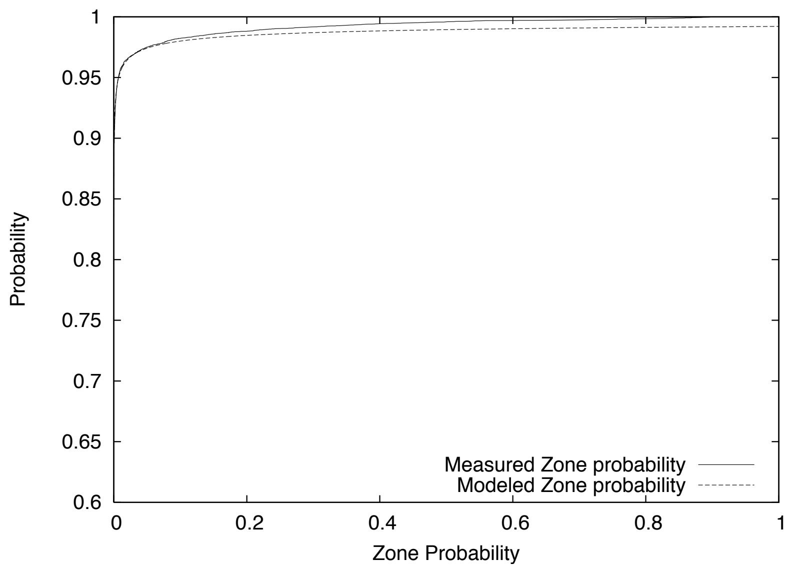

Figure 4.7: CDF of Markov Chain Probabilities in WoW*: This figure shows the modeled log-normal CDF of the probabilities from the measured Markov chains in WoW.*

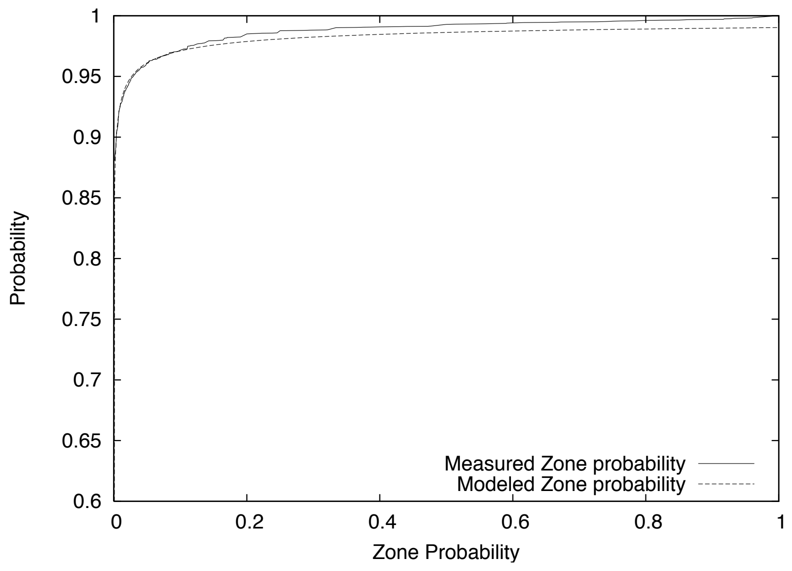

Figure 4.8: CDF of Markov Chain Probabilities in WAR*: This figure shows the modeled log-normal CDF of the probabilities from the measured Markov chains in WAR.*

- How often do players arrive and leave the game?

4.2.1 Population Over Time

We measured the total number of players in the game every 15 minutes during our measurement period and averaged the results by hour for each day of the week. Our hypothesis was that more players were online during evening and weekend hours, due to weekly obligations such as work and school and therefore architectures would need to address these cycles. Figure 4.9 shows the 24 hour daily cycle with each line representing one of the days of the week.

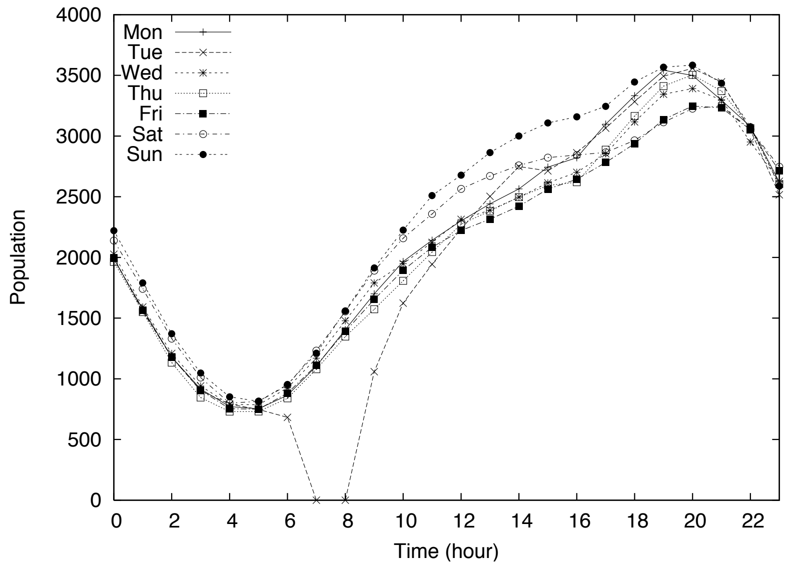

Figure 4.9: Average daily population*: This figure shows the average daily population of the WoW server we measured. Players typically play more in the evenings and both earlier and later on the weekdays. Tuesdays are "patch days", when server maintenance occurs, explaining the empty server at that time.*

We note three important aspects of our graphs. First, populations have an average peak at almost 3600 players. Given the imprecision incurred by the measurement method, we estimate that a typical World of Warcraft server will support up to 4000 players. Second, we see that weekend play stands out from weekday play in that the realm experiences a significantly higher average population earlier in the day. This implies that servers must be provisioned for weekend play. Last, we see an almost 5-fold increase in the number of players from the lowest point (at 4AM) to the highest point (7PM) of the realm population. This implies that servers must also be over-provisioned to handle peak loads during the evenings and are only partially loaded during the early mornings.

4.2.2 Arrival and Departure Rates

To further understand the population fluctuations and to help understand the amount of churn that occurs in a MMORPG, we measured the number of arrivals and number of departures per hour and averaged this again on each day of the week. Figures 4.10 and 4.11 show these results.

In these two figures, we see that the amount of *churn*, or the number of players joining and leaving the game, is high during peak playing times. Figure 4.10 shows similar trends of arrivals during the weekdays, but has an increased arrival rate on the weekends during earlier hours of the day. Figure 4.11 shows that the number of departures increases towards the end of the day. Together, *we see that during peak playing times, over 1,000 players join and leave the game per hour.*

In terms of the magnitude of the difference between minimum and maximum arrival and departure rates, these results show that arrival rates and departure rates differ by a fac-

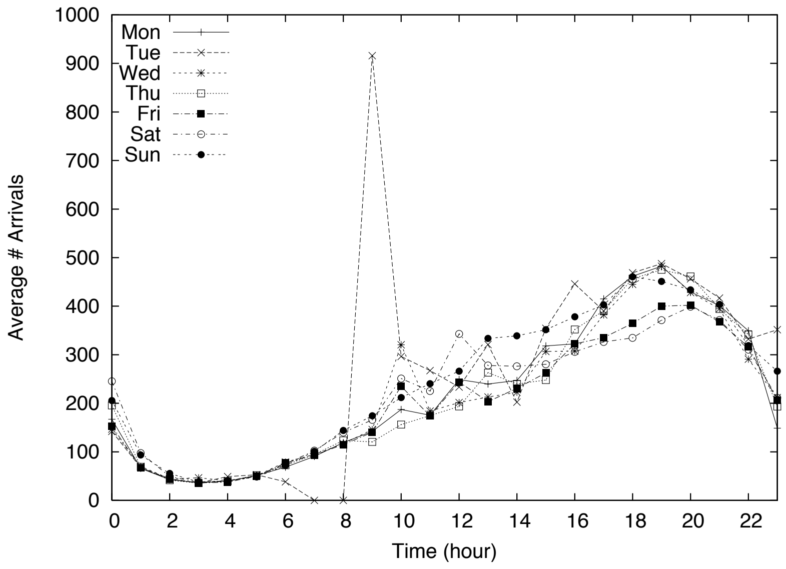

Figure 4.10: Arrival Rates in WoW*: This figure shows the arrival rate in terms of the number of new players seen this hour in WoW.*

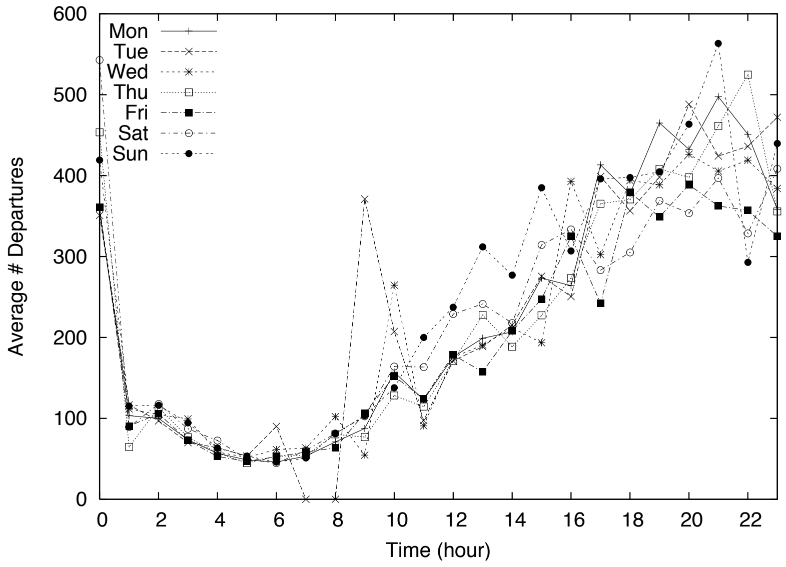

Figure 4.11: Departure Rates in WoW*: This figure shows the departure rate in terms of the number of players seen in the prior hour that are no longer online in WoW.*

tor of 10. In terms of MMO architectures, 1,000 players joining and leaving per hour may not appear to be a huge burden. However, given that WoW claims to have over 10 million subscriptions, a theoretical maximum of 4,000 players per realm indicates that approximately 1 million players log on and off *per hour* of the WoW servers. This is a significant amount of churn that a MMORPG architecture would need to handle. Each time a player logs on or off, there is overhead associated with establishing and releasing connections. In addition, notifications must be sent to all remaining players within the immediate area that the player left or joined the game. Finally, any clients that have registered for updates about that player must be notified.

4.2.3 Session Lengths

Session lengths were measured by adding a random subset of players seen in the most recent snapshot to the *friends list*, allowing us to track how long a character is played in the game. Our measurements in Figures 4.12 and 4.13 show that contrary to anecdotal stories, most sessions were short lived.

Figure 4.12 shows the CDF calculated from all observed session times in WoW. From this figure, we see that only a small percentage of players we observed played for longer than 400 minutes ($\approx$ 8 hours), while most players played for less than 200 minutes ($\approx$ 3 hours). We calculated the mean session time to be 80 minutes, with a maximum observed session time of 1440 minutes (24 hours) and a minimum session time of 1 second. Note that we did not track players for longer than 24 hours, though for future work we will consider how many players were online for extensive periods of time.

Figure 4.13 shows the CDF calculated from all sessions observed in WAR. In WAR, almost 90% of all sessions were less than 200 minutes with a mean session time of 89 minutes, a maximum time of 882 minutes (14 hours), and a minimum session time of 1 second.

For both models, we used the least-squares method to find a fit for a model of the measured data. Similar to what we found with zone populations, given the trend of the data, we determined that a Weibull distribution would fit well. The models are plotted in Figure 4.12 and 4.13.

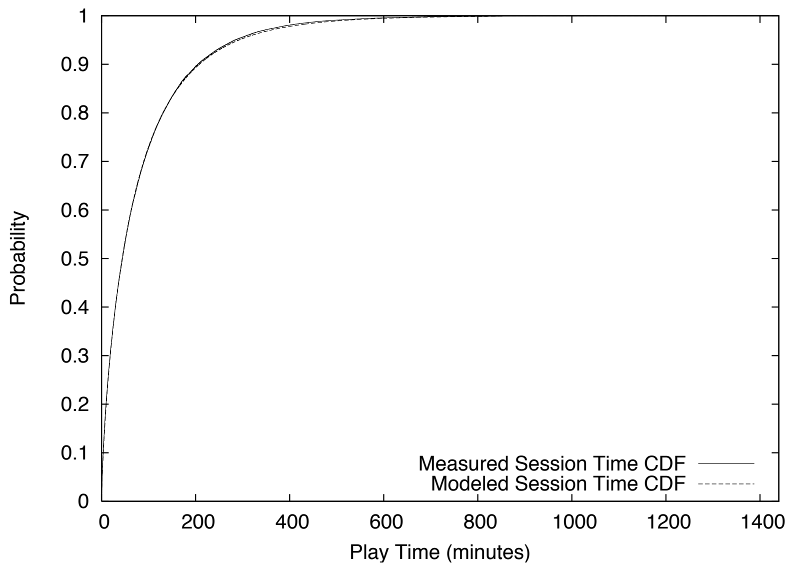

Figure 4.12: WoW Session Times Observed*: This figure shows the CDF of all the sessions we recorded in WoW. In addition, the data was fit to a Weibull distribution, and plotted on the same graph. Both lines in the figure are barely distinguishable due to the close fit of the model.*

Next, we validated our models by plotting the residuals between the predicted and measured values, and found that the residuals for both models had a standard deviation of 0.004, indicating an extremely close fit. Thus, given a uniformly distributed random

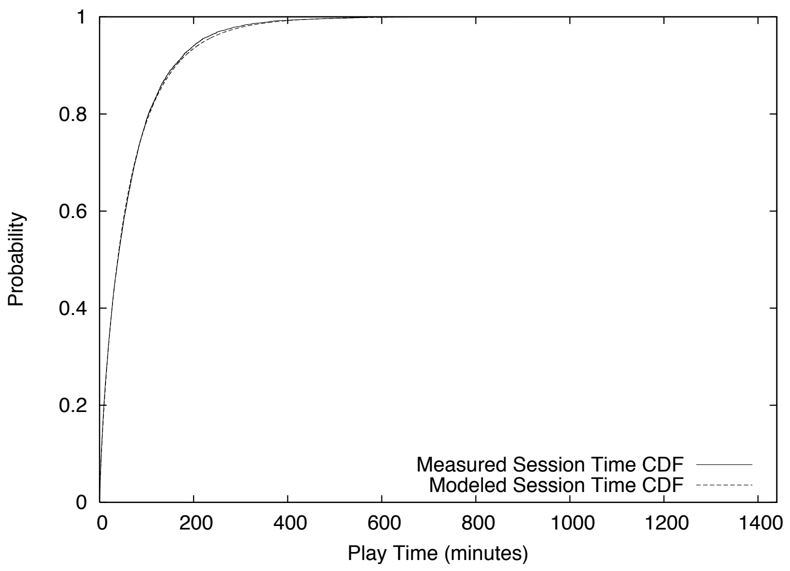

Figure 4.13: WAR Session Times Observed*: This figure shows the CDF of all the sessions recorded in WAR along with the model created by fitting the data to a Weibull distribution. Both lines in the figure are barely distinguishable due to the close fit of the model.*

variable $0 \leq x \leq 1$, the session lengths for WoW and WAR can be modeled as follows:

$$Session_{WoW}[x] = 1 - e^{-(x/69.75)^{0.7522}}, Session_{WAR}[x] = 1 - e^{-(x/59.81)^{0.8322}}$$

This result verifies that MMORPGs experience considerable churn. A large fraction of sessions are short lived while only a small fraction are stable. We believe that what may be happening here is that players may be logging on to check to see if friends or guild members are currently online, checking in-game mail, or checking auctions at the auction house. If this is true, then the implication is that load on an architecture could be reduced by providing an external interface to these services that does not require logging into the game. *Given the predictability of player session time, one may conclude that game developers should target playing experiences for session times that reach the majority of* *players.* Researchers, on the other hand, can use session times to predict how long players will connect to a given architecture.

4.2.4 Time in Zones

We then observed the distribution of time that players spend in any given zone. This information is important because it helps us understand whether players spend an even amount of time in each zone or perhaps spend only a small amount of time in a majority of zones but a large amount of time in one or two zones. Figure 4.14 and 4.15 shows the measured and modeled CDFs of the time players spend in a zone in both WoW and WAR. As with session times, the time in each zone also followed a Weibull distribution, indicating that players did in fact spend most of there time in a few zones, and a small amount of time in the rest of the zones they visited. These distributions are as follows:

$$ZoneTime_{WoW}[p] = 1 - e^{-(x/8.189)^{0.5674}}, ZoneTime_{WAR}[p] = 1 - e^{-(x/24.42)^{0.6669}}$$

As with the previous models, we measured the residuals between the measured and modeled data and found that the standard deviation of the WoW function was 0.009 and for WAR it was 0.002.

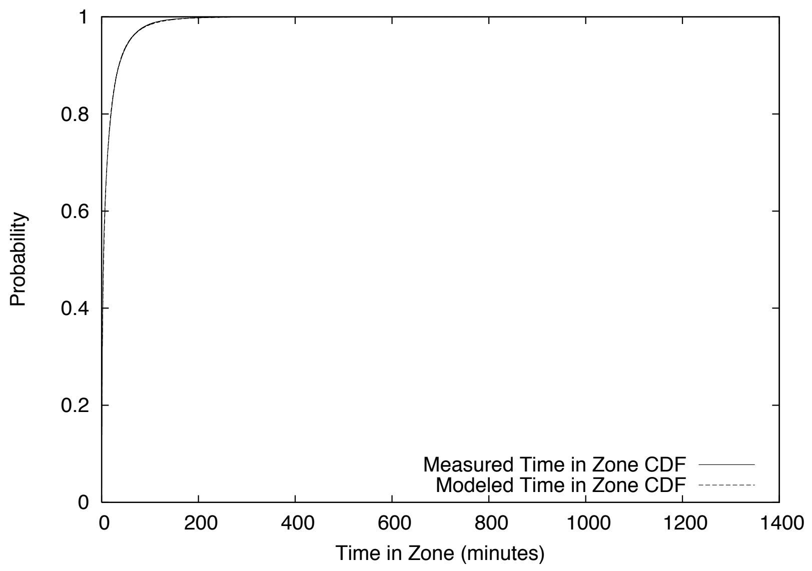

Figure 4.14: Time Spent in a Zone in WoW*: This figure shows the measured and modeled CDFs of the time spent in a zone in WoW.*

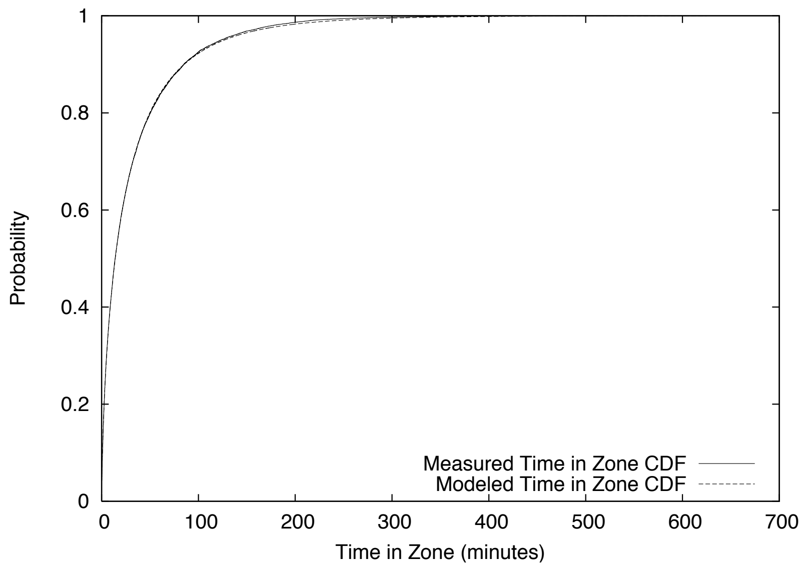

Figure 4.15: Time Spent in a Zone in WAR*: This figure shows the measured and modeled CDF time spent per zone in WAR.*

Chapter 5

Match+Guardian: A Secure Peer-to-Peer Trading Card Game Protocol

Trading card games (TCGs), also known as collectible card games, are a type of card game where players purchase, collect, or trade cards, allowing them to create a playing deck from their collection which can be used to compete with other players. Cards in the game have different features or abilities and may be used strategically and in conjunction with other cards in their deck. An example of the fields used in a trading card are shown in Figure 5.1. Each card has associated abilities, costs, attack power, and defense power. Higher attack power cards can do more damage to another player, but generally have much higher associated costs. Cards with smaller defense values might cost less, but can be destroyed more easily.

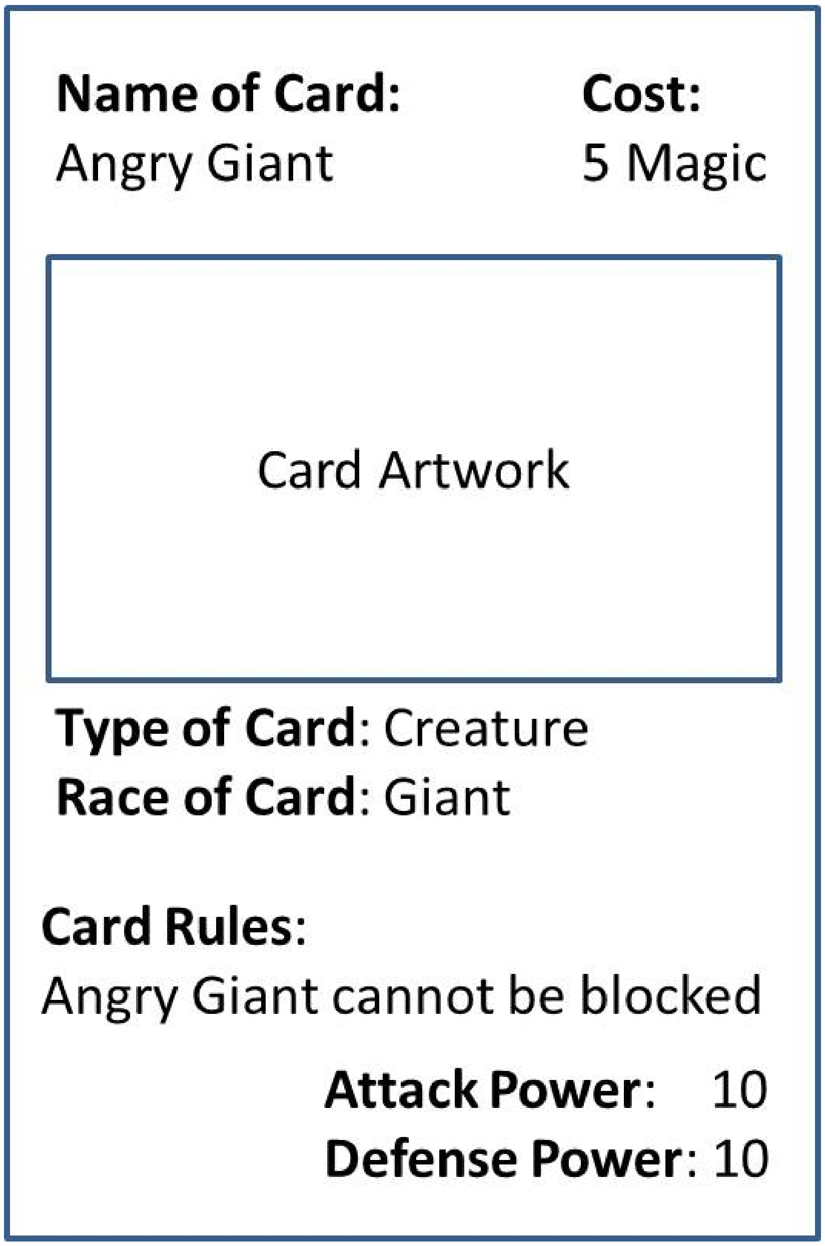

Figure 5.1: An example of the fields in a trading card*: This picture shows some of the example fields present in a trading card*

Recently, TCGs have started to move to the computer game realm and typically use client/server architectures, primarily because a centralized system is needed for handling game transactions and because servers can act as a referee for game play, thereby preventing players from cheating. The question naturally arises: can TCGs be played without cheating in a purely peer-to-peer fashion? We present a protocol showing that this is indeed possible.

In designing a protocol for online TCGs, player expectations must be considered. Two players competing against each other typically have the following expectations:

- They expect to be able to play their TCG wherever and whenever they want
- They expect that the other player cannot easily cheat without being caught
- They expect to be able to play *instant*-type cards, which are cards played in response to an opponent's card(s) and are usually played in rapid succession between players

From these requirements, we have designed a peer-to-peer protocol called Match+Guardian which, unlike a client/server protocol:

- Allows players to play with ad-hoc network connections since they do not rely on Internet connectivity to a server
- Is extremely scalable since servers are not required to arbitrate the games
- Either prevents an opponent from cheating in the game or at minimum detects and provably shows the opponent has cheated
- Provides low-latency interaction, which facilitates TCGs with *instant*-speed cards

Peer-to-peer protocols typically introduce the problem of cheating when used with games and further due to network address translation (NAT), peers may have difficulty connecting to each other. Modern trading card games may in fact consist of several styles of play, including sealed-deck tournament games, draft games, and constructed deck games. We decompose these types of games into a primary sets of actions: from randomly and fairly generating decks of cards, to ensuring that any card played came from a deck that was fixed prior to when play commenced. We then show how to securely perform these steps in a peer-to-peer manner so that all styles of play can be supported. We discuss work in this area in Chapter 2.3.4.

One might assume that the problems in TCGs are exactly those in the game of *mental poker* [SRA81], a fictional game where two people can play poker without seeing each other's cards (e.g., over a phone without a built-in video camera). However, TCGs are different in that the deck of cards is not shared between players, which nicely side-steps the impossibility results of mental poker. Further, TCGs typically have different types of rules of play and therefore require slightly different techniques to prevent cheating.

As with many card games, most TCGs do not have specific time limits for playing individual cards to resolve a turn, except those limits associated with social norms. While players do expect to be able to play *instant* cards quickly in response to another card, in general the timeframe for playing an individual card is longer than that in most other types of games. This gives TCGs some advantage in preventing cheating since certain types of cheats no longer apply when time constraints are not as tightly bound as other genres of games. For example, research in the past has looked at cheating in specific types of games such as role-playing, first-person shooters, and real-time strategy games

[BLL07, CFFS05, GZLM04]. However, this prior work does not address cheating *within* the actual game, such as, by not actually shuffling cards, or choosing your most desired card instead of the next one from the top of your deck. In particular, we see that many of the issues related to cheating in TCGs can be solved by securely and fairly generating random numbers. This is mainly because TCGs rely on random generation of decks and the fair shuffling of those decks for play.

M+G is concerned primarily with preventing or catching cheating within the game play of the TCG. We further propose a more complete architecture with additional services, some of which are necessarily client/server, and include a centralized repository of the entire collection of cards and a centralized store service to securely purchase cards. We describe this architecture in Chapter 6.

For M+G to be viable as a P2P protocol for TCGs, its performance must allow a game to be played at the same pace that players using a physical TCG would play. We compare the performance of our implementation of M+G on the Android$^{\text{TM}}$ platform to real-world measurements of a popular TCG and we also benchmark its performance in terms of latency and time. Our results show that M+G works well on modern mobile devices and would allow players to play P2P TCGs.

5.1 Play Styles

In modern trading card games, there are multiple styles of play. For each style, there are different steps and techniques associated with them. When referring to the different gameplay steps, play styles, and card selection techniques, it is important to be clear on the terminology involved.

5.1.1 Definitions

1. **Universe deck** (D_u) - The set of all cards that exist in the game. This set is defined by the designer of the trading card game.

2. **Base deck** (D_b) - The set of all cards a player owns and is therefore authorized to use during a gameplay session. Note that, $D_b \subseteq D_u$, since any card outside of the *universe deck* cannot exist. Each player has their own *base deck*, determined by their purchases or trades.

3. **Play deck** (D_p) - The set of cards from their *base deck* a player has selected to use during this gameplay session. The *play deck* must be a subset of the *base deck*, thus $D_p \subseteq D_b \subseteq D_u$. The *play deck* is typically constructed ahead of time based on the synergies of using particular cards together.

5.1.2 Styles of Play

In modern trading card games, play styles can be divided into the following common forms:

1. **Sealed deck**, where each player begins with a fresh random set of cards. The random set of cards becomes the player's *base deck* for the duration of the session.

2. **Identical deck**, where each player begins with an identical set of cards. These cards become the player's *base deck* for the duration of the session.

3. **Draft deck**, where each player *drafts*, or picks, cards from a random set of cards. The drafted cards become the player's *base deck* for the duration of the session.

4. **Constructed deck**, where each player has already purchased, collected, or traded cards to create their *base deck* from which a subset of cards are chosen to construct a *play deck*.

Although the gameplay styles of these three forms are unique, the individual steps that compose these styles have significant overlap. By decomposing these styles into discrete securable steps, we can reduce the problem space, allowing us to present a common solution for each kind of problem faced by these different play styles. Note that from these common steps, new gameplay styles can be crafted.

5.1.3 Sealed Deck Play

In a sealed deck game, each player is given an unopened deck of cards which is used to strategically construct a *play deck* prior to the actual match. This set of cards represents the player's *base deck*. Sealed deck games come from tournaments, where the idea is that if the decks are chosen randomly (consisting of some distribution of common, rare and very rare cards), then the matches are more representative of the skills of the players. Beyond this initial step in creating the play deck, sealed deck games are similar to constructed play styles. In the online equivalent of this type of game, a randomly generated deck of k cards (from the entire universe of cards) must be chosen fairly for each player. The securable steps needed to play this game are described in Algorithm 1.

Algorithm 1: Securable Steps Needed to Support Sealed Deck Play

1. Randomly generate a set of cards from the *universe deck* to represent each player's *base deck*. Typically a server would perform the random deck generation, but in a peer-to-peer system, the protocol must handle this step.
2. Have each player select a *play deck* from their *base deck* in a verifiable manner.
3. Draw a card at random from the *play deck*. This simulates the step of shuffling cards.
4. Verify when a card is played, that it came from the set of that players' *play deck*.

5.1.4 Identical Deck Play

In an identical deck game, each player is given a copy of the same deck of cards which is used to strategically construct a *play deck* prior to the actual match. This set of cards represents the player's *base deck*. The identical deck game's origins also come from tournaments, and are a modification on the sealed deck style of play. In this scenario, in addition to testing the player's ability to create an effective *play deck* given a predetermined set of cards, there is the added component of knowing the opponent's *base deck*. This increases the skill required to be effective because the player needs to construct a deck that is effective at countering a known set of cards.

Beyond the initial step of creating the play deck, identical deck games are similar to constructed play styles. In the online equivalent of this type of game, a randomly generated deck of k cards (from the entire universe of cards) must be chosen fairly, and then be given to each player. The securable steps needed to play this game are described in Algorithm 2.

Algorithm 2: Securable Steps Needed to Support Identical Deck Play

1. Randomly generate one set of cards from the *universe deck* to represent the *base deck* from which each player will receive a copy. The only differences between this step in identical deck play and the step in sealed deck play is that only one set of random numbers need be generated, and that after the numbers are generated they can be revealed to all players for use.
2. Have each player select a *play deck* from their *base deck* in a verifiable manner.
3. Draw a card at random from the *play deck*. This simulates the step of shuffling cards.
4. Verify when a card is played that it came from the set of that players' *play deck*.

5.1.5 Draft Deck Play

In draft deck play, each player participates in N *draft steps*. In each draft step, a player starts with *draft deck* consisting of a small number of random cards from the *universe deck*. The player then selects one card from this deck and passes the deck to the next player. This "select and pass" step repeats until all cards are selected, at which point the next draft step starts, except with the order of passing reversed. After all draft steps have completed, each player uses their drafted cards as their *base deck* and then constructs a *play deck* from it. The securable steps needed to play this game are described in Algorithm 3.

5.1.6 Constructed Deck Play

Constructed play is a game where each player creates a *play deck* by strategically choosing a subset of cards from their *base deck* and then plays according to the card game rules. Their *base deck* consists of all cards which they have purchased or traded with other players. Verification that a player owns a particular card can be achieved by verifying the signature

Algorithm 3: Securable Steps Needed to Support Draft Deck Play

1. Randomly generate N sets of cards from the *universe deck* to represent each player's starting *draft deck* each draft round. Note that this problem can be reduced to holding N rounds of the random base deck generation step used in the sealed deck play style.
2. Pass a player's *draft deck* to another player in a verifiable manner. This verification is similar to the verification of a card during gameplay. The main difference is that all cards are verified at once instead of one at a time as they are played.
3. Have each player select a *play deck* from their *base deck* in a verifiable manner.
4. Draw a card at random from the *play deck*. This simulates the step of shuffling cards.
5. Verify when a card is played that it came from the set of that player's *play deck*.

from a purchasing authority. Constructed games represent those games where players or friends compete with each other. The securable steps needed to play this game are described in Algorithm 4.

Algorithm 4: Securable Steps Needed to Support Constructed Deck Play

1. Have each player select a *play deck* from their *base deck* in a verifiable manner. Selecting a *play deck* from a player's collection is a simplification of the problem of selecting a *play deck* from a randomly generated deck, so this problem can be solved similarly.
2. Draw a card at random from the *play deck*. This simulates the step of shuffling cards.
3. When a card is played, verify that it came from the set of that player's *play deck*.

5.1.7 Required Securable Steps for Preventing Cheating in TCGs

Now that we have enumerated the popular play styles and their individual steps, we can create a master list of securable play steps for online trading card games:

- Randomly generate a set of cards from the *universe deck* to represent each player's *base deck* (or *draft deck*).
- Pass a player's *base deck* (or *draft deck*) to another player in a verifiable manner.
- Have each player select a *play deck* from their *base deck* in a verifiable manner.
- Draw a card at random from the *play deck*.
- Verify when a card is played that it came from the set of that player's *play deck*.

With this common securable framework, game designers could mix and match game steps to create completely new styles of play, allowing flexibility in the game style. We acknowledge that while this list may not be completely comprehensive for creating all possible game styles, it may be possible to add new securable steps using similar ideas to those presented here.

5.2 Protocols

For each of the play steps which must be secured, we have developed an appropriate method to ensure that a player cannot cheat in that step. Composing a game from these steps leads to a particular play style. We begin by describing our assumptions, notation, the list of threats we are attempting to prevent, and then detail the protocols individually.

5.2.1 Preliminaries

In order for our protocols to work successfully, we make a few important but reasonable assumptions. First, each player has a unique identifier. Second, the size of the *universe deck* in the TCG is n. Without a loss of generality, we assign the numbers $1...n$ as unique identifiers for each card. Third, we assume that since duplicates of each card in the set of cards can exist in a player's deck of cards, each valid card has a second number from $1...m$. When a player purchases a card from a company, this card first has its unique identifier, and second has this monotonically increasing sequence number ($1...m$) depending on how many of that card the player owns. Cards are then signed by the company with both numbers and with the player's unique identifier to prove that they were purchased legally.

5.2.1.1 Notation

We use the letter U to indicate the entire universe of cards in the trading card game, where $|U|$ is the number of cards in the set. $H(i)$ is the cryptographically secure hash of i while $E_A(i)$ is i encrypted by A (usually Alice in this case). A digital signature of i is noted as $S_A(i)$. Recall that $S_A(i) = E_A(H(i))$. As such, $S_A(i)$ does not reveal any information about i but can be held as proof that i was the value when i is either revealed (i.e., if using a public-key cryptography system) or if the key K_A was revealed with i since $K_A(S_A(i)) = H(i)$ and the cryptographic hash functions are known to all parties.

In certain messages, a $nonce$ is used. A $nonce$ is a one time use randomly generated number that is including during signature creation. The gain in using it is twofold. First, any malicious third party listening to communications between peers will be unable to tell if an identical message is sent, as the $nonce$ will be unique each time. Second, if a third-

Field	**Purpose**
cuid	Card ID: each card must be distinguished from another card by an ID
sn	Sequence number: each duplicate card a player owns has an increasing sequence number
pid	Player ID: each player has a unique identifier, which is attached to the card on purchase
S_c	Company digital signature: each card is signed, after a purchase, by the game company

Table 5.1: *Fields in a digital version of a trading card for M+G.*

party agent tries to replay a message, it can be detected, as no two messages should be identical.

$S_c(cuid, sn, pid)$ indicates a card which is digitally signed by the company (S_c), with its card unique identifier ($cuid$), its purchased sequence number (sn) and the player's unique identifier (pid). sn, the number, is not unique, but the signed tuple $S_c(cuid, sn, pid)$ is a unique tuple. Table 5.1 lists the contents of a digital trading card. Note that the functionality of a card is distributed by the game company with the purchase of a game, so it need not be included.

5.2.2 Threat Model

For our threat model, we assume a typical computationally-bounded adversary, capable of injecting packets and passive listening. We assume standard cryptography will prevent the attacker from decrypting packets in a reasonable amount of time (i.e., before the end of the game). Peer-to-peer TCGs are susceptible to the following types of threats:

1. **Unfairness in Card Selection** - We must be sure that while cards are generated for a player, that the player cannot unfairly influence the outcome of that operation. For example, during the generation of a random deck for a sealed deck game, we must prevent a player from influencing the set of cards generated for the random deck.

2. **Discovery of Private Information** - We must ensure that an opponent cannot garner private data concerning which cards another player has. We must enforce this both during the information exchanges needed in the setup of the game, as well as during the running of a gameplay session.

3. **Changing Cards at Playtime** - As with a real TCG, players cannot be allowed to add or remove cards from their play deck *during* game play. Thus, any played card must be verifiable during game play, to prove that it was actually a card from the player's play deck.

4. **Collusion** - The mechanism developed for generating and verifying cards must be resistant to collusion. Group operations (such as generating a random *base deck*) must be performed in such a way as to mitigate the case where some, but not all, of the players in the game are attempting to influence the outcome.

5. **Replay** - We must be able to prevent an adversary from replaying moves they eavesdropped on so that they cannot fool another player into thinking that the replayed packet is a cheating attempt by the originator of the move.

To verify if specific game rules (such as how cards behave) are violated or not, each player must keep a log of the game. Since each card is identified and digital signatures are used with moves, one can prove if a player cheats by their signed invalid actions during gameplay. However, for a log system to work, an additional sequence number is required for each move in a match so that a total ordering on the TCG moves can be identified.

5.2.3 Securely Generating a Base Deck

We begin by describing how Alice and Bob fairly generate a random *base deck* from a *universe deck* in a pure distribution fashion. This deck forms the basis for providing a sealed deck for the sealed deck game style, and the base decks for the draft play styles. Algorithm 5 describes the process.

Algorithm 5: Algorithm for Securely Generating a Base Deck

1. Alice randomly generates a private number i_A and a public number j_A.
2. Alice signs her private number and only sends the signature $S_A(i_A, \text{nonce})$ to Bob. Recall from our notation that this is simply Alice's digital signature of the tuple (i_A, nonce) and does not include the actual values.
3. Bob randomly generates a private number i_B and a public number j_B.
4. Bob signs his private number and gives $S_B(i_B, \text{nonce})$ to Alice.
5. Alice and Bob exchange the tuples $(j_A, S_A(j_A))$ and $(j_B, S_B(j_B))$. In other words, they exchange their public numbers and sign those numbers (so that they cannot later argue that they gave *different* public numbers).
6. Alice XORs j_B with i_A to create a new random number, k_A, while Bob XORs j_A with i_B to create k_B.
7. The unique identifier of Alice's card from U is $k_A \mod |U|$, while $k_B \mod |U|$ is Bob's first card from U.

At any step, either of the players may refuse to continue. For example, after Alice gives Bob her $S_A(i_A)$ for a particular card, she may wait for Bob's $S_B(j_B)$ in step 5, but not give her $S_A(j_A)$ to Bob. If so, Bob can simply refuse to continue playing as nothing (but time) has been lost. As with the coin-flipping protocol, Alice cannot choose her i_A in such a way so that the resulting k_A is a card that she wants because she does not know j_B before she has encrypted and sent i_A to Bob. The same holds for Bob's choice of j_B—he cannot influence k_A so that Alice gets a poor card from the deck because he cannot identify i_A. Thus, Alice and Bob can fairly and randomly choose a card from U to be part of their tournament deck.

The above sequence of steps can be repeated k times so that each player has a *base deck* of k cards. However, the players can speed up the processes by generating a sequence of private and public numbers. In the first step, Alice generates k private numbers $i_{1A}...i_{kA}$ and public numbers $j_{1A}...j_{kA}$. Bob does the same thing for private and public numbers. In the second and fourth steps, each private number is signed individually (instead of encrypting the entire sequence). The values and nonces are revealed as the cards are played to show that they indeed came from the *base deck* of k cards.

At this point, Alice and Bob have *base decks* consisting of k cards. Figure 5.2 diagrams the flow of steps for random card selection.

5.2.4 Play Deck Creation with a Generated Base Deck

Once a *base deck* of k cards has been generated, Alice and Bob must typically choose a subset of s cards from the *base deck* to form their *play deck*. Note that Alice and Bob choose cards for their *play deck* strategically as certain cards may work better with other cards. Further, the *play deck* does not have a specific size (i.e., Alice and Bob need not

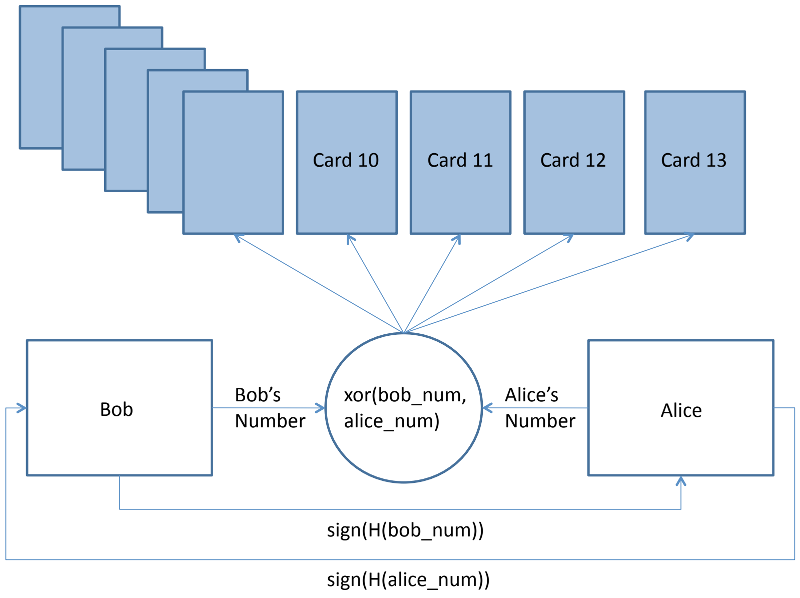

Figure 5.2: Random Card Selection*: This diagram shows how Bob and Alice can participate in random number selection in a verifiable manner while not revealing information about their random number to each other*

have exactly the same sized *play deck*). For strategic reasons they may choose to construct a larger (for more variety) or smaller (for a higher probability of certain cards) deck.

The primary rule for creating the *play deck* is that it must be done entirely before the game begins. One cannot add cards to the *play deck* from the *base deck during* gameplay. Thus, the following steps must occur to fairly choose the *play deck*:

- Alice chooses a card for her *play deck*. Recall that Bob sent her k public numbers for each of the k cards in her *base deck*. Alice sends $S_A(p, S_A(p))$ where p corresponds to the order of the $1...k$ values Bob sent to her. For example, if the card she chose for her *play deck* was selected by XORing her 5th private number with his 5th public number, she sends $S_A(5, S_A(5))$ to Bob. Alice repeats this for every card she adds to her *play deck* from her *base deck*. This prevents Bob from knowing Alice's play deck.

- Bob chooses a card for his *play deck*, sending Alice the tuple $S_B(p, S_B(p))$ for his chosen card, where p is the order of the public values corresponding to the card he chose. Bob repeats this step for every card he adds to his *play deck* from his *base deck*.

Choosing the *play deck* must occur before gameplay begins and both Alice and Bob may create their decks simultaneously, though order does not matter in this case. When Alice or Bob play a card from their *play deck*, they reveal the associated private number and the order value (which they sent to represent each card). For example, when Alice plays the card that was chosen by Bob's 5th public number, Alice sends the tuple $(5, i_5, S_A(5, i_5))$ to Bob. As Bob knows his 5th public value was previously sent the hash of Alice's 5th private value, he can calculate the hash of i_5 to see if it matches the previously sent hash. Also, he can XOR i_5 with his 5th public number to determine the $cuid$ of the card and verify that the correct card was played. Finally, he can verify the hash of $(5, S_A(5))$ against what was sent during the *play deck* creation phase.

A diagram describing the process of selecting a card from the *base deck* is shown in Figure 5.3.

5.2.5 Play Deck Creation with a Non-Generated Base Deck

In styles such as constructed deck play, the *base deck* of the player is not generated randomly and therefore does not need the private number commitment described in Chap-

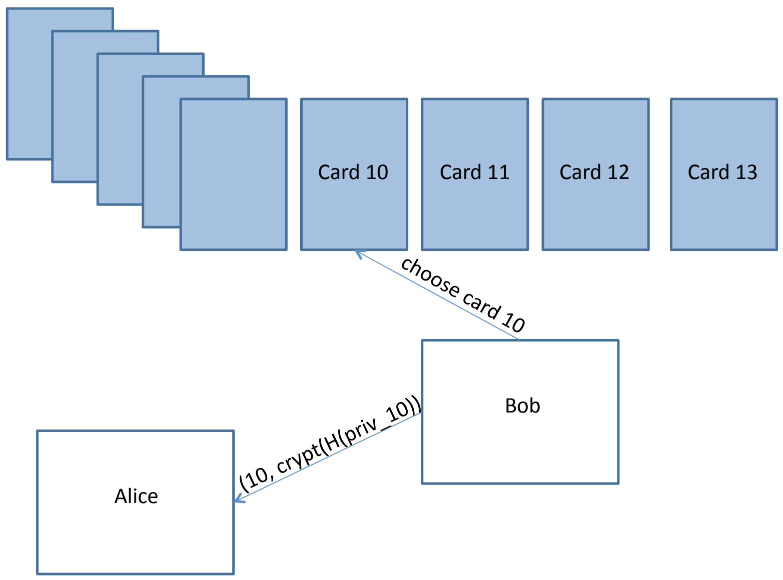

Figure 5.3: Play Deck Selection From Generated Base Deck*: This diagram shows how Alice can be informed what cards Bob is selecting from his generated* base deck *for his* play deck *ahead of time without revealing the value of the card*

ter 5.2.4. Instead, all the player needs to do is send a signed hash of the card id and sequence number of the card they intend to include in their play deck to the other player. For instance, if Alice is sending to Bob that she intends to use the second copy of card 12, she would send $S_A(12, 2, \text{nonce})$ to Bob. This allows Alice to commit to using a card from her *base deck* without revealing what it is to Bob. He must know her *base deck*, however, so that the ordering of cards can be known.

5.2.6 Drawing a Card from the Play Deck

Once a *play deck* has been created, we need to ensure that when a player chooses a card randomly from their deck during gameplay that the card they chose is truly random. In a physical game, the decks are shuffled and opponents may even *cut* the cards to introduce additional randomness. Players in fact typically want their cards shuffled so that they get an even distribution of various types of resource cards while playing as they cannot predict how the game will unfold. However, since the players cannot observe each other during play, we have to ensure that we get the equivalent of a deck shuffle without cheating. The protocol for this scenario is described in Algorithm 6:

Algorithm 6: Protocol for Shuffling Play Deck

1. Alice's *play deck*, consisting of p cards are shuffled. Recall that she has already told Bob which cards are in her *play deck*, and can refer to them by their pth order value.
2. For each card in her deck, Alice sends $S_A(p, \text{nonce})$ to Bob.
3. Bob further shuffles the deck to ensure that Alice shuffled it properly. When Alice needs a card, she simply asks Bob for the next one, which he sends.

Bob repeats the procedure for his *play deck*, so that Alice can ensure his cards are shuffled. When either player plays a card, they can reveal the values (p, nonce) so that the other player can verify that the card is not still in his or her deck, from which cards are being dealt.

5.2.7 Playtime Verification with a Generated Base Deck

Playtime verification of cards from a generated base deck is a two step process. First, the card has to be determined to be in the set of legitimately selected cards. Second, the card has to be verified as a card that exists in the *play deck*.

Since both Alice and Bob know the set U, and hence all the cards have unique identifiers, it suffices for a player to reveal the kth private value with its associated nonce to the other player which the other player can then XOR with the kth public value and match the unique identifier with the card just played. For example, Alice plays a card that was generated from the 5th public number, $(j_{5B}, S_B(j_{5B}))$, which Bob gave to her. When she plays the card, she also sends $(i_5, \text{nonce}_5), S_A(i_5, \text{nonce}_5)$ to Bob. Using i_5 and nonce_5, Bob can check to see if this was the same value that Alice gave to him previously. If not, he knows she is cheating and has proof of it since he already has her hash of $H_A(i_5, \text{nonce}_5)$ that she sent in step 2.

A diagram describing the process of verifying that a card came from a player's randomly generated *base deck* is shown in Figure 5.4.

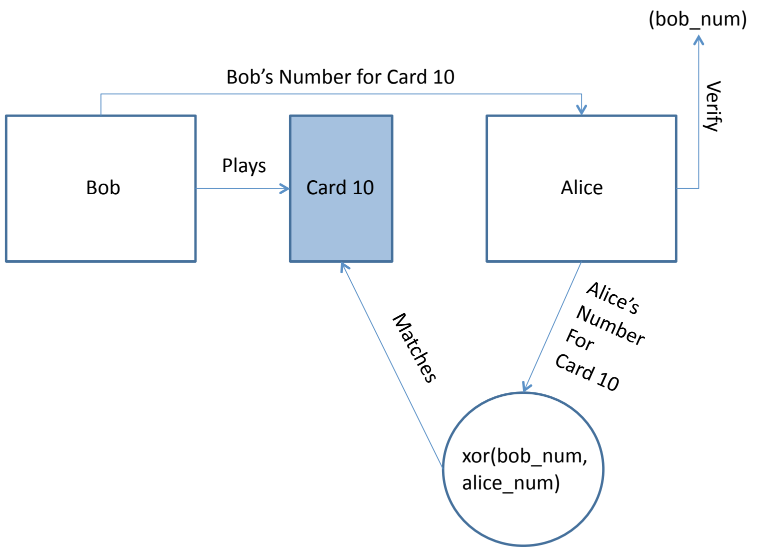

Figure 5.4: Playtime Verification Flow From a Generated Base Deck*: This diagram shows how Alice can verify a card played by Bob is part of the* base deck *that was generated, allowing for real-time verification of correctness in gameplay*

To verify that the card being played comes from the *play deck*, Alice sends the tuple $(p, S_A(p))$ where p indicates the order of the commitment step that declared that card. Since she has already told Bob what order values she was using previously, he can easily verify if she is lying or not about its real value.

5.2.8 Playtime Verification From a Non-Generated Base Deck

In the case of a non-generated *base deck*, playtime verification of cards is a two step process. First, the card has to be verified as existing in the *play deck*. Second, the ownership of the card by the player must be validated.

Since Alice and Bob have both committed earlier the card ID and sequence numbers of the cards they intend to use when declaring their *play deck*, they can simply verify that information. For example, say that Alice plays her first copy of card 11 in her base deck. When she plays the card, she would also send $(11, 1, \text{nonce\_11\_1}), S_A(11, 1, \text{nonce\_11\_1})$ to Bob. He can then verify that the hash computed from the given values matches what was declared by Alice in the *play deck* creation step and that the value has not been seen before.

Once the verification of the play is complete, the purchasing authority's signature can be verified to prove ownership of that card by the player.

5.2.9 Passing a Base Deck to Another Player

When a *base deck* (or *draft deck*) is passed amongst players, it is important to make sure that the deck of cards being passed is not changed. Assuming the decks being passed were generated using the algorithms described above, verification occurs when the private values used to choose the base deck are revealed. Note that this occurs only when using the Draft

Deck or Sealed Deck play styles. For example, Alice who generated a random base deck would reveal all the k private numbers $i_{1A} ... i_{kA}$.

With multiple players, revealing the private numbers for the randomly generated base decks does not leak information, as all players must know all finalized base decks before play begins. Furthermore, a player cannot insert a new card for this exact same reason.

5.3 Evaluation

In order to better understand the performance characteristics of our protocol we have designed a realistic set of success criteria under which M+G must perform. First, we observed people playing Magic the Gathering$^{\text{TM}}$, a popular TCG, and modeled the time it took players to perform certain actions. These results serve as a benchmark to compare the performance of M+G against. Second, we designed a simulation environment using the Android$^{\text{TM}}$ development platform. Given the proliferation of mobile games, and the very likely scenario that someone might be in an environment in which they have access to a mobile device without an Internet connection, we thought this platform appropriate. By comparing the performance of our simulation with real measurements of a popular TCG, we demonstrate that M+G can indeed be used for peer-to-peer trading card games.

5.3.1 Observed Behavior

There is little work done to date in the area of modeling player behavior in TCGs, so we decided to start with the most basic sample set we could: real-life behavior. While observing players, our goal was to focus on the time it took to perform the following activities:

- Draw a card from their deck
- Play a card from their hand
- End their turn

To gather these times, we video-recorded people playing Magic the Gathering^{TM}. After the game was complete, we measured the time of the players' actions to derive a model for comparison with our simulations. The cumulative distribution function of the measured turn lengths is shown in Figure 5.5. From this, we can see that even short turns last close to 20 seconds, while 80% of the turns are between 30 and 140 seconds long.

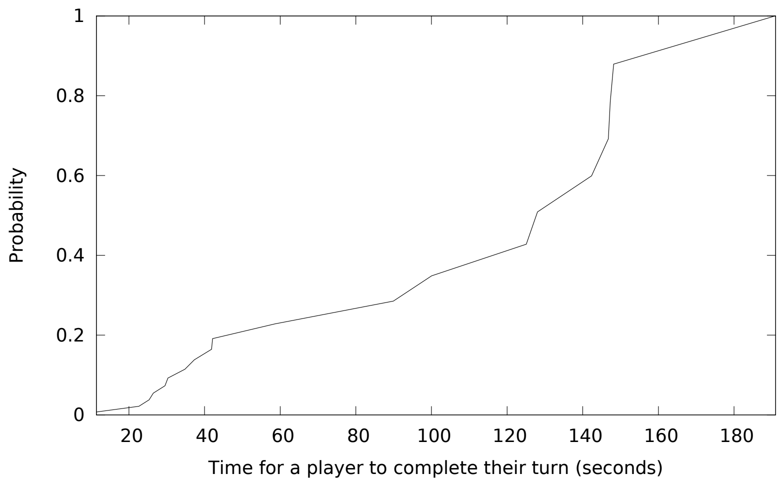

Figure 5.5: Observed Turn Times*: This graph shows the CDF of turn lengths throughout the observed gameplay session.*

Figures 5.6 and 5.7 show the lengths of drawing cards and playing cards. From these figures we see that drawing a card typically ranges from 1 to 4 seconds in duration while playing a card ranges from just over .5 seconds to 5 seconds. Note that the fidelity of measurements were at 1/30th of a second (i.e., we could advance frame by frame), though the margin of error was likely several video frames, due to the subjectiveness of determining when a player was starting or finishing the drawing or playing of a card.

By measuring how long it takes a player to draw and play a card, we can identify the range of time available for our protocol to encode, transmit, and validate a message. As long as we can perform those tasks within the window that is represented by our modeled user behavior, there would be no greater perceived delay to the user than they would expect when picking up a card from a draw deck or taking a card from their hand and placing it onto the table.

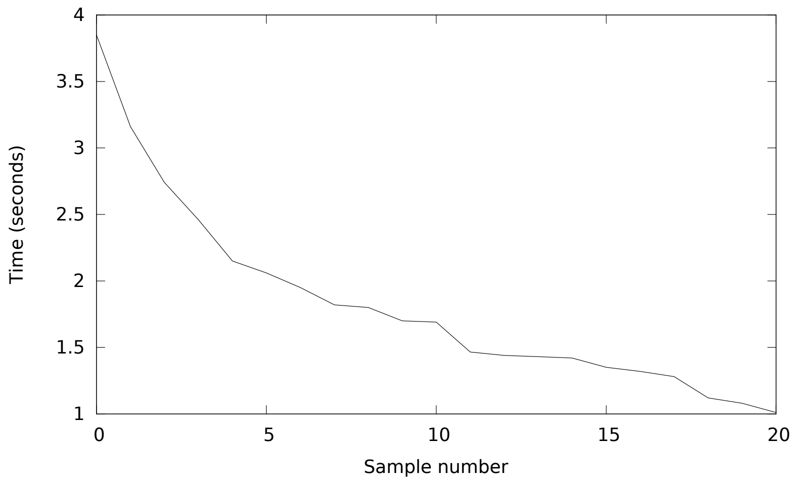

Figure 5.6: Observed Card Drawing Times*: This graph of all measured times for drawing a card from the play deck.*

5.3.2 Simulation

Once we had a set of evaluation criteria we designed a Peer-to-Peer simulation using the Android^{TM} platform to benchmark our protocol. The Android^{TM} environment uses Java

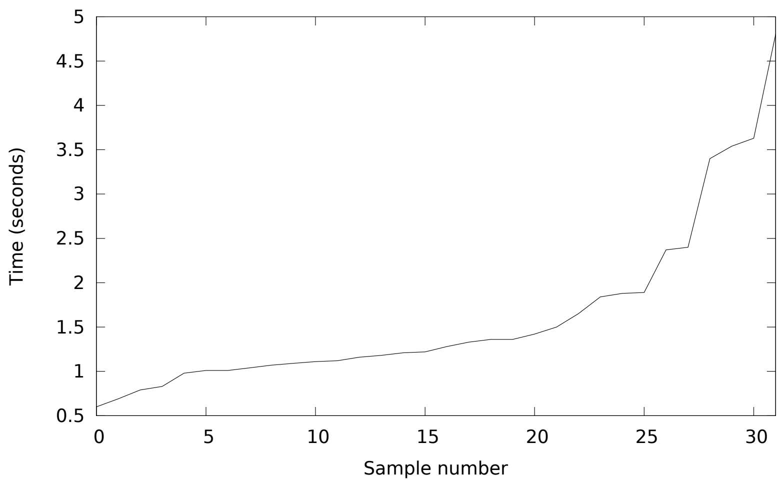

Figure 5.7: Observed Card Playing Times*: The graph of all measured times for playing a single card.*

and has a large user community, which made it ideal for us to develop a reference implementation that can be distributed widely.

The simulation was run under an Android^{TM} emulation environment in which each client's OS reported a BogoMIPS (an artificial benchmark of system performance) of 287.53. By comparison, the Motorola Droid Razr^{TM} reports a BogoMIPS of 1996.68. Therefore, we consider the emulation environment to be representative of a low-end smart phone with worse-case performance characteristics. For asymmetric encryption and signatures we used SHA-256 and RSA with a 1280 bit key. For symmetric encryption we used AES with a 128 bit key.

5.3.3 Results

After running the simulation and processing the video data, we produced combined graphs of two of the performance metrics we were interested in: card draw time and card play time.

In the card draw time graph shown in Figure 5.8 our simulation clearly beats the performance requirements of the observed game, in most cases by over a full second. In particular, 90% of the samples from our simulation were able to complete the card drawing operation in under 1.5 seconds. In comparison, only about 30% of the observed card drawing occurred faster than 1.5 seconds.

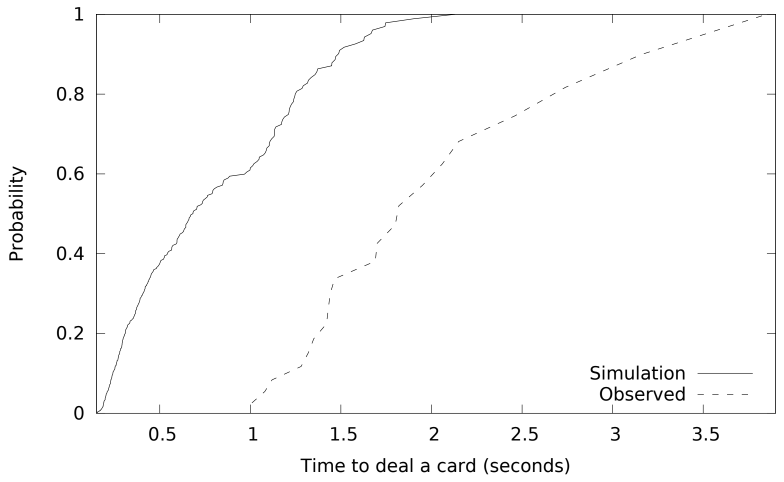

Figure 5.8: Card Draw Time Comparison*: This graph shows a comparison of the distribution of times to draw cards between our simulation and observed play behavior*

In the play time graph displayed in Figure 5.9 once again our simulation was able to outperform the observed behavior, in some cases by several seconds. When players actually played cards, 95% of the time, they took between 1 and 5 seconds. However, M+G was able to complete its operation in 1 second or less approximately 95% of the time.

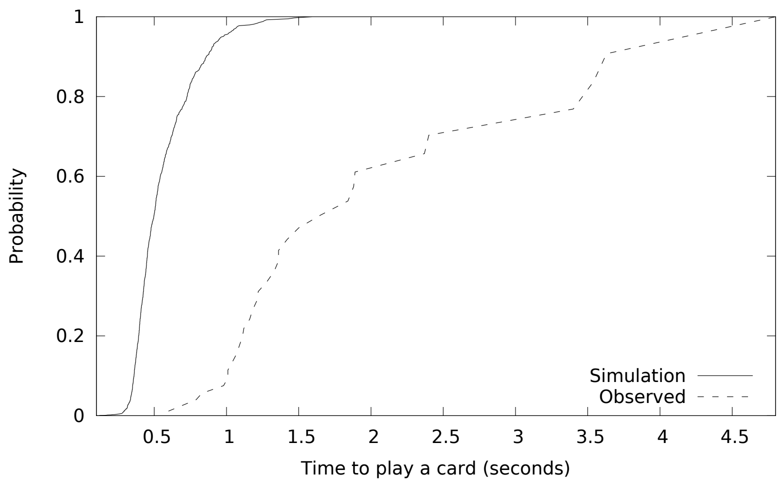

Figure 5.9: Card Play Time Comparison*: This graph shows a comparison of the distribution of times required to play a card*

The results of our analysis is promising and shows that it is possible, given the current state of technology, to implement the encryption required by our protocol while still realizing a performance model that is consistent or better than how players perform in real life. However, we acknowledge that user expectation is hard to judge. In particular, players typically have different expectations of response times when they're reading human cues, such as when physically playing a game, than when they're playing using a computer interface. They might be more willing to wait for an opponent to play a card when they can visibly tell that the other player is thinking of a move. In a networked game, this is not the case.

It is important to note, however, that the cryptography performed in our simulation was done so on very low-end hardware on a mobile device. With a more modern device, the expectation is that the signature and verification operations would run much faster, and perform in a fraction of the time observed in our simulation.

5.3.4 Understanding and Improving M+G's Performance

Although our initial results show a promising picture in terms of performance, our dataset representing real-life behavior is limited. In addition, user perception varies greatly, and delays should always be as short as possible. In order to mitigate these concerns, we need to break down both the communication and computation requirements of the protocol so that we can better understand the critical performance areas in our design. Once we do that, we can present optimizations to our protocol, so that if future data shows an increased demand for performance, we can address that concern.

5.3.4.1 Communication Requirement

During gameplay, a message is sent by the player each time they announce that they are playing a card. That message contains, both, information about the card being played and information about how that card was committed to by the player during the initialization of the game. The response from the opponent is either a positive message indicating the play was accepted or a negative message indicating that cheating was detected and the game will end. Details about the format of the *play card* message in a play style where the *base deck* is generated is shown in Table 5.2.

5.3.4.2 Computation Requirement

Once a request to play a card is received, several verification checks are performed. Algorithm 7 details the steps required to verify a *play card* request as legitimate. If any of the steps fail, a cheat is detected and the game is halted.

Field	**Purpose**
cuid	Card ID: each card must be distinguished from another card by an ID
sn	Sequence number: each duplicate card a player owns has an increasing sequence number
pid	Player ID: each player has a unique identifier, which is attached to the card on purchase
cinfo	Card Information: Vital statistics about the card such as artwork, rules, and description
p	The index of the public number used to generate this card during *base deck* generation
$S_Player(p)$	Player digital signature: the signature by the player sent during the *play deck* commitment for card p
i_p	The private number used to generate this card during *base deck* generation

Table 5.2: Message Format for M+G Play Card Request*: Fields in the play card message.*

Algorithm 7: Steps Necessary to Verify an Opponent's Play Card Request

1. Verify that the hash of $(p, S_Player(p))$ matches what was declared during the *play deck* creation phase

2. Verify that the card has been dealt and has not been played before, by checking that the hash of $(p, S_Player(p))$ exists in the list of dealt cards and has not been used before

3. Verify that $i_p\ XOR\ j_p \mod |U|$, where j_p equals the p^{th} public number created by the player during the *base deck* creation phase, equals the index in $|U|$ of the card attempting to be played

4. Verify that p^{th} signature sent by the opponent during the *base deck* creation phase was generated using the private number i_p being sent

5.3.4.3 M+G Performance Improvements

As stated earlier, verification of cards takes place in real time. That being said, it does *not* mean that the verification need block player action. As a performance improvement, player moves could be accepted at face value when announced and verified in a background execution thread while the game continues. In some styles of gameplay, this asynchronous form of verification would require a *go back N moves* style mechanism to reconcile discrepancies. In M+G, however, the only discrepancy that need be identified is cheating, which halts the game. As a result, there is no need to design a sophisticated go back in time system. If, at any point, a cheat is identified, the game will halt.

In order to optimize the size of messages sent during gameplay, static information, such as a card's rules, artwork, and other vital statistics could be exchanged once, up front, when the player's *base deck* is determined. Matches using play styles such as constructed deck, as stated earlier, must know of the *base deck* of another player so that consistent ordering can be maintained. This optimization would expand that requirement to affect all styles of game play. The benefit to this, though, is that during the initial exchange players can be made aware of any cards that are unknown to them. This would allow for sending just the required information about a card to the other player during a *play card* request. An updated *play card* message format is shown in Table 5.3.

5.3.5 M+G in Mobile Environments

As stated earlier, we chose the Android^{TM} platform for our simulation because we intend for this protocol capable of use in an ad-hoc P2P environments. Instead of the large overlays previous work has relied on, we wanted to look beyond that to today's disconnected but

Field	**Purpose**
cuid	Card ID: each card must be distinguished from another card by an ID
sn	Sequence number: each duplicate card a player owns has an increasing sequence number
pid	Player ID: each player has a unique identifier, which is attached to the card on purchase
p	The index of the public number used to generate this card during *base deck* generation
$S_Player(p)$	Player digital signature: the signature by the player sent during the *play deck* commitment for card p
i_p	The private number used to generate this card during *base deck* generation

Table 5.3: *Fields in an optimized play card message.*

mobile environment. For instance, what if two people waiting at an airport or waiting for a bus, decided they wanted to play a TCG with each other, but one (or both) of them does not have a data connection?

In order for such a scenario to work, the clients must have all of the information and data necessary to validate cards and play the game without being able to check in with a central authority (CA). The design of M+G allows just that.

In the *constructed deck* play style, the information available to the player is the public key of the CA, as well as the signed information about cards the player owns. There is no need for a player to know about all cards in existence since the only cards that matter are the ones being played. Any cards that an opponent might own, that the player does not, will be made aware to them during the play stage of the game. The rule set associated with a card will be sent along with a CA signature, allowing for the player to verify that the CA created this card and the rules are valid.

In the *sealed deck*, *identical deck*, and *draft deck* play styles, information about all cards in the *universe deck* must be known, so the range of the random numbers generated is correct. No validation of ownership is needed, however, since none of the players own the cards being used in these styles. The only signature verification needed in this case is to verify the card being played is a valid card in the game.

Validation Requirement	M+G's Solution
Card Existence	Check the card's CA signature against known CA key
Card Ownership	Check the card's CA's signature of player ID / card ID / sequence number against known CA key
Card Properties and Rules	Check the card's CA signature against known CA key

Table 5.4: Categorization of M+G Validation*: A categorization of M+G's offline validation*

Table 5.4 shows each validation requirement needed for a client to work in a disconnected, ad-hoc P2P manner, and how M+G provides a mechanism for that verification.

Chapter 6

HYPAR-TCG: A Hybrid P2P Architecture for Trading Card Games using Match+Guardian

In this chapter we outline a HYbrid P2P Architecture for Trading Card Games, HYPAR-TCG. To implement and evaluate all aspects of an architecture of this complexity is well beyond the scope of this work. Instead, we will outline the key required capabilities of the system. The core component of HYPAR-TCG is Match+Guardian, a P2P based gameplay protocol introduced in chapter 5. This allows for us to offload all of the computation and messaging costs of running a game to the game clients themselves. We will be introducing other P2P systems, in addition to M+G, necessary to build an entire TCG system. This will let us increase the utilization of the P2P network, and increase the cost savings for the server. Specifically, we will be describing systems for: matchmaking, content caching, player ranking, card purchasing, and card trading.

When designing a hybrid P2P architecture for supporting trading card games, there are four main criteria by which we will measure success:

- *Scalability*: Can our architecture handle a high level of volume and churn in players?
- *Responsiveness*: Can our architecture perform well enough when users are playing the game that perceived delay is within acceptable limits? Does it provide a means to allow clients to connect to each other, regardless of the direct connectivity allowed between them?
- *Market Viability*: Does our architecture have a mechanism to allow users to purchase cards and prove to other players that they own those cards?
- *Security*: Does our architecture protect items bought by players from being duplicated? Is there a secure means for players to trade cards with each other?

In this chapter we present a HYbrid P2P ARchitecture for TCGs (HYPAR-TCG) that address all four of these criteria.

6.1 High Level Architecture

HYPAR-TCG's core is a traditional, centralized server architecture used to support trusted actions such as account management, billing, signing purchased items, and facilitating trade between players. It also serves as the authoritative data store for card vital statistics, player rankings, and other required information.

Supporting the centralized core, however, are two separate P2P overlays built from users currently active in the game. One of the overlays is a distributed implementation of skip lists and is used to facilitate matchmaking amongst peers. The other is a Distributed Hash Table and serves as a caching mechanism for public data recently accessed from the central server. In this capacity, the DHT serves a role very similar to a Level 1 (primary) cache in a modern Central Processing Unit (CPU). A diagram depicting this architecture is shown in Figure 6.1.

Gameplay between peers, as described in Chapter 5.1, is handled via Match+Guardian (M+G), a secure P2P based protocol for TCG gameplay. Details about M+G are shown in Chapter 5.

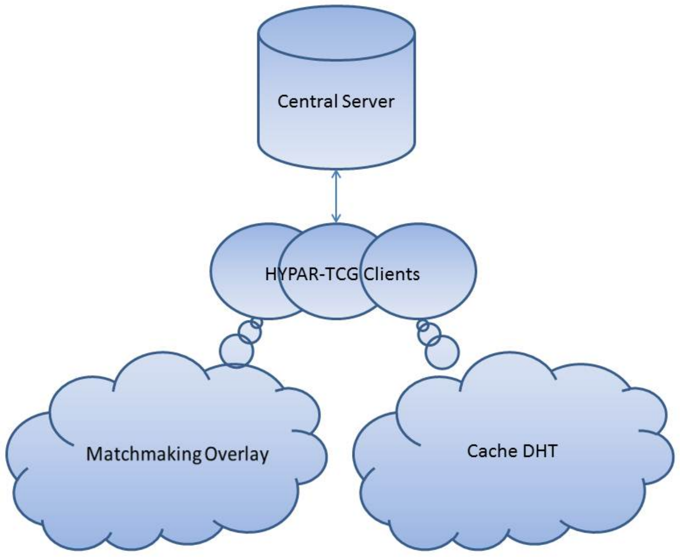

Figure 6.1: High Level Architecture*: This diagram shows a high level picture of how HYPAR-TCG is organized.*

6.2 Central Server Design

In an ideal design, one would have a purely distributed game framework without the need for a central server. This would eliminate all hardware and network costs and would allow game designers to focus on producing content only. Unfortunately, government regulations concerning the protection of private data, not to mention the reaction of credit card companies to the idea that user's account numbers would be stored on uncontrolled machines, prevent this from being a reality. Thus, HYPAR-TCG contains a centralized server in its design.

The job of the central server in HYPAR-TCG involves any player action that requires global connectivity or trusted authority. Table 6.1 breaks down the individual responsibilities of the server.

6.2.1 Central Server as a Signature, Purchasing, and Trade Authority

When a user purchases a card, it contacts the central server to do. At that time, the server creates a record of the sale and provides a signed player ID / card ID / sequence number tuple to the client. The client can use this signed tuple when playing the TCG to prove to other clients that they legitimately own the card. Details involving the verification method of the server generated signature are outlined in Chapter 5.

However, what if the player wishes to trade his card with another player? The central server facilitates this kind of interaction as well. When two players wish to trade a card with each other, they contact the central server. The recipient player in the trade is issued a signed tuple similar to the one issued when purchasing a card. The player releasing control

Responsibility	**Description**
Signature and Purchasing Authority	The server will act as a trusted signature and purchasing authority to allow players to purchase and verify ownership of new cards
Trade Broker	The server will act as a trusted trade broker between players
Master Data Repository	The server will house master copies of player and card data for retrieval and insertion into DHT cache (discussed further in Chapter 6.4)
Player Ratings Calculator	The server will validate and analyse the outcomes of player matches and update player ratings as appropriate
Communications Broker	The server will serve as a trusted broker for network communication between peers, if direct communication between them is not possible

Table 6.1: Roles of the Central Server in HYPAR-TCG*: A list of the responsibilities of the central server.*

of the card receives an updated and signed *revocation list*, a list of server signatures for this player that are no longer valid, from the server. In this capacity, the signature revocation list acts much the same way as a certificate revocation list does in public key infrastructures, and prevents the client from using the now released card in future games.

It is important to note, however, that although the player that traded the card away now has a revocation for it, that does not prevent them from acquiring another copy of the card in the future. The revocation information includes both the card ID *and* the sequence number of the card being revoked. If the player later purchases another copy of the card, it will be issued with a higher sequence number and will not be affected by the revocation.

6.2.1.1 Revocation List Enforcement

During gameplay, players must present a signed version of the revocation list to their opponent so that cards can be verified during play. However, what forces the player to present the *most recent* version of this revocation list? Could the player just present the list originally given to them when they created their account, which contains no cards?

In order to prevent this, we propose the following modifications to the digital trading card and revocation list:

- Add a *latest deck version* (LDV) property to the revocation list. Each time a player purchases or trades a card, a new revocation list is generated with its LDV counter increased by one.
- Add a *deck version* (DV) value to each new card granted to the player equal to the incremented *latest deck version* value of the revocation list.
- Add an expiration date to the revocation list so that opponents can verify that a player is using a recent copy of the list.

A list of the fields in a player's revocation list is shown in Table 6.2. The required fields of a digital trading card needed for validation are shown in Table 6.3.

As an example, consider the trade between player depicted in Figure 6.2. Before the trade, player A owned card 12 sequence number 1, purchased at deck version 3. The player also has a latest deck version of 3. When trading the card, the server issues a new revocation list to player A stating that card 12 sequence number 1, purchased at *deck version* 3 has been revoked, and increases the *latest deck version* of the revocation list to 4. Player B receives

Field	**Purpose**
revoked	Revoked Card List: The card ID, sequence number, and *deck version* of each card that has been revoked by the server
LDV	Latest Deck Version: The latest deck version number issued by the server
expiration	Expiration Date: The date at which this revocation list is no longer considered current

Table 6.2: List of Fields in the Revocation List*: The list of fields present in a server issued revocation list for a player*

Field	**Purpose**
cid	Card ID: The card ID of the card being played
seqnum	Sequence Number: The sequence number of this card. This allows for duplicate cards owned by the player to be distinguished.
dv	Deck Version: The *latest deck version* of the player's revocation list after the card was originally issued. This value does *not* have to match the current *latest deck version* for the player, but does have to be less than or equal to it.

Table 6.3: List of Validation Fields in a Digital Trading Card*: The list of fields needed to validate a player's revocation list against a digital trading card during gameplay*

a new signed message from the server stating that it now owns card 12 sequence number 1 at *deck version* 2, the new *latest deck version* for player B..

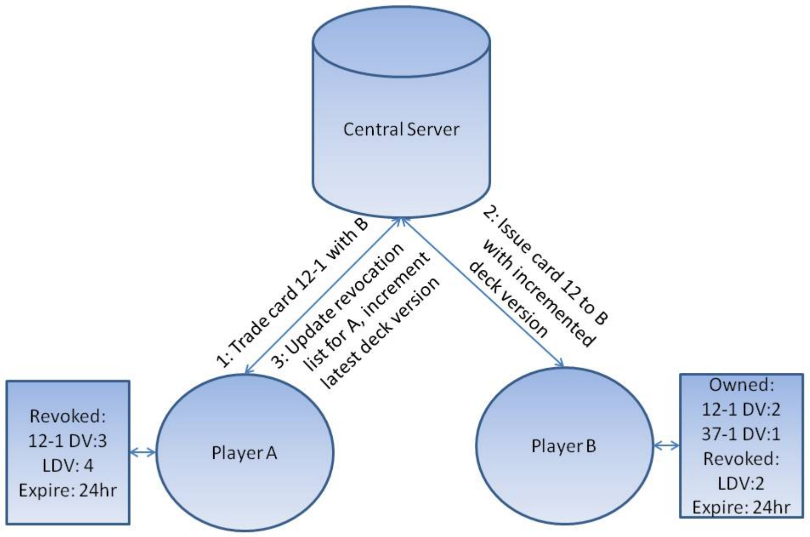
Central Server
1: Trade card 12-1 with B
3: Update revocation list for A, increment latest deck version
2: Issue card 12 to B with incremented deck version
Revoked:
12-1 DV:3
LDV: 4
Expire: 24hr
Player A
Player B
Owned:
12-1 DV:2
37-1 DV:1
Revoked:
LDV:2
Expire: 24hr

Figure 6.2: Trading Example*: This diagram shows an example of two nodes trading a card with each other via the central server.*

With these additions in place for both the trading card and the revocation list, it is now possible to perform comprehensive validation of the ownership of cards a player reveals during play. Verification of revocation lists will take place both at the protocol level during gameplay, and on the server during ratings calculation. Algorithm 8 describes the different validation steps that will take place throughout the system.

6.2.1.2 Server LAM Validation

During LAM validation the server's goal is to prevent clients from being able to take credit for matches won using older versions of their decks, even though they have traded and

Algorithm 8: Steps Necessary to Verify the Revocation List of a Player

1. Before gameplay begins, players exchange revocation lists and sign a *list assertion message* (LAM) to each other verifying the *latest deck version* of the list presented to them. If the *expiration* of the revocation list presented to the player is out of date, a warning will be issued to the player stating that, if they proceed with the match, it might not be rated by the server and the win/loss record might not be recorded.
2. During gameplay, in addition to other validation steps described in Chapter 5.2.7, the opponent checks to make sure that the *deck version* of the card being played is not larger than the *latest deck version* of the revocation list presented to them during LAM creation, and that the card is not on the revocation list.
3. After gameplay is complete, in addition to the list of signed moves that took place during the game, the LAM message is sent to the server. This allows for clients to be able to prove the *LDV* value of the other player, even if that player refuses to upload their match results. The server then checks the periods of time during which both player's *LDV* values were active, and makes a determination if the match is valid and should be rated.

purchased cards since then. It is not the intention of this validation to prevent clients from working in a disconnected state for long periods of time, however. If a player wishes to travel to Antarctica and play TCGs for 3 months in a disconnected state, that is fine, as long as the player does *not* purchase or trade cards while they are using the disconnected client.

The manner in which this validation is enforced by the server involves inspecting the periods of time in which each player's *latest deck value*, as reported by the clients in their LAM messages, was valid. If, at any point in time, all LDV values reported were the active LDV values for each player, the match hypothetically could have taken place and is therefore considered legitimate by the server. If, however, any of the reported LDV values could not have been active during the timeframe in which other player's LDV values were active, one or more of the clients were using an out of date deck in the match. This causes the server to reject the match. The win/loss record, as well as all signed moves reported, will be thrown out.

6.2.2 Player Rankings Calculations

A core component to any competitive game is the ability to impartially rank players according to their performance. HYPAR-TCG will use the World Chess Federation's (FIDE) implementation of the Elo rating system [Elo78] for determining player rankings. The specifics of the system will not be described in detail here, but at a high level, an Elo rating is a single number value that represents the skill of a player, and is computed by analyzing the performance of the player over all of their previous matches. At the end of a match, the amount of points that one player gains and another player loses are determined by the Elo rating of the players involved.

Since the central server is not directly involved in gameplay, it is not able to automatically know the outcome of matches. To solve this problem, we require that clients upload the results of the match to the server so that they can be given credit for the outcome. This mechanism immediately opens the door to cheating, however. What if two players don't agree on the outcome of the match?

Since we are using M+G as the game protocol to play the TCG, cheating by a player when reporting wins and losses can be prevented by submitting the signed moves generated by each player during gameplay. Since the cheating player will be unable to fake the signature of their opponent, they will be unable to successfully create a sequence of messages indicating they won when they did not. As an added bonus, by submitting the sequence of signed messages, the server could allow players to replay matches by downloading and stepping through the messages for themselves.

6.2.3 Central Server as Network Proxy

As stated in Chapter 2.3.4, when communicating in a P2P environment there are several workarounds that allow a majority of peers are able to directly communicate with each other regardless of the network topology between them. In the cases that remain, however, a network proxy is needed that is visible to both peers in order to facilitate communication. In HYPAR-TCG, the central server provides that capability. A high-level picture of how the network proxy will work is presented in Figure 6.3.

In the case that two peers cannot connect with each other, they both contact the central server to ask for a network proxy node to connect with. These proxy nodes need not be very complicated. The computational demand on them will be negligible, since all of the gameplay mechanics are handled on the clients. All that is required of the proxy is to forward packets from one peer to the other. In terms of security to the clients, there is no chance of a man-in-the-middle style attack on the peers. The proxy node is controlled by the central server, and is therefore considered safe.

6.3 Matchmaking P2P Overlay Design

Another key capability for a TCG architecture is the ability for players to find opponents of similar skill that are wanting to play the same style of game. In order to provide this matchmaking capability to the user without incurring network and computation cost to the

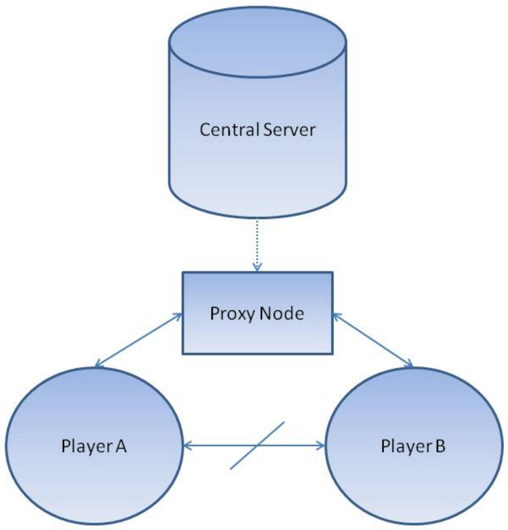

Figure 6.3: Proxy Node Example*: This diagram shows an example of two nodes, unable to communicate with each other directly due to network limitations, using a proxy node provided by the central server as a workaround.*

central server, we propose joining clients together into a SkipNet [HJS+03] distributed overlay.

At a high level, Skipnet is based on the concept of skip-lists, an efficient mechanism for excluding items quickly from searches through ordering nodes by content and then providing pointers to skip over sections of the data. This idea was first introduced by Pugh [Pug90]. As a data structure, a skip list is a sorted linked list in which some nodes have supplemental pointers that "skip" over many elements. The number of nodes that are skipped depends on the entry's height in the list. Height is calculated such that the "height of the i^{th} node is the exponent of the largest power of two that divides i" [HJS+03]. Pointers for nodes at height h skip 2^h nodes, thus allowing for a search time of $O(logN)$.

SkipNet organizes nodes by ordering them based on the longest common prefix of their name into a ring routing infrastructure. In HYPAR-TCG's matchmaking overlay, we propose that nodes are named based on a convention that will group nodes of similar skills and gameplay styles together. This is an ideal solution for a matchmaking system, as the kinds of searches that will be performed in the overlay could then be simplified to looking at adjacent peers in the ring. For instance, if a player named *Bob* of *Master* skill looking to play a *Constructed deck* style match enters the game, they would be added to the matchmaking overlay as *Master/Constructed/Bob*. Once added, Bob could check nodes to the left and right of him in the SkipNet ring to find candidate opponents. A example of this discovery mechanism is shown in Figure 6.4.

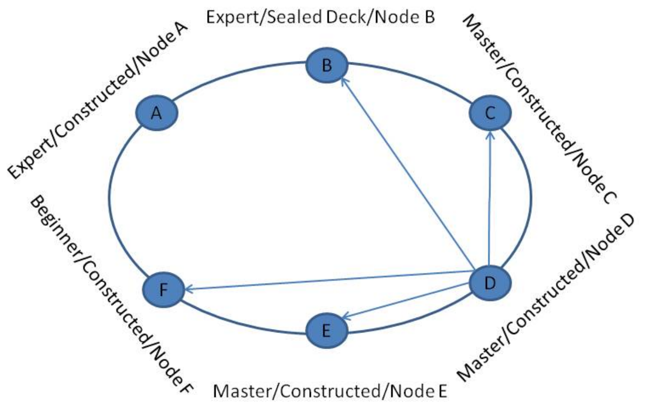
Expert/Sealed Deck/Node B
Expert/Constructed/Node A
Master/Constructed/Node C
Master/Constructed/Node D
Beginner/Constructed/Node F
Master/Constructed/Node E
A
B
C
D
E
F

Figure 6.4: Matchmaking DHT Example*: This diagram shows an example of how a client would identify candidate players in their level of expertise, looking to play the same style of game, in a SkipNet overlay.*

6.4 Caching DHT Design

In order to reduce the network load to the central server from clients retrieving information about the game in HYPAR-TCG, we propose creating a P2P based data cache by joining together clients into a Chord [SMK$^+$01] overlay.

Chord is a system based on the concept of Distributed Hash Tables (DHT)s. DHTs, at a high level, are a class of distributed systems that allow for data lookup using a mechanism similar to a traditional hash table. The advantages to this design is that data is uniformly distributed between peers, allowing for load balancing of data. The average length of a query in Chord is $O(logN)$.

When requesting card or player data from HYPAR-TCG, the client first checks to see if a cached version of the data is available in the Chord overlay. If the data is not available, the client requests a copy of the data from the central server and then inserts that copy into the cache so that subsequent requests do not have to involve the server. An example of this flow is shown in Figure 6.5.

6.4.1 Temporal Locality in the Cache

While some data, such as card artwork, may never change, data such as player rankings do change over time. Although the server copy of this data will be accurate and will reflect all changes to date, the version in the cache can get out of sync. In order to prevent costly pushes of changes into the cache each time data is updated, we propose a system by which cache data expires over time. When a node in Chord receives a request to retrieve a given set of information, it will first check the expiration date for that file. If the information is

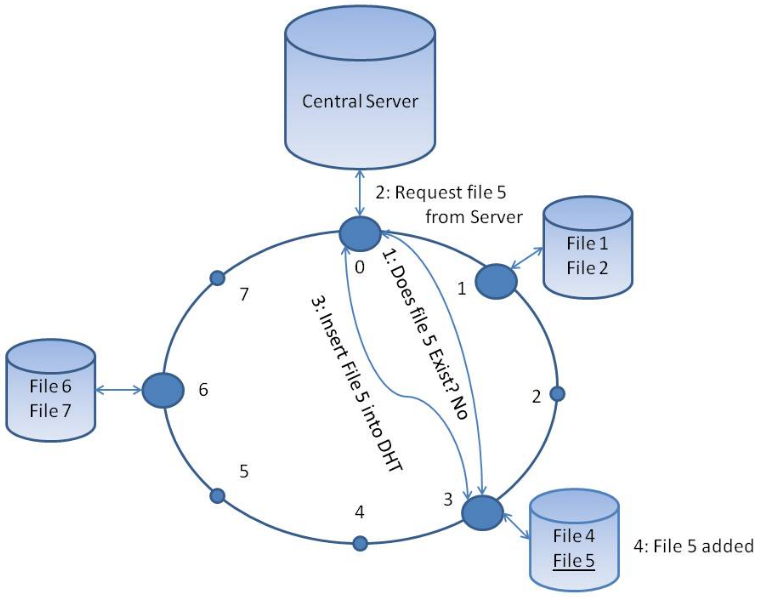

Figure 6.5: Caching DHT Example*: This diagram shows an example flow of a client attempting to find a file in the Chord overlay cache and then, when failing to do so, requesting the file from the central server and inserting it into the DHT.*

out of date, it will be removed and a *not found* response will be sent to the requesting node. The central server will then be contacted, similar to the flow for data not in the cache, and then a new copy will be inserted back into Chord.

6.4.2 Spatial Locality in the Cache

In order to prevent multiple requests to the server for related data about a topic, we propose pre-fetching that data and sending it along with the requested information to the client. For instance, suppose a player is requesting to see information about the last match for a friend. The server can, in addition to sending the match information, also send vital statics about the friend to the client. This information will then be inserted into the cache. If the player then decides to look at the data, it will not have to contact the server again and can instead go directly to the P2P cache for the information.

As a potential improvement, the server could consider including observed player behavior when determining which data is related to a piece of requested information. For instance, consider a scenario where a player, requesting to see information about player A's last match, also requests to see information about the other players in that match 85% of the time. In that case, the server could send information to the client about the other players involved in a match, each time details about a match for a player are requested.

6.5 Centralized vs. HYPAR-TCG System Performance

In order to better understand the motivation for wanting to create a hybrid P2P based system, it is important to quantify the potential server-side savings of the architecture. In order to do this, we need to break down the number of messages and server-side operations performed in both a fully centralized and HYPAR-TCG system. Once that is done, we can compare how those numbers grow as client populations grow. The specific operations that we will be characterizing for our comparison are:

- Purchasing a card
- Trading a card
- Generating cards for players to use in tournament style play
- Shuffling cards for players to use during gameplay
- Broadcasting card draw/play messages and maintaining game state during a match

- Ranking players after a match

When categorizing actions on the server necessary to service client operations, we will be organizing them into three groups: *database operations*, *signature operations*, and *other operations*. When computing the total cost of an action, it is important to account for the relative cost of different kinds of server-side operations. Database operations, for instance, can be I/O intensive. Signature operations, while purely CPU bound, involve many complex math operations and can be equally intensive. By separating messages into these three groups, we can vary the relative cost of each group and allow for us to determine what overhead is tolerable for each kind of operation.

When describing the messages necessary to coordinate actions between the client and the server, we will categorize them into two distinct groups: *signed messages* and *unsigned messages*. All of the messages in both the client/server architecture and the centralized server architecture send the same kinds of information back and forth, such as card IDs and status information. With signed messages, however, a fixed overhead equal to the length of the signature associated with the message is added. Because of this, we want to distinguish signed vs. unsigned messages so that we can vary the overhead percentage associated with the server signature. This will allow us to determine what strength of signatures the system can support without taxing the system too greatly.

In both HYPAR-TCG, and a normal CSA, clients can connect via Secure Socket Layer (SSL) to the central server. This prevents both the client and the server from having to sign or encrypt messages. In HYPAR-TCG, however, clients need to be able to verify ownership of cards with each other during gameplay, without the assistance of the central server. This requires the additional server-side step of signing ownership messages for each card traded or purchased. Thus, *signed messages* are only created and sent during specific steps in HYPAR-TCG, and not at all in a normal CSA.

It is not the claim of this work that a P2P architecture will out-perform a centralized server architecture (CSA) in all types of operations. There are certain kinds of functions (such as trading) that are inherently easier when the central server is solely responsible for enforcing different aspects of gameplay. What we will show, however, is that in the areas that a hybrid P2P architecture *is* able to improve server-side performance, those gains outweigh the increased complexity in other operations.

There is also an increase to client-side responsibilities as part of the hybrid-P2P architecture. As is shown in Chapter 5.3, however, the client is capable of handling this extra responsibility without introducing perceived delay greater than real-life observed behavior.

6.5.1 Purchasing a card

One of the most basic operations a trading card system would have to support is the purchasing of cards to include in a player's deck. Without this capability, clients would have no way in which to acquire cards with which to play with.

6.5.1.1 CSA

In a centralized server architecture (CSA), a client simply sends one message to the server requesting to purchase a card. The server would then update its database to indicate ownership of the card by the player, and a response would be sent back to the client stating that the purchase was successful. An algorithm describing this process is shown in Algorithm 9.

Algorithm 9: CSA Operations and Messages Required When Purchasing a Card

1. *Unsigned Message:* Client sends message to server requesting purchase
2. *Database Operation:* Server updates database to indicate ownership of card
3. *Unsigned Message:* Server responds to client stating purchase was successful

Based on this description, a card purchase involves two unsigned messages and one database operation on the server. Assuming that X players are active in the game, and that each player purchases C cards per hour, unsigned server messages grow according to the formula $2XC$ and server database operations grow by XC per hour.

6.5.1.2 HYPAR-TCG

In HYPAR-TCG, card purchasing requires more steps because of the signatures and revocation list updates involved. A detailed description of these steps is shown in algorithm 10.

In this scenario two unsigned messages, two signed messages, two signature operations, and two database operations are necessary to complete a card purchase. Assuming that X players are active in the game, and that each player purchases C cards per hour, server unsigned messages grow according to the formula $2XC$, server signed messages grow by $2XC$, server signature operations grow by $2XC$, and server database operations grow by $2XC$ per hour.

6.5.2 Trading a Card

Equally as important as the ability to purchase a card in a TCG is the ability to trade that card with another player. For the person receiving the card being traded, the required

Algorithm 10: HYPAR-TCG Operations and Messages Required When Purchasing a Card

1. *Unsigned Message:* Client sends message to server requesting purchase
2. *Database Operation:* Server updates database to indicate ownership of card
3. *Signature Operation:* Server creates signed ownership message indicating player owns the card
4. *Database Operation:* Server updates revocation list LDV for player
5. *Signature Operation:* Server signs revocation list for player
6. *Unsigned Message:* Server responds to client stating purchase was successful
7. *Signed Message:* Server sends signed ownership message to client
8. *Signed Message:* Server sends signed and updated revocation list to client

actions by the server are very similar to those needed during a card purchase. For the person giving the card away, however, additional actions are required.

6.5.2.1 CSA

In a CSA if two players, A and B, are trading a card, there are two different sets of actions that are performed. Both A's and B's inventory must be updated. First, A sends a request to the server to indicate that it is entering a trade transaction with B. Once A agrees to the trade, the server removes it from A's inventory and issues the card to B. The steps involved in issuing a card to B are very similar to purchasing a card. The update for player A is described in Algorithm 11. An algorithm describing the trade for player B is shown in Algorithm 12.

Algorithm 11: CSA Operations and Messages Required for Player A During a Trade

1. *Unsigned Message:* Client sends message to server initiating trade with player B
2. *Database Operation:* Server updates database to remove ownership of card
3. *Unsigned Message:* Server responds to client stating trade was successful

Algorithm 12: CSA Operations and Messages Required for Player B During a Trade

1. *Unsigned Message:* Client sends message to initiate trade with player A
2. *Database Operation:* Server updates database to indicate ownership of card
3. *Unsigned Message:* Server responds to client stating trade was successful

In this case four unsigned messages and two server database operations were required to process the trade. Assuming X players active in the game and D trades are performed in an hour, unsigned server messages grow according to the formula $4XD$ and server database operations grow by $2XD$ per hour.

6.5.2.2 HYPAR-TCG

Similar to a CSA, if players A and B are trading a card, there are two different sets of actions that are performed to update both player's inventories. The update for player A is described in Algorithm 13. An algorithm describing the trade for player B is shown in Algorithm 14.

In this case four unsigned messages, three signed messages, four database operations, and three signature operations were required by the server in HYPAR-TCG to facilitate the trade between players A and B. If X players were active in the game and performed

Algorithm 13: HYPAR-TCG Operations and Messages Required for Player A During a Trade

1. *Unsigned Message:* Client sends message to server initiating trade with player B
2. *Database Operation:* Server updates database to remove ownership of card
3. *Database Operation:* Server updates revocation list, incrementing LDV and revoking traded card
4. *Signature Operation:* Server signs updated revocation list
5. *Unsigned Message:* Server responds to client stating trade was successful
6. *Signed Message:* Server sends updated revocation list to client

Algorithm 14: HYPAR-TCG Operations and Messages Required for Player B During a Trade

1. *Unsigned Message:* Client sends message to initiate trade with player A
2. *Database Operation:* Server updates database to indicate ownership of card
3. *Signature Operation:* Server creates signed ownership message indicating player owns the card
4. *Database Operation:* Server updates revocation list LDV for player
5. *Signature Operation:* Server signs revocation list for player
6. *Unsigned Message:* Server responds to client stating trade was successful
7. *Signed Message:* Server sends signed ownership message to client
8. *Signed Message:* Server sends signed and updated revocation list to client

D trades per hour, signed server messages grow according to the formula $3XD$, unsigned server messages grow by $4XD$, server database operations grow by $4XD$, and server signature operations grow by $3XD$.

6.5.3 Generating Cards

During both *Sealed Deck* and *Identical Deck* game types, a critical part of the match is the generation of random cards for the players to use. In HYPAR-TCG the cost of this random generation is solely the responsibility of the clients, since the cards are generated using a P2P protocol. In a CSA, however, the cards must be generated randomly by the server. In the worst case, *Sealed Deck* play, a unique deck must be generated for each player participating in the game. Algorithm 15 describes the steps required for each client to request a random deck from the server.

Algorithm 15: CSA Operations and Messages Required to Generate a Deck of Cards

1. *Unsigned Message:* Client sends message to server requesting a random *base deck*
2. *Other Operation:* For each card generated, the server computes a random number
3. *Unsigned Message:* Server sends random deck to the client

Two unsigned messages, and other operations equal to the number of cards being generated, are required by the server for each player when generating a random *base deck* each match. Assume that there are X active players in the game and that matches contain, on average M players. Furthermore, assume that P percent of games are participating in a gameplay style requiring generated decks, and the *base deck* of each player consists of B

cards. In that case, if players participate in N matches per hour, unsigned server messages grow by $2PNX/M$, and server other operations grow by $PBNX/M$.

6.5.4 Shuffling Cards

Before starting a match, players shuffle the cards in their *play deck* to ensure fairness in the card drawing process. In HYPAR-TCG, shuffling is handled in a P2P manner and does not incur a server side cost. However, in a CSA, shuffling must be performed on the server. Algorithm 16 describes the process of shuffling a player's *play deck*.

Algorithm 16: CSA Operations and Messages Required to Shuffle a Play Deck

1. *Unsigned Message:* Client indicates to server the *play deck* it will be using for the match
2. *Other Operation:* The server shuffles the play deck
3. *Unsigned Message:* Server sends notification that the deck has been shuffled to the client

Shuffling a deck with B cards has been shown to be possible in $O(B)$ steps [Dur64]. Assume that N matches take place per hour, containing, on average, M players. In that case, $2NM$ unsigned messages and NMB server other operations are required to shuffle a deck.

6.5.5 Maintaining Game State and Broadcasting Messages

During a match, each player must be made aware of the actions of each other player. Depending on the rules of the game, any player action could be interrupted, which means that all clients must have a chance to respond to each player choice. In HYPAR-TCG, player draw and play messages are handled entirely via P2P communication, alleviating the responsibility of the server to broker the messages. In a CSA, however, the server must redeliver each message sent to it during gameplay by any client to each other client in play. The server must also update the server-side state of the game to represent the cards that have been played. Algorithm 17 describes the scenario of a client drawing a card, and Algorithm 18 details the steps required during the playing of a card.

Algorithm 17: CSA Operations and Messages Required For a Client to Draw a Card

1. *Unsigned Message:* Client sends message to server requesting to draw a card
2. *Other Operation:* Server removes a card from the player's *play deck*
3. *Unsigned Message:* Server sends the drawn card to the client
4. *Unsigned Message:* For each other player in the game, a message is sent stating that the player drew a card

Algorithm 18: CSA Operations and Messages Required For a Client to Play a Card

1. *Unsigned Message:* Client sends message to server requesting to play a card
2. *Other Operation:* Server updates state to indicate the card has been played
3. *Unsigned Message:* For each other player in the game, a message is sent stating that the player played a card

For a player to draw a card, one server side other operation and two unsigned messages are required. In addition, an unsigned message to each client participating in the game must be sent. For a player to play a card, one unsigned message is required, plus one server side other operation. In addition, one message to each client participating in the game must be sent indicating the card has been played. Assume that X active players are playing, and on average M players participate in a game. Furthermore, assume that E cards are played each match by every player, and F cards are drawn each match by every player. If players participate in N matches per hour, server unsigned messages for a CSA grow according to the formula $NX(2F + FM + E + EM)/M$. Server other operations grow by NXF/M.

6.5.6 Ranking players

After a match is complete, the server must update the rankings of all players that participated in the match. This is done to reflect the new win and loss records for each involved player.

6.5.6.1 CSA

In a CSA, the win/loss record of the match is already known to the server, since game state was being maintained during the match. All that is required is for the server to perform the rankings calculation, and then to update the ranking record for each player. An algorithm describing this process is shown in Algorithm 19.

Algorithm 19: CSA Operations and Messages Required For Ranking a Match

1. *Other Operation:* Server identifies that the match is complete
2. *Database Operation:* The win/loss rating is updated for each participating player

One server other operation, plus database operations equal to the number of participating players, is required. Assume that X players are active in the game, and that each player participates in N matches containing M players. In that case, the formula for describing the number of server other actions required would be XN. Server database actions would grow according to XMN.

6.5.6.2 HYPAR-TCG

In HYPAR-TCG, the client machines must send the signed outcome of the match to the server, so that it can be validated. The server then will update the win/loss record of each player. Algorithm 20 describes the ranking process.

Algorithm 20: HYPAR-TCG Operations and Messages Required For Ranking a Match

1. *Signed Message:* Each client in the match sends a message to the server stating that the match is complete and who the winner was
2. *Signature Operation:* The server validates the signature of each incoming message
3. *Other Operation:* The server validates the Latest Deck Value (LDV) for each player
4. *Database Operation:* The win/loss rating is updated for each participating player

Recall that validation of all of the moves contained within a reported match will only occur if one or more players refuses to sign the outcome of the match. Therefore, there is not an explicit cost associated with that activity described in Algorithm 20.

Server other, signature, and database operations, equal to the number of participating players, is required. In addition, a signed message for each player is sent to the server. Assume that X players are active in the game, and that each player participates in N matches containing M players. In that case, the formula for describing the number of server other,

	CSA	**HYPAR-TCG**	
Server Action	**Unsigned**	**Unsigned**	**Signed**
Purchase a Card	$2XC$	$2XC$	$2XC$
Trade a Card	$4XD$	$4XD$	$3XD$
Generate Cards	$2PNX/M$	0	0
Shuffle Cards	$2NM$	0	0
Broadcast Actions	$NX(2F + FM + E + EM)/M$	0	0
Ranking Actions	0	0	XMN

Table 6.4: Summary of Server Message Growth*: A summary of the cost of signed and unsigned messages for different server operations between a CSA architecture and HYPAR-TCG*

database, and signature actions required would be XMN. In addition, server signed messages grow by XMN.

6.5.7 Performance Comparison

In order to create a unified model for server-side messaging and action costs, it is necessary to identify all the terms and formulas from each server operation. Table 6.4 lists the associated formulas for each of the server message types. Table 6.5 lists the formulas for each of the server action types. Table 6.6 lists descriptions and values used in the comparison for each unique term contained in the formulas.

With the formulas identified for each of the actions that we will be profiling between a CSA architecture and HYPAR-TCG, we can now introduce combined formulas for messaging and server-side operations. Table 6.7 lists those formulas.

Figures 6.6 and 6.7 compare the costs of actions and messages between a CSA and HYPAR-TCG, as user population grows. In HYPAR-TCG actions, there is an increased number of database accesses, along with higher costs associated with signature operations. However, the frequency of operations that require signatures and database operations in

	CSA		HYPAR-TCG		
Server Action	**DB**	**Other**	**DB**	**Sign**	**Other**
Purchase a Card	XC	0	$2XC$	$2XC$	0
Trade a Card	$2XD$	0	$4XD$	$3XD$	0
Generate Cards	0	$PBNX/M$	0	0	0
Shuffle Cards	0	NMB	0	0	0
Broadcast Messages	0	NXF/M	0	0	0
Ranking Actions	XMN	XN	XMN	XMN	XMN

Table 6.5: Summary of Server Operations Growth*: A summary of the cost of database, signature, and other operations for different server operations between a CSA architecture and HYPAR-TCG*

HYPAR-TCG is not very large. This allows for the relatively high cost of operations to be offset. In terms of messaging, a CSA must be responsible for all aspects of gameplay. This puts a much higher burden on the server, as compared to HYPAR-TCG. Given the values shown in Table 6.6, the number of messages in HYPAR-TCG is reduced by over an order of magnitude, compared to a CSA. In addition, the number of server actions is reduced by 50%.

In order to see at what point HYPAR-TCG stops being effective at reducing server-side action costs, we identified the number of purchases and trades that results in the action performance graph reversing in favor of a CSA. Figure 6.8 demonstrates this scenario. In order for a CSA to perform less server-side actions than HYPAR-TCG, *each* player must purchase at least ≈ 53 cards and trade at least 15 cards **each day**. This would mean that, in one month, every player would own approximately 1590 cards. In Magic the Gathering ™, a popular TCG, there are 10840 unique cards in existence [otC]. At the rate of ≈ 53 purchases a day, every player would own every card in seven months. Given that, it does

Term	**Description**	**Value**	**Value Justification**
X	Number of active players	Varies	This will be the variable term used when plotting the graph
C	Number of card purchases per hour	0.625	15 cards are allocated to a Magic the Gathering$^{\text{TM}}$ booster pack, a common method of acquiring new cards. Assume a purchase rate of one booster pack per day
D	Number of trades performed in an hour	0.125	Trades between players can be performed on individual cards. Assume that three trades occur between players each day
P	Percentage of active games requiring generated decks	.25	Based on equal distribution of gameplay styles listed in Chapter 5.1.2
N	Number of matches players participate in per hour	2	Maximum number of games possible based on observed session length for two players as shown in Chapter 5.3.1.
M	Average number of players in a match	2	Most matches are one-on-one style play
E	Number of cards played by a player during a match	20	Equal to half the Magic The Gathering's$^{\text{TM}}$ tournament deck minimum *play deck* size of 40
F	Number of cards drawn by a player during a match	30	Equal to 75% of Magic The Gathering's$^{\text{TM}}$ tournament deck minimum *play deck* size of 40
B	Number of cards in a player's deck	40	Based on Magic The Gathering's$^{\text{TM}}$ tournament deck minimum size
S	Scale factor for relative cost of *signature operations*	3	Assume that signatures cost three times as much as *other operations*
T	Scale factor for relative cost of *database operations*	5	Assume that database operations cost five times as much as *other operations*
U	Scale factor for signature overhead of *signed messages*	2	Assume that signing messages doubles the message size as compared to *unsigned messages*

Table 6.6: Terms Used in Architecture Comparison*: A description of each term and value*

Model Name	**Formula**
CSA Message Model	$2XC + 4XD + 2PNX/M + 2NM + NX(2F + FM + E + EM)/M$
CSA Action Model	$T(XC + 2XD + XMN) + PBNX/M + NMB + NXF/M + XN$
HYPAR-TCG Message Model	$S(2XC + 3XD + XMN) + 2XC + 4XD$
HYPAR-TCG Action Model	$T(2XC + 4XD + XMN) + U(2XC + 3XD + XMN) + XMN$

Table 6.7: Action and Message Models for a CSA and HYPAR-TCG*: A listing of the message and action models for a Centralized Server Architecture and for HYPAR-TCG*

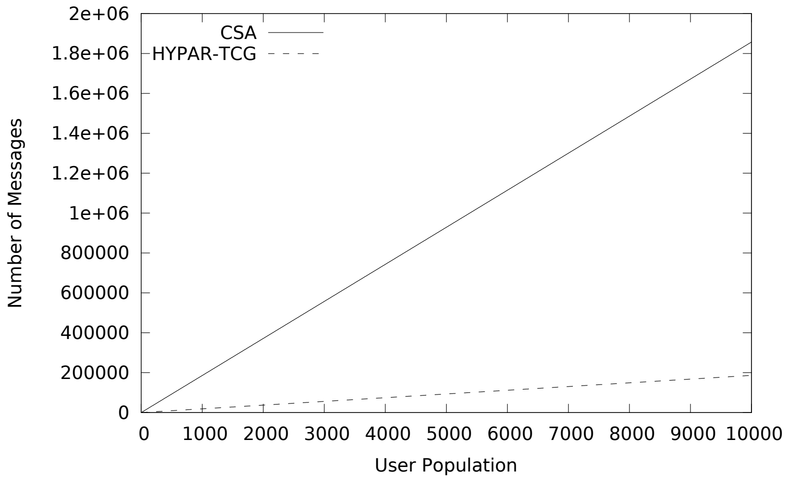

Figure 6.6: Comparison of a CSA and HYPAR-TCG Messaging Costs*: This graph shows a comparison of the messaging costs of both a Centralized Server Architecture and HYPAR-TCG*

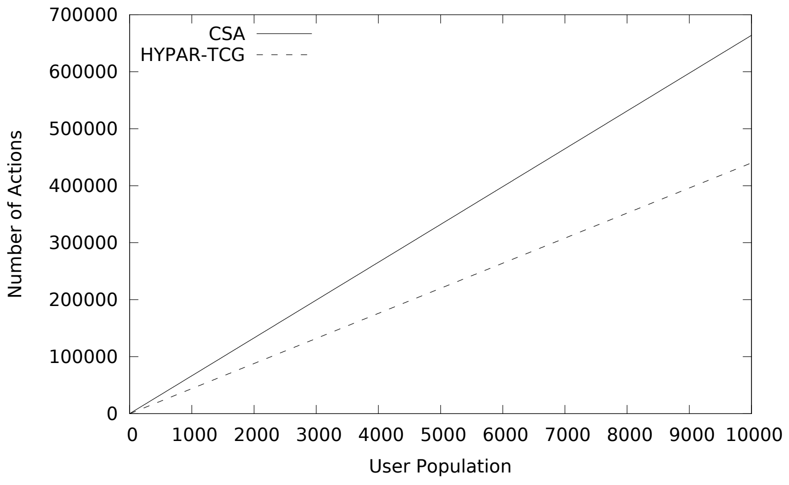

Figure 6.7: Comparison of a CSA and HYPAR-TCG Action Costs*: This graph shows a comparison of the action costs of both a Centralized Server Architecture and HYPAR-TCG*

not seem likely that a TCG would experience that level of purchasing and trading, and that the advantage of HYPAR-TCG over a CSA would hold in real life.

6.6 Effectiveness of HYPAR-TCG as a TCG Architecture

With the design of HYPAR-TCG complete, we can now assess how each of the four success criteria described above are satisfied with this architecture.

- *Scalability*: In terms of scalability, HYPAR-TCG is designed to offload server-side computation and messaging costs through both the use of M+G as the core gameplay protocol, and by redirecting duplicate network requests to the caching Distributed Hash Table (DHT). The caching DHT allows for transferring the responsibility of serving popular content to the clients, which reduces the load and network requirement of the central sever. With a configurable cache lifetime, the architecture can be

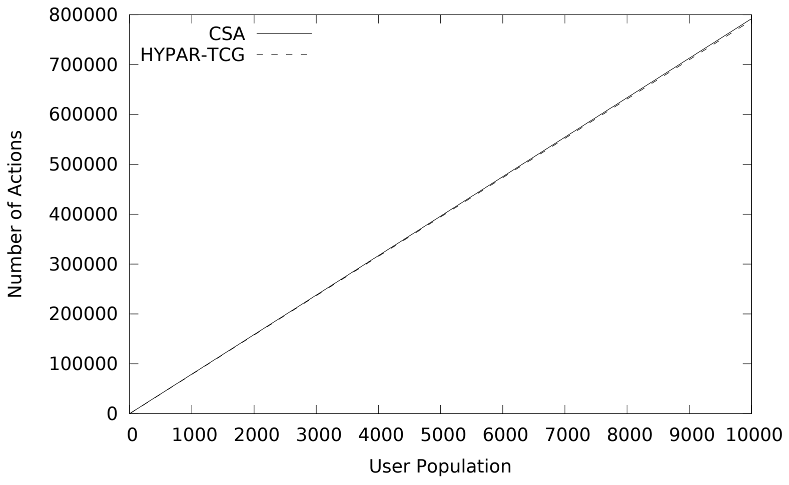

Figure 6.8: Meeting Point of CSA and HYPAR-TCG Action Costs*: This graph shows the point at which the number of purchases and trades in HYPAR-TCG causes the performance of the system to be equal to that of a CSA. Increasing the number of purchases and trades per day would allow a CSA to outperform HYPAR-TCG*

updated as actual usage is identified to create an optimal caching mechanism for data. Furthermore, by creating a system for matchmaking that is entirely handled within a P2P overlay network, we can offload the network cost of that activity entirely from the central server.

- *Responsiveness*: Between existing Network Address Translation (NAT) workarounds and the HYPAR-TCG trusted network proxy, the architecture is able to facilitate P2P interaction between any of its players. By using M+G as the gameplay protocol during matches, perceived user experience is shown to be within acceptable limits.

- *Market Viability*: Through the use of a central billing authority and trusted signatures, HYPAR-TCG is capable of providing an environment where users can purchase items and know that their purchase is safe. It provides safeguards both in terms of their private information and in terms of their ability to prove ownership.

- *Security*: By providing a trusted trade system for players to exchange cards, maintaining revocation lists for card ownership, and preventing multiple trades of the same card within a period of time, HYPAR-TCG is designed to protect player investments while still allowing the freedom of action users would expect from an online TCG.

Chapter 7

Conclusion

Measuring and modeling player movements and interactions in a virtual world are essential to research in architectures for Massively Multiplayer Online Role-Playing Games (MMO-RPG) and other forms of online gameplay. It is our belief that the amount of work in this area is inadequate, and that a great opportunity exists to expand our understanding of virtual behavior inside these games. With this increased understanding will come a more realistic set of models that researchers can use to validate future designs. The models we provided as part of our work represent a first step towards this increased understanding, and significantly improve our understanding of player behavior inside an MMORPG.

In addition, we have demonstrated how, using a set of common Peer-to-Peer protocols, one can support TCGs with different play styles while ensuring cheat-proof play. This common framework for Peer-to-Peer based card games enables the community to develop new styles of play, without having to resolve the common problems facing games of this genre. In order to measure the ability of our Peer-to-Peer protocol for actually playing a

P2P TCG, we implemented M+G in Java using the Android^{TM} development environment and emulated a low-end smart phone. For a benchmark, we measured players playing a real TCG and categorized their actions into playing cards and drawing cards. We then compared the timings measured from our simulation with the observed timings and found that indeed M+G worked well even in a constrained mobile environment.

Finally, we have shown how it was possible to architect a system designed from the ground up to support P2P based TCG gameplay, while still providing centralized governance of authentication and privileged data management. Using multiple P2P overlays for different aspects of server management we propose a system in which players will be able to offload computational and network requirements of the system onto the clients themselves.

7.1 Contribution of the Research

Our research has furthered the understanding of player behavior inside MMORPGs. We provided the first set of behavior models for players *inside* the game through detailed analysis of observations taken in two popular MMOs.

In addition, we have provided the first P2P based architecture optimized for playing TCGs in mobile and disconnected situations. This allows for new environments, such as ad-hoc P2P networks, for implementing games in this genre. Existing work in this area was very limited, and with the introduction of M+G, researchers can begin to create systems in this area.

Finally, We have outlined the first comprehensive architecture for hosting TCGs that uses a hybrid P2P approach. This changes the investment required by game manufacturers in terms of hardware and bandwidth, and allows for offloading CPU and network intensive tasks onto the client.

7.2 Implications of the Research

By gaining a better understanding of behavior inside MMORPGs, researchers can now design simulations and create architectures knowing that the movement and player distribution models they are using to prove their concepts are sound.

M+G and HYPAR-TCG represent a robust P2P environment for hosting many kinds of TCGs. The implications of this contribution are that researchers and game designers can now begin to implement this approach and free the user to play their games whenever and wherever they would like, regardless of their Internet connectivity.

7.2.1 Implications for Future Research

In the area of MMORPG modeling, there are still several areas of player behavior we don't have insight into. Some of the areas include:

- Player to Player interactions (both with non player characters and player characters)
- Intra-zone movement of a player
- Micro-transaction modeling for player purchasing and trading
- Player interactions with objects inside the world

In addition, M+G, although designed for TCGs, has implications far beyond that genre. We believe there is a connection between generating sealed decks for online TCGs and automatically generating the monsters, treasures, and maps for a game level. One could apply these ideas for fairly generating game content between players, especially if those levels are of a competitive nature. The basic principles introduced, such as secure random number generation, content selection, and real-time verification, are concepts that can be applied to many types of gameplay, and there exists a great opportunity to build on this work.

Finally, as future work for HYPAR-TCG, there are several interesting problems, still outstanding, that can help reduce load to the central server. Some of those problems include:

- Determining what cache timeout values optimize the usefulness of the P2P, Chord based cache
- Improving the trading system so that peer trading would be possible
- Investigating if reputation management can be handled securely between peers and then reported to the server, instead of having the server do the calculation

A simulation of HYPAR-TCG would allows us to further refine the computation and messaging scenarios for the architecture. It would also allow us to more accurately refine our centralized to P2P performance comparison model.

7.3 Limitations of Study

We were unable to get server-side logs for the MMORPGs we observed, which forced us to take probing based measurements of players using scripting interfaces made available by the game. With access to logs, it would be possible to verify our findings and expand them to include patterns of behavior that cannot be categorized given the data made publicly available, such as player interactions with objects and other players inside a zone.

In addition, our dataset for observed real-life reaction times in TCGs is limited. Although it was sufficient to give some initial understanding of how our protocol compared to expected behavior, our case could be improved with additional data.

7.4 Final Conclusion

We have provided the first set of models of player behavior inside MMORPGs, created the first P2P based protocol for cheat-proof gameplay of TCGs in mobile and disconnected environments, and outlined a comprehensive architecture for hosting TCGs that utilizes a hybrid P2P based infrastructure.

Our work in all cases has helped address gaps in understanding in the area of P2P based MOGs, and has progressed the field of Computer Science. It is our hope that this work will be found useful, and will be built upon.

Bibliography

[ABMR05] Sudhir Aggarwal, Hemant Banavar, Sarit Mukherjee, and Sampath Rangarajan. Fairness in dead-reckoning based distributed multi-player games. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–10, New York, NY, USA, 2005. ACM.

[AH95] Kiyoharu Aizawa and Thomas S. Huang. Model-based image coding advanced video coding techniques for very low bit-rate applications. *Proceedings of the IEEE*, 83(2):259 –271, feb 1995.

[AS08] Dewan Tanvir Ahmed and Shervin Shirmohammadi. A dynamic area of interest management and collaboration model for p2p mmogs. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, pages 27–34, Washington, DC, USA, 2008. IEEE Computer Society.

[AS09] Dewan Tanvir Ahmed and Shervin Shirmohammadi. Architectural challenges and solutions for peer-to-peer massively multiplayer online games. *SIGMultimedia Rec.*, 1(4):14–15, December 2009.

[BAS04] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 353–366, New York, NY, USA, 2004. ACM.

[BCG08] Carlos Eduardo Bezerra, Fábio R. Cecin, and Cláudio F. R. Geyer. A3: A novel interest management algorithm for distributed simulations of mmogs. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, pages 35–42, Washington, DC, USA, 2008. IEEE Computer Society.

[BDL$^+$08] Ashwin Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, and Xinyu Zhuang. Donnybrook: enabling large-scale, high-speed, peer-to-peer games. *SIGCOMM Comput. Commun. Rev.*, 38(4):389–400, August 2008.

[Bio] BioWare. Warhammer online: Age of reckoning. `http://ageofreckoning.warhammeronline.com/`.

[BL01] Nathaniel E. Baughman and Brian Neil Levine. Cheat-proof playout for centralized and distributed online games. In *Proceedings of IEEE Infocom*, pages 104–113, 2001.

[BLL07] Nathaniel E. Baughman, Marc Liberatore, and Brian Neil Levine. Cheat-proof playout for centralized and peer-to-peer gaming. *IEEE/ACM Trans. Netw.*, 15:1–13, February 2007.

[Blu83] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, January 1983.

[BSK05] Ford Bryan, Pyda Srisuresh, and Dan Kagel. Peer-to-peer communication across network address translators. In *Proceedings of the USENIX Annual Technical Conference*, 2005.

[BT01] P. Bettner and M. Terrano. 1500 archers on a 28.8: Network programming in the Age of Empires and beyond. In *Game Developers Conference*, March 2001.

[CC06] Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, 2006.

[CFFS05] Chris Chambers, Wu-Chang Feng, Wu-Chi Feng, and Debanjan Saha. Mitigating information exposure to cheaters in real-time strategy games. In *Proceedings of ACM NOSSDAV*, pages 7–12, 2005.

[CFJ03] Eric Cronin, Burton Filstrup, and Sugih Jamin. Cheat-proofing dead reckoned multiplayer games. In *International Conference on Application and Development of Computer Games*, January 2003.

[CFSS05] C. Chambers, W. Feng, S. Sahu, and D. Saha. Measurement-based characterization of a collection of online games. In *Proceedings of the Internet Measurement Conference*, 2005.

[CHHL05] Kuan-Ta Chen, Polly Huang, Chun-Ying Huang, and Chin-Laung Lei. Game traffic analysis: an mmorpg perspective. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 19–24, New York, NY, USA, June 2005. ACM.

[cHW98] Soo chul Han and John W. Woods. Adaptive coding of moving objects for very low bit-rates. *IEEE Journal on Selected Areas in Communications*, 16:56–70, 1998.

[CL06] Kuan-Ta Chen and Chin-Laung Lei. Network game design: hints and implications of player interaction. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 17, New York, NY, USA, 2006. ACM.

[CLC99] Wentong Cai, Francis B. S. Lee, and L. Chen. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *PADS '99: Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 82–89, Washington, DC, USA, 1999. IEEE Computer Society.

[CLZ$^+$08] Bin Cheng, Xiuzheng Liu, Zheng Zhang, Hai Jin, Lex Stein, and Xiaofei Liao. Evaluation and optimization of a peer-to-peer video-on-demand system. *J. Syst. Archit.*, 54(7):651–663, July 2008.

[CM06] Alvin Chen and Richard R. Muntz. Peer clustering: a hybrid approach to distributed virtual environments. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, NetGames '06, New York, NY, USA, 2006. ACM.

[CRJ$^+$04] Fabio Reis Cecin, Rodrigo Real, Rafael de Oliveira Jannone, Claudio Fernando Resin Geyer, Marcio Garcia Martins, and Jorge Luis Victoria Barbosa. Freemmg: A scalable and cheat-resistant distribution model for internet games. In *Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '04, pages 83–90, Washington, DC, USA, 2004. IEEE Computer Society.

[DDI$^+$95] Young Francis Day, Serhan Dagtas, Mitsutoshi Iino, Ashfaq Khokhar, and Arif Ghafoor. Spatio-temporal modeling of video data for on-line object-oriented query processing. *Multimedia Computing and Systems, International Conference on*, 0:0098, 1995.

[DG99] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. *IEEE Networks magazine*, 13(4), July/August 1999.

[Dur64] Richard Durstenfeld. Algorithm 235: Random permutation. *Commun. ACM*, 7(7):420–, July 1964.

[Elo78] A. E. Elo. *The rating of chessplayers, past and present*. Batsford, 1978.

[Ent] Blizzard Entertainment. World of warcraft. `http://us.battle.net/wow/en/`.

[FKS+07] S. Fernandes, C. Kamienski, D. Sadok, J. Moreira, and R. Antonello. Traffic analysis beyond this world: the case of second life. In *NOSSDAV '07: Proceedings of the 17th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, IL, USA, June 2007.

[GA06] Assen Golaup and Hamid Aghvami. A multimedia traffic modeling framework for simulation-based performance evaluation studies. *Computer Networks*, 50(12):2071 – 2087, 2006. Network Modelling and Simulation.

[GDS+03] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. *SIGOPS Oper. Syst. Rev.*, 37(5):314–329, October 2003.

[GZLM04] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr. Low latency and cheat-proof event ordering for peer-to-peer games. In *Proceedings of ACM NOSSDAV*, June 2004.

[HJS+03] Nicholas Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *In proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA, March 2003.

[HM94] Rune Hjelsvold and Roger Midtstraum. Modelling and querying video data. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 686–694, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[JPNF10] Herbert Jordan, Radu Prodan, Vlad Nae, and Thomas Fahringer. Dynamic load management for mmogs in distributed environments. In *Proceedings of the 7th ACM international conference on Computing frontiers*, CF '10, pages 337–346, New York, NY, USA, 2010. ACM.

[JZ08] Jared Jardine and Daniel Zappala. A hybrid architecture for massively multiplayer online games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 60–65, New York, NY, USA, 2008. ACM.

[KB07] Patric Kabus and Alejandro P. Buchmann. Design of a cheat-resistant p2p online gaming system. In *Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*, DIMEA '07, pages 113–120, New York, NY, USA, 2007. ACM.

[KC08] James Kinicki and Mark Claypool. Traffic analysis of avatars in second life. In *NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 69–74, New York, NY, USA, May 2008. ACM.

[KCC$^+$05] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi, and E. Yuk. Traffic Characteristics of a Massively Multiplayer Online Role Playing Game. In *Proceedings of ACM NetGames*, 2005.

[KPS09] Jaya Kawale, Aditya Pal, and Jaideep Srivastava. Churn prediction in mmorpgs: A social influence based approach. In *CSE '09: Proceedings of the 2009 IEEE International Conference on Computational Science and Engineering*, pages 423–428, Washington, DC, USA, 2009. IEEE Computer Society.

[LCP$^+$05] Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[LDM04] K. Li, S. Ding, and D. McCreary. Analysis of state exposure control to prevent cheating in online games. In *Proceedings of ACM NOSSDAV*, June 2004.

[LM01] Qilian Liang and Jerry M. Mendel. Modeling mpeg vbr video traffic using type-2 fuzzy logic systems. In Witold Pedrycz, editor, *Granular computing*, pages 367–383. Physica-Verlag GmbH, Heidelberg, Germany, Germany, 2001.

[LOS96] John Z. Li, Tamer Ozsu, and Duane Szafron. Modeling of video spatial relationships in an object database management system. In *IW-MMDBMS '96: Proceedings of the 1996 International Workshop on Multi-Media Database Management Systems (IW-MMDBMS '96)*, page 124, Washington, DC, USA, 1996. IEEE Computer Society.

[LPP94] Aurel Lazar, Giovanni Pacifici, and Dimitrios E. Pendarakis. Modeling video sources for real-time scheduling. *Multimedia Systems*, 1:835–839, 1994.

[LTN$^+$08] Huiguang Liang, Ian Tay, Ming Feng Neo, Wei Tsang Ooi, and Mehul Motani. Avatar mobility in networked virtual environments: Measurements, analysis, and implications. *CoRR*, abs/0807.2328, 2008.

[LZXC06] Leslie S. Liu, Roger Zimmermann, Baoxuan Xiao, and Jon Christen. Partypeer: a p2p massively multiplayer online game. In *Proceedings of the 14th annual ACM international conference on Multimedia*, MULTIMEDIA '06, pages 507–508, New York, NY, USA, 2006. ACM.

[Mau00] Martin Mauve. How to keep a dead man from shooting. In *IDMS '00: Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 199–204, London, UK, 2000. Springer-Verlag.

[MC09] John L. Miller and Jon Crowcroft. Avatar movement in world of warcraft battlegrounds. In *NetGames '09: Proceedings of the 8th ACM SIGCOMM workshop on Network and system support for games*, New York, NY, USA, 2009. ACM.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[MTKE11] Philip Mildner, Tonio Triebel, Stephan Kopf, and Wolfgang Effelsberg. A scalable peer-to-peer-overlay for real-time massively multiplayer online games. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 304–311, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[nap] Napster. `http://www.napster.com`.

[NIP10] Vlad Nae, Alexandru Iosup, and Radu Prodan. Dynamic resource provisioning in massively multiplayer online games. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints), 2010.

[NZA99] Timothy D. Neame, Moshe Zukerman, and Ronald G. Addie. A practical approach for multimedia traffic modeling. In Danny H. K. Tsang and Paul J. Kühn, editors, *Broadband communications*, pages 73–82. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[NZV94] Ralph Neff, Avideh Zakhor, and Martin Vetterli. Very low bit rate video coding using matching pursuits. In *Proceedings of SPIE Conference on Visual Communication and Image Processing (VCIP*, pages 47–60, 1994.

[otC] Wizards of the Coast. Gatherer. `http://gatherer.wizards.com`.

[PG07] Daniel Pittman and Chris GauthierDickey. A measurement study of virtual populations in massively multiplayer online games. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 25–30, New York, NY, USA, 2007. ACM.

[PG10] Daniel Pittman and Chris GauthierDickey. Characterizing virtual populations in massively multiplayer online role-playing games. In *16th ACM International Conference on MultiMedia Modeling (MMM'10)*, pages 87–97, 2010.

[PG11] Daniel Pittman and Chris GauthierDickey. Cheat-proof peer-to-peer trading card games. In *Proceedings of the 10th international Workshop on Network and Operating System Support for Games (NetGames'11)*, 2011.

[PG12] Daniel Pittman and Chris GauthierDickey. Cheat-proof peer-to-peer trading card games. *Accepted With Revision, Special Issue of the ACM/Springer Multimedia Systems Journal*, February 2012.

[PJ00] M. Petkovic and W. Jonker. A framework for video modelling. In *In the Proc. of International Conference on Applied Informatics*, 2000.

[Pug90] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, June 1990.

[PW02a] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 23–29, New York, NY, USA, 2002. ACM.

[PW02b] Lothar Pantel and Lars C. Wolf. On the suitability of dead reckoning schemes for games. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 79–84, New York, NY, USA, 2002. ACM.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[SDM09] Mirko Suznjevic, Ognjen Dobrijevic, and Maja Matijasevic. Mmorpg player actions: Network performance, session patterns and latency requirements analysis. *Multimedia Tools and Applications*, May 2009.

[SGJ07] Travis Schluessler, Stephen Goglin, and Erik Johnson. 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 (NetGames'07)*, pages 1–6, 2007.

[SHLN00] Shahram Shah-Heydari and Tho Le-Ngoc. Mmpp models for multimedia traffic. *Telecommunication Systems*, 15(3-4):273–293, 2000.

[SJLK04] A. Fleming Seay, William J. Jerome, Kevin Sang Lee, and Robert E. Kraut. Project massive: a study of online gaming communities. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1421–1424, New York, NY, USA, 2004. ACM.

[SKR07] Philipp Svoboda, Wolfgang Karner, and Markus Rupp. Traffic analysis and modeling for world of warcraft. In *Communications, 2007. ICC '07. IEEE International Conference on Communications*, pages 1612–1617, June 2007.

[SMK$^+$01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[SRA81] Adi Shamir, Ronald L. Rivest, and Leonard M. Adleman. Mental Poker. *The Mathematical Gardner*, pages 37–43, 1981.

[SSB09] Richard Süselbeck, Gregor Schiele, and Christian Becker. Peer-to-peer support for low-latency massively multiplayer online games in the cloud. In *Proceedings of the 8th Annual Workshop on Network and Systems Support for Games*, NetGames '09, pages 14:1–14:2, Piscataway, NJ, USA, 2009. IEEE Press.

[SVM09] Géza Szabó, András Veres, and Sándor Molnár. On the impacts of human interactions in mmorpg traffic. *Multimedia Tools Appl.*, 45(1-3):133–161, 2009.

[TCH08] Pin-Yun Tarng, Kuan-Ta Chen, and Polly Huang. An analysis of wow players' game hours. In *NetGames '08: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 47–52, New York, NY, USA, October 2008. ACM.

[TKP96] Luis Torres, Murat Kunt, and Fernando Pereira. Second generation video coding schemes and their role in mpeg-4. In *in MPEG-4. European Conference on Multimedia Applications, Services and Techniques*, pages 799–824, 1996.

[VL98] Nuno Vasconcelos and Andrew Lippman. Bayesian modeling of video editing and structure: Semantic features for video summarization and browsing. In *IEEE Intl. Conf. on Image Processing*, pages 153–157, 1998.

[WK04] A. Wierzbicki and T. Kucharski. "p2p scrabble. can p2p games commence?". In *Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on*, pages 100 – 107, aug. 2004.

[WK05] A. Wierzbicki and T. Kucharski. Fair and scalable peer-to-peer games of turns. In *Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on*, volume 1, pages 250 – 256 Vol. 1, july 2005.

[XPi] XPilot. `http://www.xpilot.org`.

[YC06] Min Liu Ye and L. Cheng. System-performance modeling for massively multiplayer online role-playing games. *IBM Systems Journal*, 45(1):45–58, January 2006.

[Yeh08] Chun-Chao Yeh. Secure and verifiable p2p card games. In *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on*, volume 2, pages 344 –349, dec. 2008.

[YR05] Jeff Yan and Brian Randell. A systematic classification of cheating in online games. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, Oct. 2005.

[YV05] Anthony (Peiqun) Yu and Son T. Vuong. Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games.

In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '05, pages 99–104, New York, NY, USA, 2005. ACM.

Appendix A

MMO Observation Data XSD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://mmosim.cs.du.edu/mmoObservations"
    version="1.0"
    xml:lang="en"
    xmlns:tns="http://mmosim.cs.du.edu/mmoObservations"
>

    <element name="playerObservations">
        <complexType>
            <sequence>
                <element
```

```
                    name="playerObservation"
                    type="tns:playerObservation"
                    maxOccurs="unbounded" />
            </sequence>
        </complexType>
    </element>

    <element name="snapshots">
        <complexType>
            <sequence>
                <element
                    name="snapshot"
                    type="tns:snapshot"
                    maxOccurs="unbounded" />
            </sequence>
        </complexType>
    </element>

    <complexType name="snapshot">
        <sequence>
            <element
                name="characterInformation"
                type="tns:characterInformation"
                maxOccurs="unbounded" />
    </sequence>
    <attribute
        name="startTime"
        type="dateTime" />
    <attribute
        name="stopTime"
        type="dateTime" />
</complexType>

<complexType name="characterInformation">
    <sequence>
        <element
            name="playerName"
            type="string" />
        <element
            name="level"
            type="integer" />
        <element
            minOccurs="0"
            name="guild"
            type="string" />
        <element
            name="location"
            type="string" />
            <element
                  name="specialization"
                  type="string" />
            <element
                  minOccurs="0"
                  name="race"
                  type="string" />
            <element
                  minOccurs="0"
                  name="faction"
                  type="string" />
      </sequence>
</complexType>

<complexType name="playerObservation">
      <sequence>
            <element
                  name="playerName"
                  type="string" />
            <element
                  name="observationStart"
                  type="dateTime" />
            <element
                  name="observationStop"
```

```
                type="dateTime" />
            <element
                name="observations"
                type="tns:observations" />
        </sequence>
</complexType>

<complexType name="observations">
        <sequence>
            <element
                name="observation"
                type="tns:observation"
                maxOccurs="unbounded" />
        </sequence>
</complexType>

<complexType name="observation">
        <sequence>
            <element
                name="locationStart"
                type="dateTime" />
            <element
                name="locationStop"
                type="dateTime" />
```

```
                    <element
                         name="location"
                         type="string" />
               </sequence>
          </complexType>
     </schema>
```